Movie Recommendation System Using Collaborative Filtering, Matrix Factorization with Clustering, and Graph-Based PageRank

Lim Han Teng
Mat No.: A0326869J
12-11-2025

Table of Contents

1. Introduction

1.1 Background

Recommender systems have become a crucial component of many known modern platforms, from e-commerce and streaming services to social media. These systems automatically suggest relevant items to users based on their past interactions and preferences. The challenge however that these systems face lies in accurately modelling user to item relationships from typically sparse rating data.

The **MovieLens 100K** dataset is a well-known benchmark for evaluating recommendation algorithms. It provides explicit user ratings on movies, allowing for comparisons across different methods such as collaborative filtering, latent factor models, and graph-based techniques. This project leverages on this dataset to explore and contrast three classical yet complementary approaches to personalized recommendation.

1.2 Project Objectives

The primary aim of this project is to **develop, evaluate, and compare multiple recommender system models** using the MovieLens 100K dataset. Specifically, this project seeks to:

1. **Implement at least two algorithmic classes** covering neighborhood based, latent factor, and graph-based methods, to demonstrate understanding of different recommendation paradigms.

2. **Quantitatively evaluate model performance** using 2 complementary metrics, namely Root Mean Squared Error (RMSE) and Hit Rate@10 for top K recommendation quality.

3. **Discuss limitations and possible improvements,** including potential extensions such as hybridization or alternative loss functions tailored for ranking.

1.3 Overview of Approaches

Three models were implemented to capture different perspectives on user preference learning:

- **Model 1 – Item Based Collaborative Filtering:**
A neighborhood-based baseline that computes item similarity using user co-ratings. Recommendations are generated from items most similar to those a user has rated highly.

- **Model 2 – Matrix Factorization with User Clustering:**
A latent factor model that decomposes the user-item matrix into low-dimensional embeddings, capturing hidden preference patterns. User factors are subsequently clustered to refine predictions.

- **Model 3 – Graph-Based Personalized PageRank:**

A network-centric approach that models the dataset as bipartite user-movie graph. Personalized PageRank propagates influence from a target user node to discover indirectly connected, yet relevant, items.

Collectively, these models allow a comprehensive comparison between neighborhood, latent and graph propagation methods – each addressing the recommendation problem from a distinct standpoint.

2. Dataset Description

2.1 MovieLens 100K Overview

The dataset used in this project is the **MovieLens 100K** dataset, a benchmark provided by GroupLens Research. It consists of **100,000 user ratings** (on a scale of 1-5) for **1,682 movies** from **943 users**. Each user has rated at least 20 movies which gives us sufficient interaction data for collaborative learning.

The dataset includes the three main files used in this project:

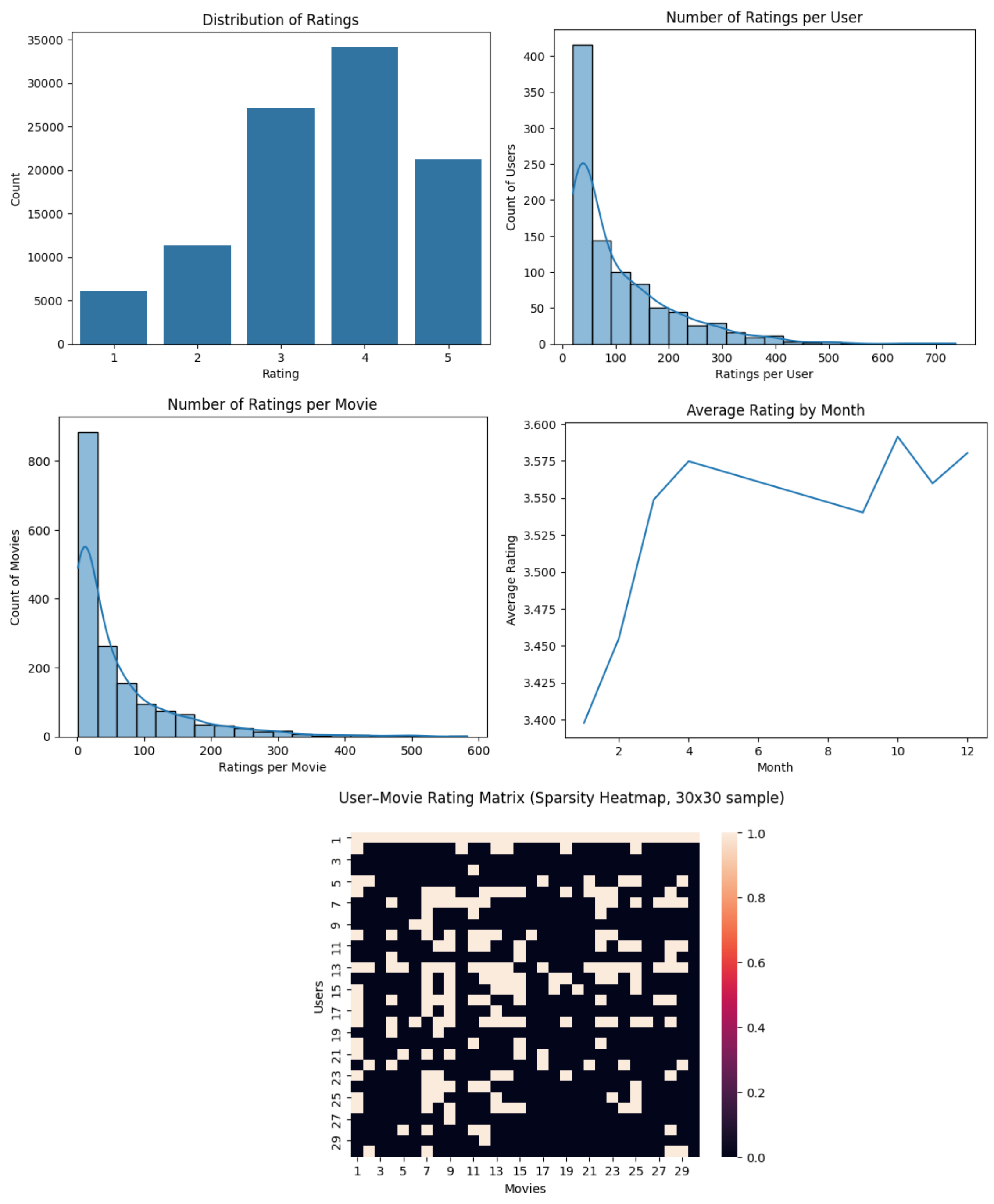
- **u.data** – user-item-ratings-timestamp interactions
- **u.item** – movie metadata such as titles, release dates, and genres (multi-hot encoded)
- **u.user** – user demographic data (age, gender, occupation, ZIP code)

2.2 Exploratory Data Analysis (EDA)

2.2.1 Ratings Dataset

The ratings dataset contains the main user-movie interactions. Each record represents a rating that a specific user gave to a specific move. It has 4 columns:

- **user_id**: unique identifier for user
- **item_id**: unique identifier for movie
- **rating**: rating value (1-5)
- **timestamp**: Unix timestamp of when the rating was given



```
Sparsity of full matrix is 93.70%
```

Figure 1: Ratings Dataset EDA

The figure above presents five key descriptive plots that highlight the characteristics of our ratings dataset, from which we observed:

1. Ratings are positively skewed, with majority of users assigning scores 3 or 4, while very few give 1- or 2- star ratings. This suggests a tendency among users to rate movies favorably, a common bias in explicit feedback data. Consequently, models need to

account for this imbalance, for instance through mean-centering or normalization when computing similarities.

2. The user activity distribution is heavily right skewed – most users have rated fewer than 100 movies, while a small fraction highly active with several hundred ratings. This uneven distribution reflects user sparsity, which can affect collaborative filtering performance. It highlights the importance of regularization in matrix factorization or positive-similarity filtering in neighborhood models to prevent overfitting by highly active users.

3. The number of ratings per movie is also a long-tail distribution. A small set of popular movies receive a large volume of ratings, while majority have only a handful. This popularity bias influences recommender performance. Therefore, it is important to evaluate recommendations using ranking metrics such as Hit Rate @10, which is better at capturing retrieval performance under this imbalance.

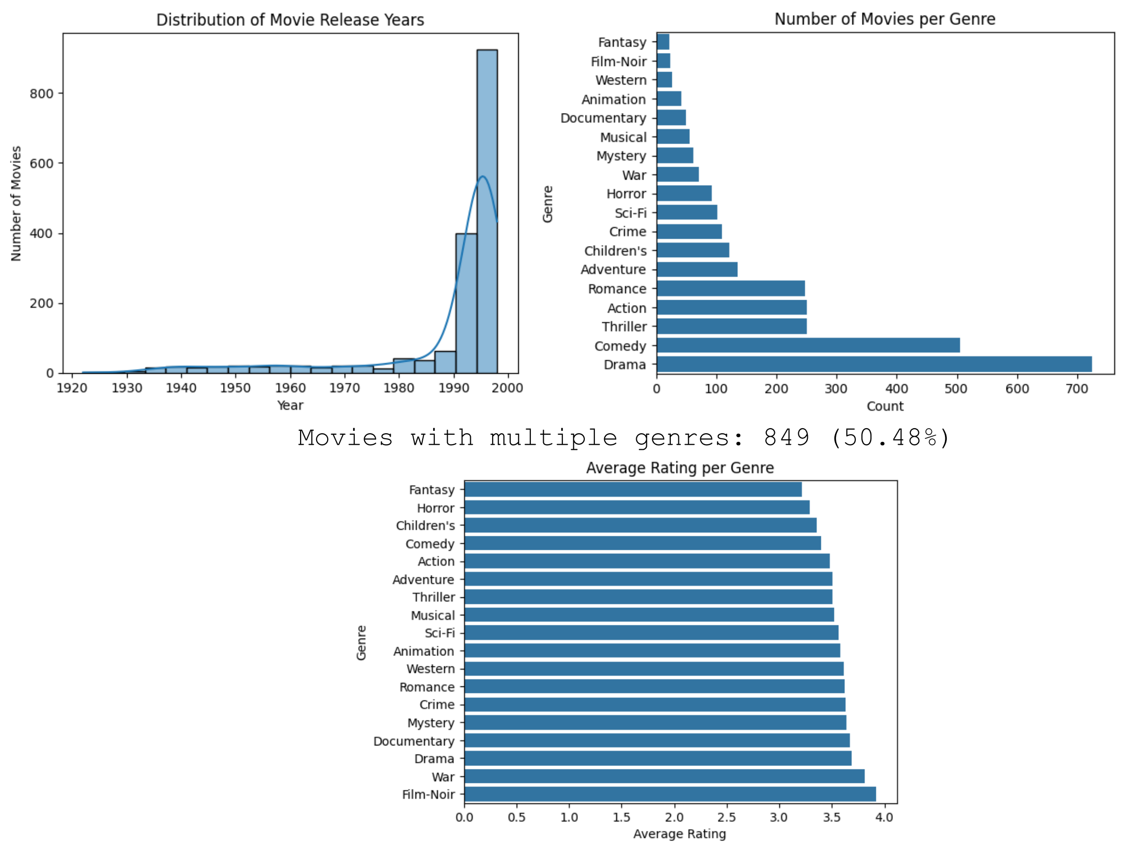
4. The monthly trend of average ratings remains relatively stable, fluctuating only slightly between 3.4 and 3.6. This indicates no significant temporal drift in user preferences over time. As such, static recommendation models are adequate for this dataset, and time-dependent modelling is not required.

5. The user-item ratings matrix is 93.7% sparse. This high level of sparsity is typical of real-world recommendation datasets. However, this level of sparsity poses significant challenges for collaborative filtering models, as there is limited overlap between users' rated items – making it harder to compute reliable similarity scores. Thus, this justifies the choice of 2 other methods in this project – Matrix Factorization & Graph Based Personalization PageRank.

2.2.2 Movies Dataset

The movies dataset contains metadata about each movie. It contains these few columns:

- **movie_id:** unique identifier for the movie
- **title**: movie title
- **release_date**: release date of movie
- 19 binary genre flags (e.g. Action, Comedy, Drama..) indicating movie categories



```
Movies with multiple genres: 849 (50.48%)
```

Figure 2: Movies Dataset EDA

The figure above summarizes 3 key properties of the movies dataset:

1. Most movies were released between 1980s and the late 1990s, with a sharp concentration around the 1995-1997 period. Very few older titles are present. This uneven temporal distribution of movies may influence popularity patterns, as newer releases are more frequently rated by users.

2. The most common genres are Drama, Comedy, Thriller and Action. Less represented genres such as Fantasy, Film – Noir and Western also appear, only sparsely. Approximately, 849 movies (50%) belong to more than one genre.

3. Average ratings across genres ranges between 3.3 and 3.9, showing moderate variation. Genres such as Fantasy, Horror and Children’s tend to receive slightly lower average ratings, while Film-Noir, War and Drama are rated lower on average.

2.2.3 Users Dataset

The **users** dataset provides demographic information about each user. It has 5 columns:

- **user_id**: unique user identifier
- **age**: user's age
- **gender**: 'M' or 'F'

- **occupation**: user's occupation category
- **zip_code**: user's postal code (not relevant for analysis)

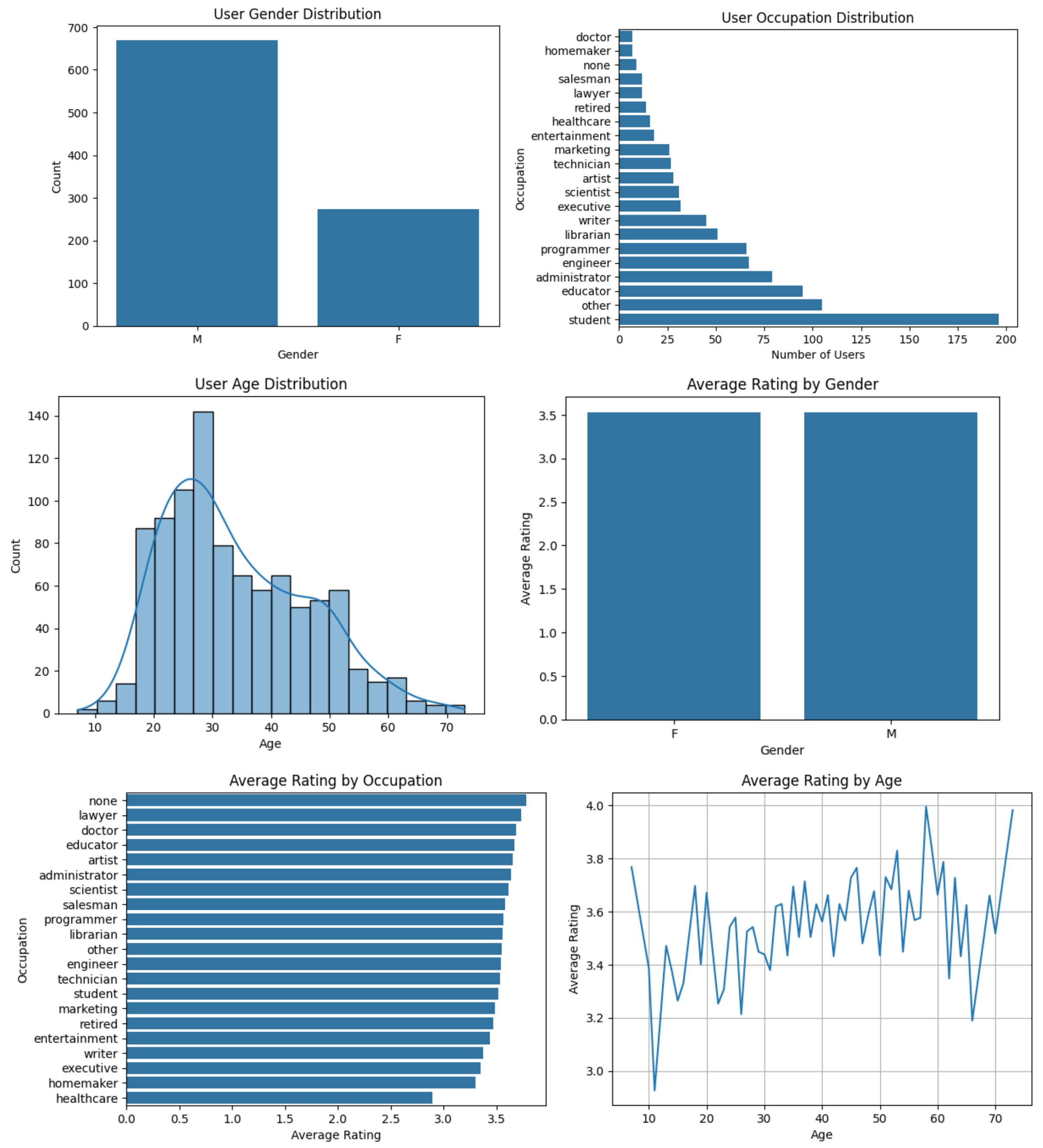


Figure 3: Users Dataset EDA

The figure above presents 6 demographic characteristics of the users dataset, together with an analysis of how different groups vary in average movie ratings, from which we observed:

1. The dataset contains substantially more male users than female users. This imbalance reflects the original MovieLens user base and may influence genre preferences.

2. Users come from diverse range of occupations, with students, engineers and other/unspecified categories being the most common. Less represented occupations such as doctors, lawyers and homemakers contribute only a small portion of the user base. The broad spread across occupations provided limited useful demographic variation.

3. User ages range from early teens to over 70, with a strong concentration around 20-35 age group. This aligns with the demographic expected on early movie-rating platforms. The distribution is right-skewed, with fewer older users.

4. Both male and female users show nearly identical average rating behavior, with mean scores around 3.5. Thus, gender does not introduce a systematic rating bias in the dataset.

5. Mean ratings across occupation vary only slights, falling around roughly between 3.1 and 3.6. Although minor differences exist, these variations are not large enough to strongly influence our model's performance.

6. The average rating by age fluctuates modestly, generally staying within 3.2 – 3.8 range. There is no strong monotonic pattern, although some older age groups appear to give slightly higher ratings. Overall, age does not show a decisive influence on rating behavior.

3. Experimental Setup

3.1 Train–Test Split (Leave-One-Out)

To evaluate recommendation performance in a realistic setting, a **Temporal Leave-One-Out (LOO)** strategy was used for each user:

1. For each user, ratings were first sorted chronologically by timestamp.

2. The most recent ratings for each user were assigned to the test set. All earlier ratings for that user were assigned to the training set.

```
Train set size: 99057
 Test set size: 943

 Users in train: 943
 Users in test: 943
```

The model is always trained on the past and evaluated on the user's most recent interaction, reducing temporal leakage.

As some movies may appear only in the test set (i.e., rated by only one user, and that rating happens to be held-out interaction). Such items cannot be scored by the models, because they do not appear in the training data.

To avoid this item cold-start problem, the test was filtered to keep only rows whose item_id also appears in the training set. After filtering, the **final test size is 940 interactions (3 less)**, and all test movies are guaranteed to have been seen during training.

```
Filtered test set size: 940
```

This setup closely mirrors a realistic scenario where a recommender must predict a user's next movie choice given their historical viewing history.

3.2 Evaluation Metrics (RMSE, Hit Rate@10)

Two complementary metrics were used:

Root Mean Squared Error
RMSE evaluates rating prediction accuracy on the test set:

$$\text{RMSE} = \sqrt{\frac{1}{N}\sum_{i}(\hat{r_i} - r_i)^2}$$

It is computed for models that output explicit rating predictions (Item Based Collaborative Filtering and Matrix Factorization), using all test interactions, regardless of rating value.

Hit Rate@10
Hit Rate@10 evaluates top K recommendation quality. For each user, the model ranks all candidate items, a 'hit' occurs if the user's held-test movie appears in the top 10 positions. Only test interactions with **ratings greater than or equal 3** are treated as positive targets when computing Hit Rate@10, since they represent items, the user actually liked (we don't want a recommendation hit when the user's rating for the item is low)

$$\text{Hit Rate @10} = \frac{\text{Number of users with a hit}}{\text{Number of users with a positive test item}}$$

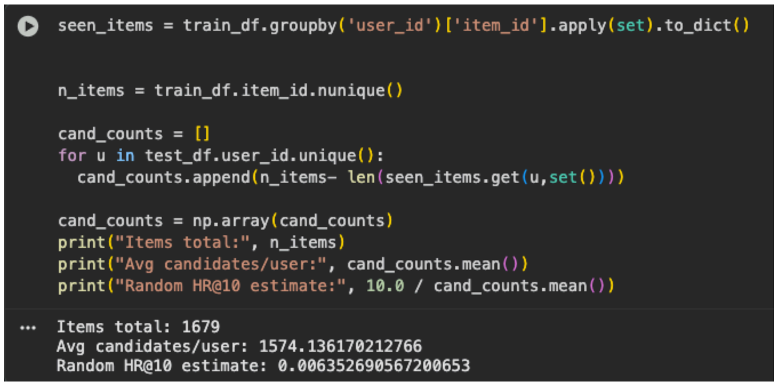
For each user in the filtered test set, the candidate set used for ranking consisted of:

- The held-out test movie, and
- All other movies the user has not rated in the training data

Thus, each user faces a large candidate pool (hundreds of unseen movies). This makes the ranking task nontrivial and ensures that top K metrics such as Hit Rate @ 10 are informative.

To contextualize model performance for Hit Rate@10, a simple random baseline was computed:

- For each user, items in the candidate set were ranked uniformly at random.
- Given an average candidate set size of around 1574 items, the probability of the true item appearing in top 10 by chance is approximately 0.0063



```
seen_items = train_df.groupby('user_id')['item_id'].apply(set).to_dict()


n_items = train_df.item_id.nunique()

cand_counts = []
for u in test_df.user_id.unique():
  cand_counts.append(n_items- len(seen_items.get(u,set())))

cand_counts = np.array(cand_counts)
print("Items total:", n_items)
print("Avg candidates/user:", cand_counts.mean())
print("Random HR@10 estimate:", 10.0 / cand_counts.mean())
```

```
Items total: 1679
Avg candidates/user: 1574.136170212766
Random HR@10 estimate: 0.006352690567200653
```

Figure 4: Random baseline

This baseline highlights how much better the learned models perform compared to random guessing.

4. Models & Methods Analysis

4.1 Model 1 — Item–Item Collaborative Filtering

4.1.1 Overview & Intuition

The Item Based Collaborative Filtering model recommends items based on their similarity to items that the user has previously rated. The underlying lying idea is that users tend to like items similar to items they liked before. Instead of modelling user-user similarity, the algorithm focuses on relationships between items, which is often more stable because items do not change their intrinsic characteristics over time, whereas users' preferences can shift due to mood, trends, or personal changes. A movie's attributes (genres, style, cast etc.) remain fixed, so the pattern of how users rate that item is relatively consistent, making similarity between items more reliable in the long run.

4.1.2 Methodology

The item-item CF pipeline used in this project consists of three main steps:

1. **Construct the item rating vectors**
   For each movie, we build a vector representing how different users rated it. The matrix is extremely sparse, but the relative patterns still encode similarity between items.

2. Compute item-item adjusted cosine similarities
An adjusted cosine similarity is computed between all pairs of movies:

$$\mathrm{sim}(i,j) = \frac{v_i \cdot v_j}{|v_i|\,|v_j|}$$

To reduce noise, similarity is computed using centered ratings which remove user biasness (some users rate harshly, others generously).

3. Compute predicted ratings (used for RMSE)
For each user-item pair (u,i) in the test set, let:

- $N(u)$ be the set of items rated by user u in the training data
- $sim(i,j)$ be the adjusted cosine similarity between items i and j, taken from precomputed similarity matrix
- Only positive similarities are used in ratings predictions to reduce noise (negative similarities often indicate opposing taste, which hurts ranking).

The predicted ratings are computed as:

$$\widehat{r_{u,i}} = \frac{\sum_{j\in N(u),\,\mathrm{sim}(i,j)>0} \mathrm{sim}(i,j)\; r_{u,j}}{\sum_{j\in N(u),\,\mathrm{sim}(i,j)>0} \mathrm{sim}(i,j)}$$

If the denominator is zero (i.e., no positively similar neighbors exist), the prediction is returned as **NaN**, and this sample is excluded from the RMSE calculations. This ensures that RMSE reflects only user-item pairs where the model has sufficient similarity evidence.

4. Compute efficient scoring strategy (used for Hit Rate@10)
Computing full rating predictions for all candidate items per user would be computationally expensive because each user may have hundreds of unrated items.

To improve efficiency, a **proxy scoring rule** was used:

- For each user u , let:
  - $seen(u)$ be movies rated in training set by user
  - i be candidate item not in $seen(u)$
- The score assigned to item is:

$$\mathrm{score}_u(i) = \sum_{j\in \mathrm{seen}(u)} \max(\mathrm{sim}(i,j),\, 0)$$

This similarity-sum heuristic is used because it produces ranking result consistent with similarity-weighted rating predictions in practice and is extremely fast and scalable.

The top 10 highest scoring items form the recommendation list. A hit occurs if the held-out test movie i^* satisfies:

$$i^* \in Top10(u)$$

Once again, only test items with **ratings greater than or equals 3** are included in HR @10 evaluation, since they represent meaningful positive preferences. This reduces our test set from 940 to 726 eligible users.

4.1.3 Performance

Using the temporal LOO split:

Metric	**Score**
RMSE	1.0570
Hit Rate @10	0.0854

4.1.4 Discussion & Analysis

The RMSE score of 1.0570 indicates moderate accuracy in predicting explicit rating values, it also highlights the challenge of estimating accurate rating values when many movies have very limited overlap in users who rated them. The Item CF model relies heavily on co-rated items to compute similarities, and sparse item-item interactions frequently lead to weak or unreliable similarity signals.

The Hit Rate @10 score of 0.0854 shows that the model does indeed capture meaningful structure compared to the random baseline of 0.0063. It successfully identifies items that are similar to the user’s previously enjoyed movies, improving retrieval performance substantially. However, one shortfall is that the model performance is limited by popularity effects. Since popular movies receive more reliable similarity estimates, as such, the model tends to favor well rated, higher degree items. While this helps to retrieve the correct item in some cases, it limits the model’s ability to recommend niche or less-rated movies.

The strength of Item-Item CF is that it is highly interpretable since recommendations can be directly explained through similar movies the user has enjoyed. Additionally, after computing the item-item similarity matrix, generating recommendations is relatively inexpensive since it only requires vector lookups and similarity sums.

However, Item-Item CF is highly sensitive to sparse data. If many movies have few ratings, it leads to unstable or poor similarity values, this limits coverage and often harms ranking quality. Additionally, the Item-Item CF model is unable to discover hidden relationships between users and items, since these relationships are not directly encoded in co-rating patterns.

Given the limitations observed, especially around sparsity and inability to capture hidden structures, it is natural to progress to Matrix Factorization models to address the weakness of Item-Item CF.

4.2 Model 2 — Matrix Factorization with User Clustering

4.2.1 Overview & Intuition

The Matrix Factorization (MF) model is a latent-factor model that represents each user and each item by a low dimensional vector. Instead of relying on direct co-ratings signals (as in Item-Item CF), MF learns hidden preference dimensions automatically. The core intuition is that users who rate items similarly should have similar latent vectors. This allows MF to infer missing ratings even when co-rating overlap is low, addressing several limitations of similarity-based collaborative filtering. To further stabilize predictions, user latent vectors were **later clustered using K-Means**, and cluster level average predictions were blended with user-specific MF predictions.

4.2.2 Methodology

The Matrix Factorization with User Clustering pipeline used in this project consists of five main steps:

1. **Data preparation**

   Each rating is expressed as a tuple (u, i, r_{ui}) where:
   - u is the user index
   - i is the item index
   - r_{ui} is the observed rating

   User and item IDs from training and testing set were mapped to consecutive integer indices to enable faster row lookups during SGD training.

2. **Latent factor model**

   Each user and item is assigned an f-dimensional latent vector

   $$P_u \in R^f, \qquad Q_i \in R^f$$

   Where the predicted rating is:

   $$\widehat{r_{ui}} = P_u \cdot Q_i$$

   Vectors were initialized with small random values drawn from a normal distribution.

3. **Training with Stochastic Gradient Descent**

   The loss function to minimize is given as such:

   $$L = \sum_{(u,i)\in\Omega} (r_{ui} - P_u \cdot Q_i)^2 + \lambda\left(\sum_u |P_u|^2 + \sum_i |Q_i|^2\right)$$

For each observed rating (u, i, r_{ui}):

1. Predict the rating

$$\widehat{r_{ui}} = P_u \cdot Q_i$$

2. Compute prediction error

$$e_{ui} = r_{ui} - \widehat{r_{ui}}$$

3. Update the latent vectors using L2-Regularized SGD

$$P_u \leftarrow P_u + \eta(e_{ui}Q_i - \lambda P_u)$$
$$Q_i \leftarrow Q_i + \eta(e_{ui}P_u - \lambda Q_i)$$

   Where:
   - η is the learning rate
   - λ is the regularization strength

   The update is repeated across all ratings **for 25 epochs**, until training RMSE stabilizes.

Although no explicit tuning was performed, the chosen parameters reflect conservative values used in MF for MovieLens-100K-scale datasets:

- Latent factors of dimension $f = 32$ is small enough to avoid overfitting under high sparsity yet large enough to capture meaningful interactions.

- Learning rate $\eta = 0.005$ is a stable step size ensuring smooth convergence

- Regularization $\lambda = 0.01$ controls overfitting by penalizing overly large latent values

- Epochs $= 25$ is sufficient for optimization curve to plateau in practice and during training we do observe training loss reduced significantly before stabilizing.

These settings strike a practical balance between model complexity, training stability, and computational cost.

4. Blending MF with user clustering (used for RMSE)

Although pure matrix factorization already allows us to predict a user's rating for any item, we will take an additional step to improve stability. In particular, we smooth out noisy user-level MF predictions by grouping users with similar latent vectors using K-means clustering and leveraging the shared behavior within each cluster:

1. Using the elbow method, K = 30 was selected.

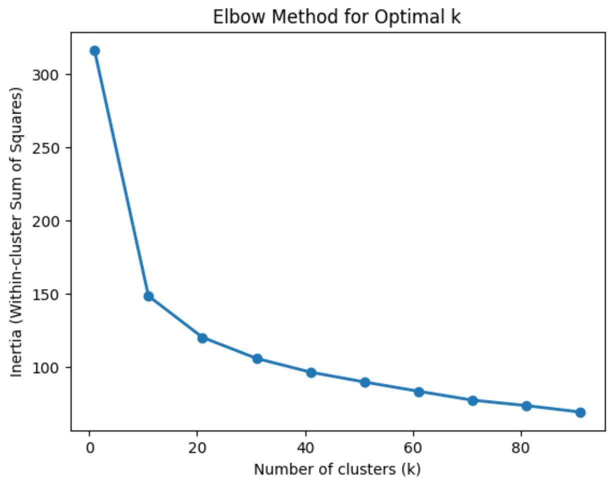


Figure 5: K Elbow Method

2. Apply K-Means on rows of $P \in R^{n_u \times f}$:

$$C(u) \in \{0,1,\dots,K-1\}$$

3. For each cluster k, compute centroid:

$$C_k = \frac{1}{|C_k|} \sum_{u \in C_k} P_u$$

4. Compute cluster-average prediction:

$$\widehat{r_{ui}^{(c)}} = C_{C(u)} \cdot Q_i$$

5. Final ratings is a convex combination:

$$\widehat{r_{ui}^{\text{blend}}} = \beta \widehat{r_{ui}} + (1-\beta) \widehat{r_{ui}^{(c)}}, \qquad \beta \in [0,1]$$

This reduces noise from users with sparse histories while still keeping personalization.

5. Top K recommendation with MF (used for Hit Rate @10)

Because users and items share the same latent space, the dot product between user vector P_u and an item vector Q_i gives us a relevance score:

$$s_{ui} = P_u \cdot Q_i$$

For each user u, we proceed as follows:

1. Compute scores s_{ui} for all items i.

2. Exclude items that user u has already rated in the training set.
3. Rank the remaining items by descending s_{ui} and take the top 10 as the recommendation list Top-10_u.
4. Check wether the held-out test item i_u^* for that user appears in Top-10_u , only if its rating is greater than or equal to 3 (meaningful positive preferences)

4.2.3 Performance

Using the temporal LOO split:

Metric	**Score**
RMSE (Pure MF)	1.0302
RMSE (MF + Clustering)	1.0296
Hit Rate@10	0.0303

4.2.4 Discussion & Analysis

Pure MF achieved an RMSE of 1.0302, and MF with clustering achieved RMSE of 1.0296, both outperforming Item-Item CF (1.0570). This improvement is expected since MF directly optimizes the squared error objective during training, whereas item-item similarity does not optimize any global loss function. MF also captures latent preference dimensions that are not visible through simple co-rating patterns, resulting in more accurate rating estimates.

Blending the user-specific MF prediction with the cluster-level average reduces variance in users with sparse or noisy rating histories. The improvement is small but consistent, showing that cluster centroids act as a form of regularization, preventing extreme user vectors from dominating predictions.

A potential extension would be to customize the blending weight B per user – a user with very few ratings (high noise) could rely more on the cluster average, while a user with many ratings could rely more on the individual MF predictions. This adaptive scheme might further reduce overfitting.

The Hit Rate@10 of 0.0303 is still significantly higher than the random baseline of 0.0062, this means that MF does produce meaningful ranking signals in its latent space. However, it performs worse than Item-Item CF (0.0854). This suggests that MF, when optimized only for RMSE, may not position relevant items at the top of a user's ranking list as effectively as similarity-based methods.

One possible reason for this might be that MF tries to represent all users and items onto one common latent space, because of this MF tends to produce smooth embeddings that capture global relationships. On the other hand, Item-Item CF leverages direct local co-rating similarities, prioritizing items very close to held out item. This gives Item-Item CF an advantage in Top K retrieval tasks.

The strength of MF models is that it can learn hidden factors like genre taste, theme preferences that similarity-based models cannot detect. This leads to better generalization, particularly in sparse regions of the matrix.

However, because MF focuses on minimizing squared error, the learned latent embeddings are not guaranteed to rank unseen items effectively, explaining the weaker Hit Rate@10 performance.

Although the matrix-based models (Item CF and MF) achieved reasonable accuracy in terms of RMSE, both however struggled to deliver strong top K recommendations. This suggest that relying solely on rating patterns may not be enough to capture user-item interactions. To address this limitation, the final model adopts a graph-based approach, allowing us to leverage connectivity patterns in the network.

4.3 Model 3 — Graph-Based Personalized PageRank

4.3.1 Overview & Intuition

While the first two models learned preferences purely from rating values, they did not explicitly account for connectivity structure between users and movies. Perhaps, by representing the data as a bipartite graph, it may allow us to leverage:

- User-movie relationships
- Multi-hop similarity paths through intermediate nodes
- Network structure that matrix methods overlook

Personalized PageRank (PPR) is a natural choice for this setting – instead of relying solely on direct ratings, PPR measures how strongly each movie is connected to a specific user through random walks on the graph, allowing us to capture direct and indirect relationships. This enables better top K retrieval, especially for items connected through longer paths that MF and Item CF may not capture.

4.3.2 Methodology

The Graph Based Personalized PageRank pipeline used in this project consists of three main steps:

1. **Bipartite Graph Construction**
   We construct a bipartite graph $G = (V, E)$ where:
   - User nodes: $u_0, u_1, \ldots$ and enriched with attributes like **age, gender and occupation** (from users dataset)
   - Movie nodes: $m_0, m_1, \ldots.$ and enriched with attributes **title and genre** (from movie dataset)
   - Edges: Between user u and movie m if u rated m
   - Edge weights: Rating values

Nodes are prefixed as:

- **u_<id>** for users
- **m_<id>** for movies

This makes the graph interpretable and compatible.

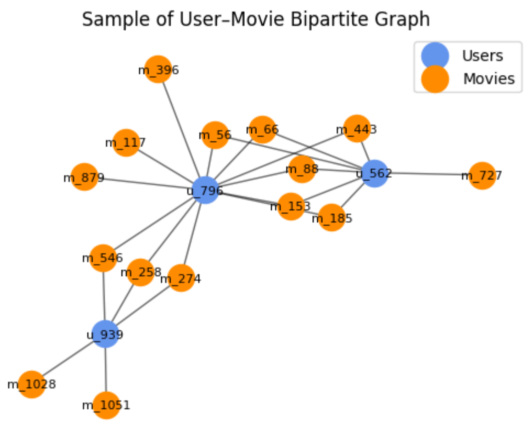


Figure 6: User-Movie Graph

2. Personalized PageRank (PPR)

In classical PageRank, a random walker:

- Moves to neighbors with probability a
- Teleports to a random node with probability $1 - a$

In personalized PageRank, the teleportation set is **restricted to a specific target user node.** Thus, for a user u, PPR solves:

$$\pi_u = \alpha A^{\top} \pi_u + (1 - \alpha) e_u$$

Where:

- π_u is the personalized PageRank vector
- A is the column-stochastic adjacency matrix
- e_u is the one-hot vector with 1 at user u's node

This behavior describes a random walker that repeatedly moves through connected users/movies and occasionally returns to the target user while accumulating high scores on nodes frequently visited and strongly connected to the user.

Movies with higher $\pi_u(m)$ are more relevant to user u.

3. Generating Recommendations

For each user u:

- Run Personalized PageRank staring at node u.
- Extract PageRank scores **for movie nodes only**.

- Remove movies the user has already rated.
- Recommend the Top 10 movies with highest PPR scores.

For the choice of damping factor a, we used the value of 0.85 which provides a good balance between **exploration** (following graph edges) and **personalization** (teleporting back to target user).

4.3.3 Performance

Using the temporal LOO split:

Metric	**Score**
Hit Rate@10	0.1171

4.3.4 Discussion & Analysis

Personalized PageRank has achieved the strongest top K performance among all models evaluated, and this follows from two key structural properties of the algorithm.

Firstly, Item-Item CF and MF depend mainly on local evidence – Item-Item CF uses direct co-rating similarity and MF uses pairwise user-item interactions encoded in latent features. In contrast, PPR considers **all possible paths** connecting a user to a movie through the graph. The random walker moves from target user to movies they rated, to other users who rated those movies and to those movies these users rated. As a result, this graph structure reveals indirect but strong item connections to a given user, allowing the algorithm to recommend relevant items even when the user has few ratings or direct similarity is weak or missing.

Secondly, from the PPR equation, we note that we are personalizing the walk to a user's own neighborhood, reinforcing items in the vicinity of the user in the graph. While MF learns a global latent space across users and items, PPR produces ranking tightly linked to the user's own local structure. This bias explains why PPR has the highest Hit Rate@10 scores.

The strength of PPR is that it remains effective when user histories are short or item co-ratings are low, just like in sparse matrices. This is because it infers relevance through network structure rather than solely on direct overlaps, making it more robust.

However, there is no rating prediction for PPR models, and it cannot participate in rating prediction tasks. Metrics like RMSE cannot be used on it. Even though PPR might be significantly stronger in retrieval tasks, it cannot replace MF or item CF when numeric ratings are needed.

4.3.5 Bonus: Genre Level Analysis of Global PageRank

Since we have enriched all our users and movies nodes with attributes, it would be a pity not to use them – to further interpret how the graph-based model behaves, we compute **Global PageRank (GPR)** scores on the full graph and aggregate these scores at the genre level.

While PPR reveals relevance to a specific user, GPR reflects structural importance – how central or connected movies of each genre are within the entire network.

This will allow us to compare **what users prefer** (via average ratings in EDA) against **what the network structural amplifies** (via PR scores).

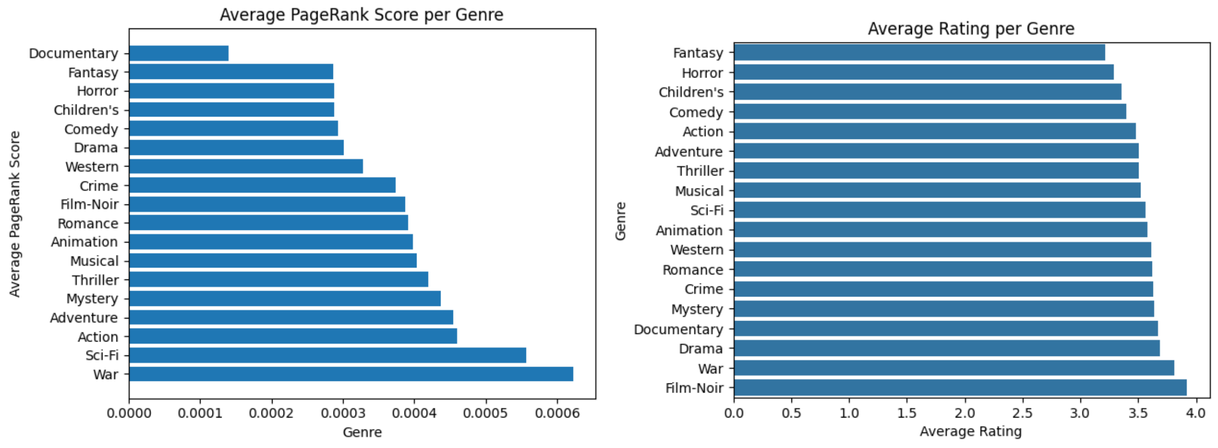


Figure 7: PR scores on Genres (Left) vs Average Rating on Genre (Right)

We see that preferences don't always align with network importance – genres such as Drama, and Film-Noir receive high average ratings. However, their PR scores are only moderate, indicating that although these genres are well liked, they are not as well-connected across the graph. Action, Sci-Fi and War have highest PR scores, these genres are structural hubs in the graph, and they bridge multiple user groups together, giving them strong network influence.

Beyond this, additional graph-based analysis such as community detection can be performed, but these analyses fall outside the scope of this project.

5. Conclusion

In this project, we evaluated three distinct recommender system approaches – Item-Item Collaborative Filtering, Matrix Factorization with Clustering and Personalized PageRank – on the MovieLens-100k dataset. Each model offering different strengths and limitations. From our results, we note no single model excels across all metrics, thus suggesting that in real-world recommendations, hybrid approaches that blend several models are often used, just like the Bellkor Solution to Netflix.