Movie Recommendation System Report

1. Introduction

Recommender systems have become a core component of modern digital platforms. Streaming services, e-commerce sites, social networks, and news applications all rely heavily on recommendation algorithms to help users navigate large catalogs of content. Without recommendation, users would be overwhelmed by the volume of items and would struggle to find content that matches their interests. By learning from historical user–item interactions, recommender systems can personalize the user experience, improve engagement, and ultimately support key business metrics such as retention and revenue.

In this project, we focus on building a movie recommendation system using the MovieLens 100K dataset. The goal is not only to generate reasonable recommendations, but also to understand different algorithmic approaches and the trade-offs between them. We implement three representative techniques: (1) user-based collaborative filtering, which recommends movies based on similar users; (2) item-based collaborative filtering, which recommends movies similar to a given title; and (3) a Pixie-inspired graph-based random walk algorithm, which explores a user–movie graph via random walks to surface relevant items. Each method uses the same underlying data but organizes and exploits it differently.

This report documents the dataset, preprocessing steps, methodology, implementation details, and qualitative evaluation of these models. We conclude with a discussion of limitations and potential extensions, such as hybrid approaches and richer feature sets, that could further improve recommendation quality.

2. Dataset Description

All experiments are performed using the MovieLens 100K dataset, a widely used benchmark for research in collaborative filtering. The dataset consists of exactly 100,000 ratings on a 1–5 scale. These ratings are provided by 943 distinct users on 1,682 distinct movies. It is intentionally sparse: most users rate only a small fraction of the available movies, and most movies receive feedback from only a subset of users. This sparsity reflects realistic conditions in production recommender systems.

The raw data is provided in three core files. The ratings file (u.data) contains four columns: user ID, movie ID, rating value, and a UNIX timestamp indicating when the rating was recorded. The movie metadata file (u.item) contains movie ID, movie title, release date, a video release date (often empty), an IMDb URL, and a set of binary genre indicator columns. The user metadata file (u.user) contains user ID, age, gender, occupation, and ZIP code. In the notebook, these files are loaded into pandas DataFrames named ratings, movies, and users using the correct delimiters (tab for ratings, pipe for movies and users).

After loading, we perform basic inspection and cleaning. We validate that the number of unique user IDs and movie IDs matches the dataset description, and we check for missing values in critical columns. The rating column is complete, and any missing values in optional movie fields (such as video release date) do not affect our algorithms. We also convert the

UNIX timestamp in the ratings table to a human-readable datetime type to make the data easier to explore, although time-based effects are not directly used in the models. Finally, we export the cleaned DataFrames as ratings.csv, movies.csv, and users.csv so that the processed data can be reused without re-parsing the original text files.

For the algorithms implemented in this assignment, the main features we rely on are: user_id, movie_id, and rating from the ratings table, and movie_id and title from the movies table. Demographic attributes from the users table and genre indicators from the movies table are not consumed by the current models, but they provide useful context and could be leveraged in future work for content-aware or demographic-aware recommendations.

3. Methodology

We implement three complementary recommendation approaches. Together, they illustrate how the same underlying data can be modeled from user-centric, item-centric, and graph-centric perspectives.

3.1 User-Based Collaborative Filtering

User-based collaborative filtering is one of the earliest and most intuitive recommendation techniques. The central idea is that if two users have rated many of the same movies in similar ways, then they likely share similar preferences. Therefore, a good way to recommend movies to a user is to examine what similar users have liked that the target user has not yet seen.

To operationalize this idea, we first construct a user–movie rating matrix using a pivot operation. Each row corresponds to a unique user ID, each column corresponds to a unique movie ID, and each cell contains the rating that user gave to that movie. Because not every user rates every movie, many entries in this matrix are missing. For similarity computation, we fill these missing ratings with 0. Although substituting 0 is a simplification, it enables the use of standard vector similarity measures and works well enough for this assignment.

We then compute a user–user similarity matrix using cosine similarity. Cosine similarity measures the cosine of the angle between two rating vectors and ranges from –1 to 1, with higher values indicating more similar rating patterns. The result is a square matrix user_sim_df whose rows and columns are indexed by user IDs. To generate recommendations for a specific user, we retrieve the row corresponding to that user, sort other users by decreasing similarity, and select the top-k neighbors. For each candidate movie that the target user has not rated, we compute a predicted score as a weighted average of neighbors’ ratings, where the weights are the similarity values. The movies with the highest predicted scores are recommended.

3.2 Item-Based Collaborative Filtering

Item-based collaborative filtering takes a complementary view. Instead of comparing users to each other, it compares movies. Two movies are considered similar if they are rated in similar ways by many of the same users. This approach is particularly useful for producing statements like “users who liked X also liked Y,” making it a natural fit for interfaces that recommend items similar to the one currently being viewed.

We construct the item–item similarity matrix by transposing the user–movie rating matrix, so that rows correspond to movies and columns correspond to users. Again, missing entries are filled with 0. We compute cosine similarity between movie vectors to obtain a square matrix item_sim_df indexed by movie IDs. To recommend movies similar to a given title, we look up its movie ID, retrieve the corresponding row from item_sim_df, sort other movies by descending similarity, and select the top-N results (excluding the movie itself). We then map these movie IDs back to titles using the movies DataFrame. This yields a ranked list of movies that share similar rating patterns with the input movie.

3.3 Pixie-Inspired Graph-Based Random Walk

The third approach is inspired by Pixie, a graph-based recommendation algorithm used at Pinterest. Instead of representing the data as a matrix, we represent it as a bipartite graph with two types of nodes: users and movies. An undirected edge connects a user to a movie if the user has rated that movie. The intuition is that relevant recommendations can be found by exploring paths in this graph that connect the target user to other movies through chains of intermediate users and items.

Before building the graph, we perform an additional normalization step on the ratings data. For each user, we compute their mean rating and subtract it from each of their individual ratings, centering their rating distribution around zero. This normalization reduces the impact of users who systematically rate higher or lower than average and yields a more balanced view of relative preferences. We then build an adjacency list called graph, implemented as a Python dictionary where each key is a node ID (user ID or movie ID) and the value is the set of neighboring node IDs.

To generate recommendations, we perform a random walk starting from the target user’s node. At each step, we randomly choose one of the current node’s neighbors and move to that node. Whenever we land on a movie node, we increment a counter for that movie. After a predefined number of steps, we stop the walk, filter out movies the user has already rated, and rank the remaining movies by visit count. The highest-ranked movies are recommended. This procedure captures multi-hop relationships in the graph and can surface movies that are indirectly connected to the user through a sequence of similar users and items.

4. Implementation Details

All models are implemented in a single Jupyter Notebook using Python, pandas, NumPy, and scikit-learn. The notebook structure mirrors the stages of the assignment: data loading and cleaning, construction of the user–movie matrix, development of collaborative filtering models, graph construction, and implementation of the random walk recommender.

For user-based collaborative filtering, the user–movie matrix is created using the pandas pivot function. Missing values are filled with 0, and cosine similarity is computed using sklearn.metrics.pairwise.cosine_similarity. The function recommend_movies_for_user(user_id, num) retrieves the similarity scores for the given user, selects the top neighboring users, and computes a weighted score for each candidate movie.

The function returns a neatly formatted DataFrame listing the top-N recommended movie titles ordered by their predicted scores.

For item-based collaborative filtering, the transposed rating matrix is processed in a similar way. The function recommend_movies(movie_name, num) first finds the movie ID that matches the given title, then reads the similarity scores from item_sim_df. It sorts these scores, ignores the movie itself, and returns the titles corresponding to the top-N most similar movies. By leveraging the same cosine similarity infrastructure, we keep the implementation concise while changing only the perspective from users to items.

The graph-based recommender begins by merging the ratings and movies DataFrames so that each rating row includes a movie title, then grouping by user_id and movie_id to compute mean ratings. Ratings are normalized per user, and the graph is constructed by iterating over each row and inserting edges into the adjacency list. The function weighted_pixie_recommend(user_id, walk_length, num) precomputes the set of all user IDs and movie IDs so that it can distinguish node types during the walk. It initializes the current node to the starting user, performs a loop of the specified walk_length, and at each step chooses a random neighbor of the current node using Python’s random.choice. Each time a movie node is visited, its visit count is incremented. At the end of the walk, the function removes movies already rated by the user, sorts the remaining movies by count, and returns the top-N as recommendations.

5. Results and Evaluation

The evaluation in this project is qualitative rather than quantitative, as the focus is on understanding algorithm behavior rather than optimizing for a particular metric. For user-based collaborative filtering, when we query the recommender for a specific user, the recommended movies tend to be popular titles with strong average ratings that align with the genres the user has rated highly in the past. For example, users who rate action and science-fiction movies favorably typically receive recommendations that include other high-rated entries in those genres.

In the item-based collaborative filtering model, using a seed movie such as “Jurassic Park (1993)” returns recommendations that are intuitive and easy to interpret. The top similar movies generally include other action or adventure films that many of the same users have rated highly, such as classic blockbusters or thematically related titles. This confirms that cosine similarity on rating vectors is able to capture meaningful relations between movies, even without any explicit genre information.

The Pixie-inspired random walk recommender produces recommendations that are influenced by the local structure of the user–movie graph. Because the walk alternates between users and movies, it can reach movies that are multiple hops away from the starting user. Some of the suggested movies overlap with those recommended by collaborative filtering, but others are more surprising: they may be connected through users who share niche preferences with the target user. This can introduce serendipity, which is often desirable from a user-experience perspective.

One limitation of the current evaluation is the lack of a held-out test set and formal metrics such as precision@k or recall@k. Without these metrics, we cannot precisely quantify which algorithm is “best.” Additionally, the user-based and item-based models fill missing ratings with zero, which may distort similarity calculations. The random walk model currently uses a single short walk with uniform neighbor selection; in practice, using multiple walks, weighted edges, and restart mechanisms would yield more stable and accurate results.

6. Conclusion and Future Work

This project demonstrates three foundational techniques for building a movie recommendation system on the MovieLens 100K dataset. User-based collaborative filtering leverages similarity between users to estimate ratings for unseen movies. Item-based collaborative filtering focuses on relationships between movies and is especially effective for generating “similar items” recommendations. The Pixie-inspired graph-based random walk model exploits the structure of the user–movie graph to uncover multi-hop relationships and provide recommendations that can blend relevance with serendipity.

In real-world deployments, recommender systems rarely rely on a single algorithm. Instead, they often combine multiple signals and models into hybrid architectures. For example, item-based collaborative filtering might power a “similar titles” carousel, while graph-based methods and latent factor models contribute to a personalized home page. The techniques explored here could serve as building blocks within such a system, and understanding their behavior helps in designing effective hybrids.

There are several clear directions for future improvement. First, we could introduce a proper train–validation–test split and evaluate all models using standardized metrics such as precision, recall, and mean average precision at various cutoff values. Second, we could experiment with alternative similarity measures and matrix normalization schemes to reduce the impact of popularity and rating scale differences. Third, we could significantly enhance the Pixie-inspired approach by incorporating edge weights based on rating magnitude or recency, running multiple random walks with restarts, and pruning low-probability paths. Finally, adding side information—such as genres, textual descriptions, or user demographics—would allow us to explore hybrid content-aware models that can generalize better for new or sparsely rated movies.

Overall, this project underscores the versatility of collaborative filtering and graph-based techniques in recommendation tasks. By carefully modeling interactions between users and items, and by choosing algorithms appropriate to the data and use case, we can design recommendation systems that meaningfully assist users in discovering content they value.