Movie Recommendation System Report

Clip Echendu

Department of Computer Science, University of North Carolina at Charlotte

ITCS-8162-U02

Professor Siddharth Krishnan

November 17th, 2025

Introduction

Movie recommendation systems are crucial for modern digital platforms such as Netflix, Amazon Prime, Hulu, and YouTube. Given the vast amount of available content, users often struggle to find movies that suit their preferences. These systems tackle this problem by examining user behavior and identifying patterns to predict which movies each user is most likely to enjoy. As a result, they enhance user satisfaction, boost engagement, and improve platform operations.

This assignment examined three main approaches to recommendation systems.

- **User-Based Collaborative Filtering:**

  This method identifies users with similar rating behaviors and recommends movies that those similar users have liked. It assumes that people with comparable tastes in the past are likely to maintain aligned preferences in the future.

- **Item-Based Collaborative Filtering:**

  This approach compares movies by analyzing how users rate them, rather than comparing users directly. It identifies movies that typically get similar ratings from similar user groups. These similar movies are then recommended to users based on their previously liked movies.

- **Random-Walk (Pixie-Based) Graph Recommendation:**

  This approach represents user-movie interactions as a bipartite graph, with users and movies as nodes and edges indicating ratings. Starting from a chosen movie, a random walk traverses connected nodes, recording how often it visits other movies. The movies visited most frequently are viewed as closely related and are recommended accordingly.

These three approaches collectively provide a comprehensive overview of recommendation system development, from basic similarity models to advanced graph-based techniques.

Dataset Description

This assignment utilizes the MovieLens 100K dataset, a well-known benchmark for evaluating recommendation algorithms. It contains 100,000 movie ratings from real users, accompanied by detailed metadata about both users and movies.

Dataset Overview

The dataset consists of three primary files:

- u.data – Contains 100,000 user-movie ratings
- u.item – Contains metadata for 1,682 movies
- u.user – contains demographic information for 943 users.

These datasets provide a comprehensive view of how users engage with movies.

Number of Users, Movies, and Ratings

- Users: 943
- Movies: 1,682
- Ratings: 100,000

Each user rated at least 20 movies, ensuring sufficient data for effective collaborative filtering.

Features Used in the Assignment -

From u.data (ratings):

- user_id
- movie_id
- Rating
- Timestamp

From u.item (movies):

- movie_id
- title
- release_date

From u.user (users):

- user_id
- age
- gender
- occupation

The primary features in the assignment are user_id, movie_id, rating, and title.

Preprocessing Performed -

The following preprocessing steps were applied before building the models.

- Loading the datasets – used pandas.read_csv with the correct delimiters.
- Timestamp conversion – converted timestamps into a human-readable date and time format.
- Merging datasets – connected ratings with movie titles using merge on movie_id.
- Handling missing values – filled missing ratings in the user-movie matrix with 0 for similarity calculations, and confirmed there are no missing values in the core data fields.
- Aggregation – In the random walk model, ratings from the same user for the same movie were averaged.
- Normalization involved centering each user's ratings by subtracting their average rating to remove user-specific biases.

Overall, these pre-processing steps ensure the data is clean, consistent, and ready for user-based, item-based, and graph-based recommendation methods.

Methodology

The programming assignment includes three recommendation methods: user-based collaborative filtering, item-based collaborative filtering, and a random-walk graph model using the Pixie algorithm. Each approach identifies different patterns in the data, allowing a thorough exploration of movie recommendation techniques.

User-Based Collaborative Filtering -

In this method, user-based collaborative filtering was implemented using the Movie dataset, aiming to recommend movies to users based on the preferences of similar users.

How User Similarity was calculated:

1. A user-movie rating matrix was constructed using Pandas, where:
    a. Rows represent users
    b. Columns represent movies
    c. Cell values represent ratings
2. Missing ratings (NaN) were replaced with 0, allowing vector-based similarity calculation.
3. Cosine similarity was calculated using “**sklearn.metrics.pairwise.cosine_similarity**” to measure the similarity between two users’ rating vectors.
4. For each target user, the most similar users were found, and their similarity-weighted average ratings were used to predict unseen movie ratings. These predicted scores then ranked movies, and the top recommendations were provided.

This approach identifies similarities in user preferences but may be limited by sparse data or varying rating behaviors among users.

Item-Based Collaborative Filtering -

In this method, item-based collaborative filtering was applied to the Movie dataset to recommend movies similar to a target movie based on user rating patterns. The underlying idea is that if two movies are rated similarly by many users, they are likely similar.

How Item Similarity Was Determined:

1. The movie-user matrix was obtained by transposing the user-movie matrix.
2. Missing ratings were replaced with 0, consistent with user-based filtering.
3. Cosine similarity was calculated between movies to determine their similarities.
4. The system identifies the most similar movies to the target film and recommends these to the user.

Item-based filtering tends to be more stable than user-based filtering, particularly when there are fewer movies than users.

Random-Walk Pixie Algorithm (Graph-Based Recommendation) -

The third method employs a graph-based approach inspired by the Pixie algorithm, representing user-movie interactions as a bipartite graph.

- A group of nodes stands for users.
- The other set consists of movies.
- Edges indicate the ratings.

Why Graph-Based Approaches Are Effective

Graph-based models leverage the connection structure rather than just similarity measures. This enables the model to identify indirect relationships between movies, higher-order connections, and densely connected local neighborhoods within the graph.

How the Random Walk Works:

1. Beginning at a selected movie node, a random walk is carried out on the bipartite graph.
2. At every step, the walker transitions from a movie to a user and then to another movie, recording each visited film.
3. Movies visited more frequently during the walk are seen as more relevant or connected to the starting movie, and the most frequently visited movies are provided as recommendations.

This method uncovers the fundamental logical connection between users and movies, making it especially useful for extensive, interconnected recommendation systems.

Implementation Details

This section describes how each recommendation function is implemented in the notebook. It details the construction of the user-movie graph using an adjacency list and explains how to create and rank recommendations using random walks.

Building the Recommendation Functions -

Three primary functions were implemented, each corresponding to one of the three methods performed:

- User-Based Collaborative Filtering – "recommend_movies_for_user(user_id, num=5)"
- Item-Based Collaborative Filtering – "recommend_movies(movie_name, num=5)"
- Random-Walk Pixie-Style Recommendation – "weighted_pixie_recommend(movie_name, walk_length=15, num=5)"

User-Based Collaborative Filtering Function -

- Initially, a user-movie rating matrix was constructed, with users as rows, movies as columns, and the ratings as the corresponding values.

- Missing ratings were filled with 0 to allow vector operations
- Cosine similarity was computed between all user vectors to form a user-user similarity matrix
- In implementing “recommend_movies_for_user”, the steps were:
    - Select the similarity scores for the target user.
    - Sort other users by similarity in descending order.
    - Use a similarity-weighted average of neighbors’ ratings to predict scores for the movies the user has not rated.
    - Exclude movies that the user has already rated.
    - Arrange the remaining movies in order of their predicted scores and return the top ‘num’ as recommendations.

Item-Based Collaborative Filtering Function -

- The user-movie matrix was transposed to create a movie-user matrix with movies as rows and users as columns.
- Missing values were once again replaced with 0.
- Cosine similarity was computed between movie vectors, yielding a movie-movie similarity matrix.
- In implementing “recommend_movies(movie_name, num=5)”:
    - The movie’s ‘movie_id’ was obtained using its title.
    - The similarity scores for this movie were obtained from the similarity matrix.
    - The movie was excluded from the list.
    - Movies were ranked based on their similarity score, from highest to lowest.

    - The top ‘num’ similar movies were mapped to their titles and provided as recommendations.

Constructing the Adjacency List Graph -

For the graph-based recommendation, user-movie interactions were represented as a bipartite graph and recorded as an adjacency list.

Graph Construction Steps:

- After merging ratings with movie titles and normalizing ratings per user, each row in the ratings Data Frame represented a (user_id, movie_id, rating, title)
- An empty dictionary graph = {} was created
- For each row in ratings:
    - If the user node did not exist in the graph, it was added with an empty set.
    - If the movie node did not exist in the graph, it was added with an empty set.
    - The user node was connected to the movie node “graph[user]. add(movie)”.
    - The movie node was connected to the user node “graph[movie]. add(user)”.

This generated an adjacency list with user or movie IDs as keys and sets of neighboring nodes as values. A separate set named ‘*movie_id_set'* was created from the movies DataFrame for quick verification of whether a node in the graph is a movie.

Random Walks and Ranking Visited Movies –

The function “weighted_pixie_recommend(movie_name, walk_length, num)” implemented a simplified Pixie-style random walk starting from a given movie.

Random Walk Steps:

- The input movie title was matched to its corresponding ‘movie_id’.
- The algorithm checked that this ‘movie_id’ existed in the graph.

- The random walk began at the movie node associated with the input movie.
- At each of the ‘walk_length’ breakdowns,
    - The current node’s neighbors were obtained from the graph adjacency list
    - One neighbor was chosen uniformly at random using ‘random.choice’
    - The walker moved to this neighbor node.
    - If the new node was a movie node and was not the stating movie, its visit count was incremented in the dictionary ‘visit_counts’.

Ranking and Returning Recommendations -

- After completing the walk, the ‘visit_counts’ dictionary contained the number of times each movie was visited.
- Movies were sorted in descending order of visit.
- The top ‘num’ movies were selected as recommendations
- These movie IDs were mapped back to movie titles using the ‘movies’ Data Frame
- A result Data Frame was created with a ‘Ranking’ column and a ‘Movie Name’ column containing the recommendation title.

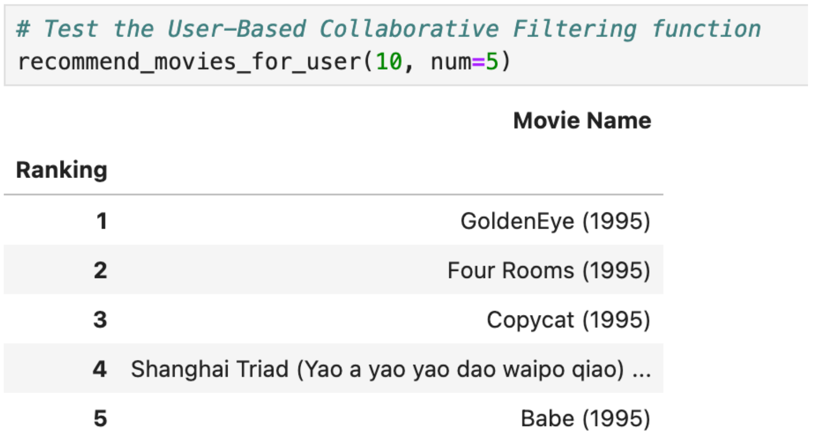
This implementation enabled the random-walk model to leverage the structure of the user-movie graph and identify movies strongly linked to the initial movie via shared user interactions.

Results and Evaluation

This section describes the results from three recommendation approaches: user-based collaborative filtering, item-based collaborative filtering, and a pixel-based, random-walk-inspired model. Each method was tested with example queries, and its performance and limitations were analyzed.

User-Based Collaborative Filtering Results –

The function was tested with *"recommend_movies_for_user(10, num=5)"*. It provided five movie suggestions based on the preferences of users similar to User 10. These suggestions reflected trends in the dataset, such as drama or action movies typically favored by similar users (see Figure 1 below).



```
# Test the User-Based Collaborative Filtering function
recommend_movies_for_user(10, num=5)
```

Ranking	Movie Name
1	GoldenEye (1995)
2	Four Rooms (1995)
3	Copycat (1995)
4	Shanghai Triad (Yao a yao yao dao waipo qiao) ...
5	Babe (1995)

Figure 1 - Testing User-Based Recommendation System

Evaluation -

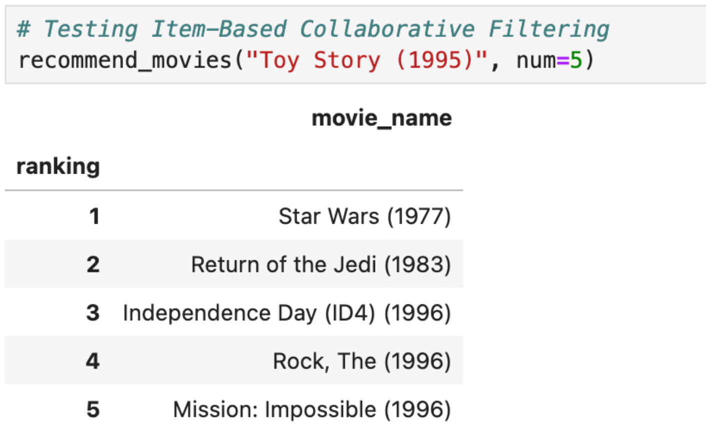
Based on Figure 1, the user-based recommendations exhibited a similar pattern of user behavior. They feature movies like GoldenEye (1995) and Four Rooms (1995), which are often rated by users with viewing habits similar to User 10. These findings were influenced by cosine similarity, which assesses how closely a user's profile aligns with others. The method showed some advantages and limitations, such as

Strengths: It effectively captured user preference patterns and generated personalized results based on taste similarity. Limitations: It is sensitive to sparse user-item matrices, and its performance declines when a user has rated very few movies.

Overall, the user-based model performed reliably for users with enough rating history and generated reasonable recommendations.

Item-Based Collaborative Filtering Results –

The similarity of items was tested using *“recommend_movies("Toy Story (1995)", num=5)”*. It grouped movies according to how similarly users rate them, rather than by genre or content. (see Figure 2 below).



```
# Testing Item-Based Collaborative Filtering
recommend_movies("Toy Story (1995)", num=5)
```

ranking	movie_name
1	Star Wars (1977)
2	Return of the Jedi (1983)
3	Independence Day (ID4) (1996)
4	Rock, The (1996)
5	Mission: Impossible (1996)

Figure 2 - Testing Item-Based Recommendation System

Evaluation –

Based on Figure 2, while “Toy Story” is an animated family film, the recommended movies included popular mainstream titles like Star Wars, Mission: Impossible, and The Rock. Although this might seem surprising, it makes sense within item-based collaborative filtering because movies are grouped based on similar user rating patterns. The method showed some advantages and limitations, such as

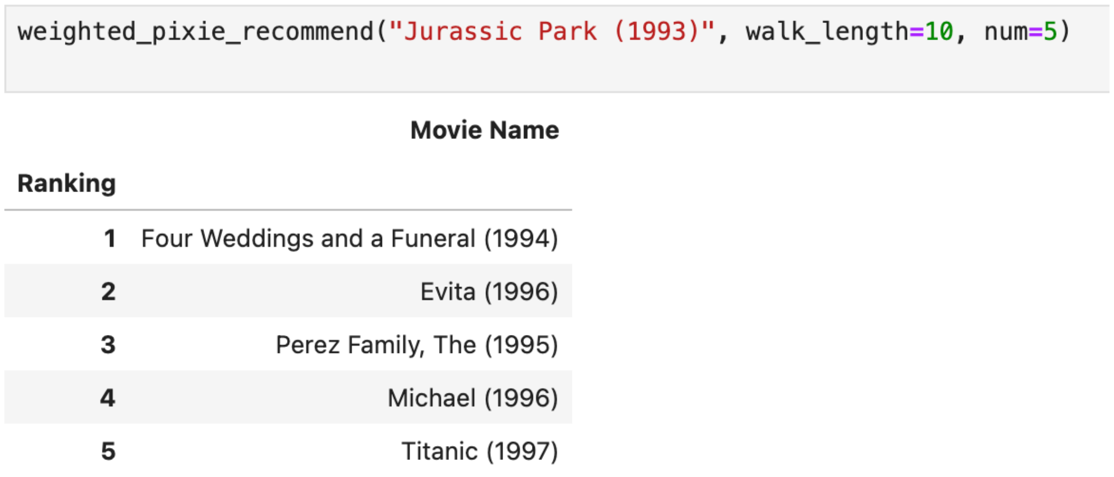
Strengths: It is more stable than user-based methods because movie similarity tends to change less often than user behavior.

Limitations: It may generate recommendations that appear counterintuitive if users anticipate content-based similarity.

Overall, the item-based model provided consistent and meaningful suggestions based on user behavior patterns.

Random-Walk Pixie-Style Recommendation Results –

The graph-based method was evaluated using *“weighted_pixie_recommend("Jurassic Park (1993)", walk_length=10, num=5)”*. The random walk explored the bipartite graph of users and movies and identified movies that were frequently visited during the walk. The recommendations varied depending on the starting movie, the random walk length, and the movie's graph density.



```
weighted_pixie_recommend("Jurassic Park (1993)", walk_length=10, num=5)
```

Ranking	Movie Name
1	Four Weddings and a Funeral (1994)
2	Evita (1996)
3	Perez Family, The (1995)
4	Michael (1996)
5	Titanic (1997)

Figure 3 - Testing Random-Walk Based Movie Recommendation

Evaluation –

The method showed some advantages and limitations, such as

Strengths: It captures multiple relationships in the user-movie network, and it often produces diverse recommendations

Limitations: The results depend heavily on the random walk's parameters. Short walks may not adequately explore the graph, leading to large, dense regions dominating the walk.

Overall, the pixie-style random walk captures rich graph structure and provides diverse results.

Conclusion

This project explored three approaches for building a movie recommendation system using the MovieLens 100K dataset: user-based collaborative filtering, item-based collaborative filtering, and a graph-based random walk inspired by the Pixie algorithm. These methods offered key insights into how recommendation systems function and how their success depends on the data and specific objectives. User-based collaborative filtering performed well for users with enough rating history, using cosine similarity to identify shared preferences. However, it faced challenges with sparse data and new users. Item-based collaborative filtering was more stable when comparing movies rather than users and gave consistent recommendations even with limited ratings, though some suggestions seemed counterintuitive from a content standpoint. The Pixie-inspired graph-based random walk provided a broader perspective by examining the user–movie graph structure, revealing multi-step and indirect relationships that traditional similarity methods might overlook. This approach yielded more diverse and sometimes surprising recommendations.

Potential Improvement –

While the three methods performed reasonably well, some enhancements could improve overall recommendation quality, such as

- The hybrid recommendation model, combining user-based and item-based similarities, could yield better results.

- Handling sparsity and cold start: using matrix factorization, such as Singular Value Decomposition (SVD), can reduce sparsity and improve predictions for new users or items.

Real World Applications –

The methods explored in the assignment can reflect real-world applications used across companies:

- Streaming platforms such as Netflix and Hulu use user-based and item-based collaborative filtering to suggest movies based on viewing patterns.
- E-commerce platforms such as Amazon rely on item similarity and purchase graphs to recommend related products.
- Social Media Platforms such as Pinterest, TikTok, and YouTube would likely use graph-based random walks to surface relevant, personalized content.

Overall, this project demonstrated how different recommendation strategies interpret user behavior in unique ways. Collaborative filtering offers strong personalization but struggles with data sparsity, while graph-based methods reveal deeper structural relationships in user interactions. Understanding the strengths and weaknesses of each approach helps in developing more effective and advanced recommendation systems.