Movie Recommendation System Report

1. Introduction

Movie recommendation systems help users discover films they are likely to enjoy by leveraging patterns in historical ratings and interactions, improving engagement and satisfaction while reducing choice overload in large catalogues. These systems are critical for platforms to personalize experiences, surface relevant content, and increase watch time, using structured signals from user–item interactions like those in MovieLens 100K. This report implements and contrasts three core approaches: user-based collaborative filtering, item-based collaborative filtering, and a Pixie-inspired random-walk method over a bipartite user–movie graph.

User-based collaborative filtering finds users with similar rating behaviours and recommends movies highly rated by those neighbours to a target user, capitalizing on shared taste profiles captured via similarity on the user–item matrix. Item-based collaborative filtering instead measures similarity between movies based on user co-rating patterns, recommending items that are most similar to a user's liked items and often yielding more stable similarities over time. A Pixie-inspired random walk models user–movie relationships as a graph and simulates weighted traversals that rank movies by visit frequency from a starting user or movie, enabling multi-hop discovery and diversity beyond direct similarity.

2. Dataset description

This project uses the MovieLens 100K dataset comprising 100,000 ratings with columns user_id, movie_id, rating, and timestamp, where ratings are integers on a 1–5 scale reflecting user preferences over time. The movies file provides 1,682 titles with metadata and release_date, while the users file describes 943 users with demographics like age, gender, and occupation, enabling analysis strictly on interactions without requiring content features. The features used in this work are user_id, movie_id, rating, timestamp from ratings, and movie_id, title, release_date from movies; demographic fields are loaded but not used in modeling to focus on collaborative signals.

Preprocessing steps include correct parsing of delimiters (tabs for ratings; pipes for movies and users), encoding handling for titles with accents, timestamp conversion to readable datetimes, and checks confirming no missing values across the core tables. Cleaned DataFrames are exported to CSV as ratings.csv, movies.csv, and users.csv for reproducibility and reuse across notebook sections and the graph-based implementation steps that reload these artifacts.

3. Methodology

User-based collaborative filtering begins by pivoting ratings into a user–movie matrix with users as rows and movies as columns and computing cosine similarity between users after filling unrated entries with zeros to enable vector operations consistently. For a target user, the system ranks other users by similarity, gathers their ratings for movies the target has not rated, aggregates scores (e.g., mean over neighbors), and returns the top-ranked movie titles, supporting interpretability as "users like you enjoyed these". This method relies on overlapping

rated items between users and benefits from groups of users with coherent tastes reflected in the similarity matrix.

Item-based collaborative filtering transposes the user–movie matrix so that movies become rows and users columns, then computes cosine similarity between movie vectors built from user ratings to measure co-preference structure. Given a query movie name, the recommender resolves its movie_id, retrieves similarity scores against all other movies, excludes the movie itself, and returns the top-N most similar titles, yielding stable and scalable recommendations grounded in aggregate user behavior over items. This approach is robust to fluctuating user bases and performs well when item co-rating patterns are strong and consistent.

The Pixie-inspired random-walk method first merges ratings with titles, aggregates repeated user–movie ratings by mean, normalizes by subtracting each user's average to reduce bias, and builds a bipartite adjacency list linking users to the movies they rated and vice versa. A weighted random walk starts from a user or a movie node and repeatedly transitions to a neighbor chosen with probability proportional to the neighbor's degree, counting visits only when landing on movie nodes and ranking movies by visit frequency to produce recommendations. This graph-based approach captures multi-hop structures (user→movie→user→movie), balances popularity and exploration via degree weighting, and often yields diverse, serendipitous results compared with direct similarity methods.

4. Implementation details

Data loading uses explicit separators and latin-1/UTF-8 encodings, with timestamp conversion and missing-value checks, followed by saving cleaned DataFrames to CSV for reliable downstream reuse and submission. The user–item matrix is created with a pandas pivot on ratings; NaNs are filled with 0.0 for cosine computations, and the matrix shape and head are inspected to validate transformation and sparsity. User-based similarity is computed via cosine_similarity on the zero-filled matrix, and the recommend_movies_for_user(user_id, num) function ranks neighbors, aggregates candidate movie scores, maps IDs to titles, and returns a tidy DataFrame with Ranking and Movie Name.

Item-based similarity uses the transposed matrix and cosine_similarity to produce an item–item similarity DataFrame indexed by movie_id; the recommend_movies(movie_name, num) function resolves the input title case-insensitively, excludes self-similarity, ranks items, and returns a ranked DataFrame of titles for display and use. For the graph-based method, ratings are merged with movies, aggregated by user_id, movie_id, title, normalized per user to remove rating bias, and then transformed into a bipartite adjacency list using sets/lists keyed by user and movie nodes for efficient neighbor lookups. Weighted random walks implement degree-proportional transitions with random.choices, track movie visit counts, sort by frequency, and return a ranked DataFrame; both user-started and movie-started variants are implemented and demonstrated with examples.

5. Results and evaluation

The user-based CF example for a target user returns a top-5 list with titles like “In the Company of Men (1997)”, “Misérables, Les (1995)”, and “Thin Blue Line, The (1988)”, reflecting consensus among the most similar users and filtering out movies the target already rated for proper discovery. The item-based CF example for “Jurassic Park (1993)” yields highly similar action/adventure titles such as “Top Gun (1986)”, “The Empire Strikes Back (1980)”, and “Raiders of the Lost Ark (1981)”, indicating strong co-rating patterns among audiences that enjoy high-production, mainstream films. The graph-based random-walk results from a user start produce a diverse set including “My Own Private Idaho (1991)”, “Aladdin (1992)”, and “12 Angry Men (1957)”, illustrating multi-hop discovery across genres and decades due to traversal dynamics and degree-based weighting.

Compared qualitatively, user-based CF excels at personalization via neighbor agreement but struggles with sparsity and cold starts; item-based CF is stable and scalable with strong interpretability but can bias toward popular, similar content; Pixie-inspired random walks offer greater discovery and diversity while balancing popularity via degree weights but introduce stochasticity and require parameter tuning for walk length. Implementation limitations include unweighted neighbor aggregation in user-based CF (no similarity weighting), reliance on basic cosine similarity without shrinkage or confidence weighting, and a simple degree-weight scheme in random walks rather than more advanced edge-weighting or restart mechanisms; nonetheless, all approaches return valid, interpretable recommendations aligned with their design.

6. Conclusion

This project demonstrates three complementary recommendation paradigms on MovieLens 100K: user-based CF for neighbor-driven personalization, item-based CF for stable similar-item retrieval, and Pixie-inspired random walks for serendipitous graph exploration, each producing coherent recommendation lists from the same interaction data. Key takeaways are that careful preprocessing (correct parsing, timestamp handling, normalization) and clear data structures (pivot matrices, adjacency lists) are essential, and that method choice trades off personalization, stability, discovery, and computational cost depending on product goals. Potential improvements include similarity-weighted neighbor aggregation, item- and user-centric normalization schemes, shrinkage or significance weighting, implicit-feedback handling, hybridization of CF and graph signals, restarts or multi-seed walks, and the inclusion of content/genre features for cold-start mitigation and re-ranking.

Real-world applications include powering home feeds and “Because you watched…” carousels, enabling item detail-page recommendations, supporting related-content discovery paths, and building scalable candidate generators where random walks act as high-recall retrieval ahead of precise rankers, all grounded in the approaches and artifacts validated in this implementation.