Recommendation Report

I | Introduction

In the digital age, users have access to an overwhelming volume of films across streaming platforms such as Netflix, Hulu, and Amazon Prime. This abundance of choices gives rise to the long-tail phenomenon, where a small number of popular films receive most of the attention, while a large number of lesser-known titles are generally undiscovered. Additionally, since the domain that streaming platforms have to select from to offer recommendations is practically infinite, it can be incredibly difficult to narrow down that infinite range to some small finite number whilst keeping it completely personalized.

Film recommendation systems address this problem by guiding users against content that matches their taste, especially in the long tail. By doing so, they not only improve user satisfaction but also help platforms maximize the value of their content libraries. Recommendation systems make it possible for niche films to reach the right audience, which will increase the diversity of watching and expand the life cycle of lesser-known films.

To achieve this goal, movie recommendation systems apply one or more of these three strategies:

1. User-Based Collaborative Filtering: Identifies users with similar rating patterns and suggests movies one user has liked to another with overlapping preferences. This technique can highlight long-tail items if they are appreciated by like-minded users.

2. Item-Based Collaborative Filtering: Focuses on relationships between movies themselves. If two movies are often rated similarly by many users, they are considered similar. This allows the system to suggest less popular but similar alternatives to mainstream favorites.
3. Random-Walk-Based Methods: These graph-based approaches simulate exploration across a network of users and movies. Random walks are particularly effective at discovering non-obvious but relevant connections, which makes them powerful tools for surfacing content buried in the long tail.

II | Dataset Description

The MovieLens 100K dataset has the following structure:

- Number of Users: 943
- Number of Movies: 1682
- Number of Ratings: 100,000

The following features of the dataset were used:

- Users[user_id]
- Movies[movie_id, title]
- Ratings[user_id, movie_id, rating]

The following preprocessing steps were performed:

- Ratings[timestamp] was converted into the 'YYYY-MM-DD HH-MM-SS' format
- Ratings had all duplicate rows and any rows with a null value dropped

- Movies had all duplicate rows and any rows with a null value dropped
- Users had all duplicate rows and any rows with a null value dropped

III | Methodology

1. User-Based Collaborative Filtering
    a. User-Based CF operates under the assumption that users with similar past preferences are likely to enjoy similar items in the future. In this implementation, cosine similarity was used to measure the likeness between users based on their movie rating patterns
    b. **Calculating User Similarity:**
        i. The user-movie matrix was used, where each row represented a user and each column represented a movie
        ii. Missing ratings were replaced with zeros to create uniform-length vectors using **user_movie_matrix.fillna(0)**.
        iii. Cosine similarity was computed between each pair of users using **cosine_similarity**(), resulting in a user-user similarity matrix
        iv. Each entry in this matrix represents the cosine of the angle between two user rating vectors, indicating how similarly they rated the same set of movies
2. Item-Based Collaborative Filtering:
    a. Item-Based CF focuses on the relationships between the movies themselves. The key assumption is that if a user likes a

particular movie, they are likely to enjoy other movies that are similar in audience preference. This method is useful for producing stable, consistent recommendations, even when user activity is limited.

b. **Calculating Item Similarity:**
    i. The same user-movie matrix was used, but it was transposed, so each row now represented a movie, and each column represented a user
    ii. Missing ratings were again replaced with zeros using **user_movie_matrix.T.fillna(0)**.
    iii. Cosine similarity was computed between each pair of movies, resulting in an item-item-similarity matrix

3. Random-Walk-Based Pixie Algorithm:
    a. Graph-based methods are effective in recommendation systems because they naturally represent complex relationships between users, items, and other entities in a structured way. By modeling data as nodes and edges, these methods capture both direct and indirect connections—such as shared interests, co-purchases, or social ties—enabling more personalized and insightful recommendations. Techniques like random walks and personalized PageRank help surface relevant items by exploring the graph based on a user's activity and preferences. This approach is especially powerful for handling sparse data, cold-start problems, and discovering niche content, making it highly scalable and well-suited for real-time recommendations

in large-scale applications like social networks, e-commerce, and streaming platforms.

IV | Implementation Details

1. User-Based Collaborative Filtering
    b. **Preprocessing and Similarity Calculation:** To begin, missing ratings in the user-movie interaction matrix were filled with zeros using **.fillna(0)**, enabling the calculation of similarity scores without distortion from null values. Then, cosine similarity was computed between users, resulting in a user-user similarity matrix where each value indicates how closely two users' rating patterns align.
    c. **Identifying Similar Users:** For a given input user, the system retrieves a ranked list of similar users from the similarity matrix. The top **threshold** users (e.g., 5) with the highest similarity scores are selected to serve as peer recommenders.
    d. **Aggregating Recommendations:** The system gathers all the movies rated by similar users and computes the average rating for each movie across the group. This step assumes that if multiple similar users liked a movie, the target user might as well.
    e. **Filtering Already Watched Content:** To ensure relevance, any movies that the target user has already rated are filtered out from the recommendation pool. This guarantees that only new, unseen movies are suggested.

f. **Ranking and Output:** The top N movies, based on the average ratings from similar users, are selected and presented in a DataFrame along with their ranking. The output gives a clear and interpretable list of recommended titles for the user.

2. Item-Based Collaborative Filtering:
    a. **Transposing Rating Matrix & Calculating Item Similarity:** The user-movie matrix is transposed so that each movie becomes a row vector, and missing ratings are filled with zeros using **.fillna(0)**. This ensures that all vectors are of equal length for proper similarity computation. Then, cosine similarity is used to calculate how similar each movie is to every other movie, based on how users have rated them. The result is an item-item similarity matrix, where each entry reflects how closely two movies are related in terms of viewer ratings.
    b. **Locating Target Movie:** The system searches for the movie the user is interested in by matching its title. It handles edge cases such as missing titles or duplicate matches to ensure accurate lookup.
    c. **Identifying Similar Movies:** Once the target movie is identified, the system retrieves its similarity scores from the item-item matrix. It then sorts these scores in descending order to identify the top **num** most similar movies, excluding the target movie itself.
    d. **Mapping Movie IDs and Titles:** The IDs of the top-ranked similar movies are converted into titles by referencing the cleaned movie metadata. A small adjustment (movie_id - 1) is

made to align index mismatches between different data structures.

e. **Returning Recommendations:** The system constructs a DataFrame containing the final list of recommendations, ranked from most to least similar to the original input movie. Each entry includes the movie title and its ranking.

3. Random-Walk-Based Pixie Algorithm:
    a. **Input Validation and Setup:** The function initially checks if the movie exists in the dataset, if the **walk_length**, **num**, and **movie_name** parameters are valid, if the global **BETA** variable is valid, and if the given movie exists in the graph
    b. **Initialization:** The function retrieves the id of the given movie name and finds the corresponding node in the graph. It then initializes a dictionary of visited movies (**visited_movies**) that is used to count how many times each movie is visited during the walk
    c. **Random Walk Simulation:** The function then repeats the following for **walk_length** steps. It randomly selects a user connected to the current movie. It then randomly selects another movie from that user. If the movie is the same as the original movie, it skips to the next step. Otherwise, it logs the visit in **visited_movies**. If that movie had been visited **DEBUG_VISIT** times, the walk stops early. With probability (1 - **BETA**), the walk restarts from the original movie node
    d. **Sorting and Recommendation:** After the walk, the function sorts **visited_movies** by visit count in descending order. It then

selects the top **num** movie IDs and then uses the **movies_cleaned** DataFrame to get the titles of the recommended movies. It reformats the results into a new DataFrame with rankings and returns it.

V | Results & Evaluation

VI | Conclusion