

In the age of digital media and streaming platforms, movie recommendation systems have become indispensable tools for enhancing user experience and engagement. With vast libraries of films and television content available at users' fingertips, these systems help viewers navigate through overwhelming choices by surfacing relevant and personalized suggestions. Whether it's Netflix recommending a drama based on your previous watches or IMDb suggesting trending thrillers, recommendation engines are central to content discovery, user retention, and platform success.

Movie recommendation systems rely on various algorithmic strategies to predict what a user might enjoy watching next. Among the most common techniques are User-Based Collaborative Filtering (UBCF) and Item-Based Collaborative Filtering (IBCF). UBCF recommends movies to a user by identifying similar users based on shared viewing or rating patterns. It assumes that users who agreed in the past will continue to agree in the future. IBCF, on the other hand, focuses on relationships between items (movies). It identifies similarities between movies based on how they were rated or interacted with across users and recommends items similar to those the user has already liked.

While collaborative filtering methods have proven effective, they often suffer from scalability and sparsity issues as the user-item matrix grows. To overcome these limitations, more advanced approaches like Graph-Based Recommendation Algorithms, including Pixie-inspired models, have gained traction. These systems represent users and items as nodes in a graph, with interactions forming edges, and leverage random walks to explore relevant connections in the network. The Pixie-inspired method enhances personalization and scalability by using multiple random walks from a user's interaction history to discover tightly connected and contextually relevant movie suggestions.

Each of these approaches brings unique strengths. UBCF is intuitive and easy to implement but struggles with new users (cold start problem). IBCF scales better and offers item-level insights, which are more stable over time. Graph-based models, though computationally more complex, provide richer contextual recommendations by capturing complex relationships in large-scale interaction graphs. Together, these techniques form the backbone of modern recommender systems, and choosing the right approach often depends on the dataset size, diversity of content, and performance requirements.

For this project, I used the MovieLens 100K dataset, which contains 100,000 movie ratings (on a scale of 1 to 5) provided by 943 users for 1,681 different movies. Each user in the dataset has rated a minimum of 20 movies, ensuring sufficient data for collaborative filtering techniques. Additionally, the dataset includes basic demographic information for each user, such as age, gender, occupation, and ZIP code, allowing for potential user segmentation and enhanced personalization. The features selected were `user_id`, `movie_id`, `rating`, `timestamp` from the ratings dataset; `title` and `release_date` for each movie; and `age`, `gender` and `occupation` for each user. For preprocessing the data, the missing values were handled by dropping the corresponding rows since the number was not significant, duplicates were identified and removed, and the `timestamp` column was converted into `datetime` format.

For implementation of recommendation system, a *user_movie_matrix* was created by pivoting the ratings dataset and plotting the users against the movies and filling the cells with their rating. Any movie not rated by a user was filled with 0 value.

For user-based collaborative filtering, I used cosine similarity to measure how similar each pair of users is based on their movie ratings. The user-based collaborative filtering implementation revolves around identifying users with similar preferences to the target user and leveraging their ratings to generate recommendations. To achieve this, I defined the function `recommend_movies_for_user(user_id, num=5)`. The function first retrieves the similarity scores for the given `user_id` from a precomputed user similarity matrix, sorting them in descending order and excluding the user themselves. It then gathers movie ratings from these top similar users. For each movie, the function computes the average rating across these users to estimate relevance. The movies are then sorted by their average ratings in descending order, and the top `num` recommendations are selected. Finally, movie IDs are mapped to their corresponding titles using the movies DataFrame, and the results are returned as a ranked Pandas DataFrame.

In item-based collaborative filtering, we compute movie-to-movie similarity using cosine similarity on user ratings. Since `cosine_similarity` doesn't handle missing values, we fill them with zeros. The *user_movie_matrix* is transposed so movies become rows, enabling similarity calculation between them. After applying

cosine_similarity, the result is converted into a Pandas DataFrame with movie titles as both rows and columns for easy reference. To recommend similar movies, we define `recommend_movies(movie_name, num=5)`. The function finds the corresponding `movie_id` for the given title, and if not found, returns an appropriate message. It then retrieves and sorts similarity scores from `item_sim_df`, excluding the movie itself. The top `num` results are mapped back to their titles and returned as a ranked Pandas DataFrame.

In the Pixie-inspired random walk recommendation method, we first preprocess the MovieLens dataset to construct a user-movie interaction graph. Each user is connected to the movies they've rated, and each movie is connected back to the users who rated it, forming a bidirectional structure ideal for graph traversal. To handle duplicate ratings, we group the dataset by `['user_id', 'movie_id', 'title']` and compute the mean rating for each movie by each user. We then normalize ratings by subtracting each user's average rating from their individual ratings, reducing personal bias and making preferences more comparable across users.

We represent the graph using an adjacency list implemented as a dictionary. Each key corresponds to a user or movie node, and the values are sets of connected nodes. By iterating through the normalized dataset, we add movies to each user's set of connections and users to each movie's set. This undirected graph structure captures all relevant relationships and serves as the basis for random walk-based exploration, allowing the recommendation algorithm to identify relevant movies by simulating user navigation through similar interests in the network. Graph-based approaches are effective because they capture complex relationships and indirect connections between users and items that traditional methods might miss. By modeling interactions as a network, these approaches enable personalized and context-aware recommendations through efficient traversal and pattern discovery.

Implementation details are as follows:

`get_sim_scores(user_sim, user_id)`

- Retrieves similarity scores between the target user and all other users.
- Sorts scores in descending order.
- Removes the user's own similarity score.
- Returns a DataFrame with other users and their similarity scores.

`get_unrated_movies(user_movie, user_id)`

- Gets the row for the target user from the user-movie matrix.
- Intended to return movies not rated by the user (ratings == 0).

`get_user_ratings(user_movie, movie_id, user_df)`

- For a given movie, gets ratings from similar users.
- Merges those ratings with the similarity scores.
- Calculates average rating (currently unweighted).

`top_n_recs(user_id, num)`

- Gets similar users using `get_sim_scores`.
- Finds movies not rated by the user using `get_unrated_movies`.
- Predicts ratings for each unrated movie using `get_user_ratings`.
- Sorts movies by predicted rating and selects top n.
- Merges with movies DataFrame to get titles.
- Returns list of recommended movie titles.

`print_movie_table(movies)`

- Prints recommended movies in a simple, markdown-style table with rankings.

`print_movie_table(n_movies)`

- Generates top 5 recommendations for user 10 and prints them as a table.

`cosine_similarity(user_movie_matrix.T.fillna(0))`

- Transposes the user-movie matrix so that movies become rows.
- Fills missing values with 0 (assumes unrated = 0).
- Computes cosine similarity between movies based on user ratings.

`item_sim_df = pd.DataFrame(...)`

- Converts the similarity matrix into a pandas DataFrame.
- Uses movie IDs as both row and column labels for easier lookup.

`get_n_movies(movie_name, num)`

- Searches for the `movie_id` corresponding to the given `movie_name`.
- If not found, returns "Movie not found".
- Retrieves the similarity scores between the given movie and all others.

- Sorts similar movies in descending order (most similar at the top).
- Drops the movie itself from the list (since it's 100% similar to itself).
- Merges with movies DataFrame to get movie titles.
- Returns a list of the top num similar movie titles.

`print_movie_table(movie_list)`

- Gets the top 5 most similar movies to Jurassic Park (1993).
- Prints them in a clean, ranked markdown-style table.

Graph Construction Section

- Creates three dictionaries (`graph`, `user_graph`, and `movie_graph`) to separately track connections between users and movies.
- Iterates over each row in the ratings DataFrame, extracting `user_id` and `movie_id`.
- Populates the graph dictionary with undirected connections between users and movies (i.e., `user ↔ movie`).
- Populates `user_graph` and `movie_graph` separately to maintain type-specific connections—`user_graph` stores movies rated by each user, and `movie_graph` stores users who rated each movie.

`weighted_pixie_recommend(user_id, walk_length=15, num=5)` Function

- Checks if the given `user_id` exists in the graph to begin the recommendation process.
- Initializes a visited dictionary to count how many times each movie is visited during the walk (initially set to 0 for all movies).
- Performs a random walk of length `walk_length`, alternating between user and movie nodes.
- Only increments the visit count for movie nodes, ignoring user visits.
- After the walk, sorts movies by visit frequency in descending order to prioritize frequently encountered items.
- Retrieves the top num most visited movies and maps their IDs to movie titles using the movies DataFrame.
- Returns a ranked Pandas DataFrame of recommended movies for the given user.
- Returns an error message if the `user_id` is not found in the graph.

In the Pixie-inspired recommendation system, random walks are used to simulate a user navigating through a user-movie interaction graph. Starting from the given `user_id`, the algorithm performs a series of alternating hops—first to a connected movie, then to a user who rated that movie, and so on—for a fixed number of steps (`walk_length`). This simulates how a user might explore related content through indirect relationships in the graph.

During the walk, the algorithm only tracks visits to movie nodes, incrementing a count in the visited dictionary each time a movie is reached. User visits are ignored for ranking purposes. After completing the walk, the visited dictionary contains the number of times each movie was encountered. The algorithm then sorts the movies in descending order of visit frequency, assuming that more frequently visited movies are more relevant. The top `num` movies from this sorted list are selected and mapped to their titles to produce the final ranked recommendations.

For results and evaluation, when comparing user-based, item-based, and Pixie-inspired random walk recommendation methods, each approach offers distinct strengths and limitations in terms of accuracy and usefulness depending on the context and scale of the application:

User-Based Collaborative Filtering

Accuracy: Performs well when users have rated many common items, enabling precise similarity calculations. However, its accuracy drops significantly in sparse datasets due to limited overlap.

Usefulness: Easy to implement and interpret. Works best in small to medium-sized systems, but struggles with scalability and the cold-start problem (new users with few ratings).

Item-Based Collaborative Filtering

Accuracy: Generally more stable and accurate than user-based filtering, especially in large datasets, because item similarities tend to remain consistent over time.

Usefulness: Highly scalable and interpretable, making it practical for production systems. It's also less affected by new user entries, though it still suffers when new items lack enough ratings (item cold-start).

Pixie-Inspired Random Walk (Graph-Based)

Accuracy: Highly accurate for personalized recommendations, especially in dense graphs, because it captures indirect relationships and user intent more effectively through multi-hop paths.

Usefulness: Excellent for real-time, large-scale systems like Pinterest or Netflix. It adapts quickly to new interactions and provides context-aware suggestions. However, it is computationally more complex and requires efficient graph data structures and memory management.

In conclusion:

User-based is best for small, dense datasets but struggles at scale.


Item-based is more robust and scalable with decent accuracy.

Pixie-inspired random walks offer the most personalized and context-aware recommendations, making them ideal for dynamic, large-scale environments, despite the added complexity.

Examples from the code:

User-based recommendation


```
[ ] n_movies = top_n_recs(34, 5)
    print_movie_table(n_movies)
```



Ranking	Movie Name
1	Entertaining Angels: The Dorothy Day Story (1996)
2	Someone Else's America (1995)
3	Great Day in Harlem, A (1994)
4	Aiqing wansui (1994)
5	They Made Me a Criminal (1939)



Item-based recommendation

```
movie_list = get_n_movies("Speed (1994)", 5)
print_movie_table(movie_list)
```





Ranking	Movie Name
1	Jurassic Park (1993)
2	True Lies (1994)
3	Top Gun (1986)
4	Fugitive, The (1993)
5	Terminator 2: Judgment Day (1991)

Pixie-inspired graph recommendation

 `weighted_pixie_recommend(20)`

Ranking	Movie Name
1	Indiana Jones and the Last Crusade (1989)
2	Star Trek: First Contact (1996)
3	L.A. Confidential (1997)
4	Jackie Brown (1997)
5	Batman (1989)



The following are the limitations of each recommendation system:

User-Based Collaborative Filtering

Scalability Issues: As the number of users grows, computing similarity between users becomes expensive.

Data Sparsity: In large datasets, users often rate only a small fraction of items, leading to few overlaps and unreliable similarity scores.

Cold Start (Users): Fails to provide recommendations for new users who haven't rated enough items.

Dynamic Preferences: Does not easily adapt to changing user interests over time without re-computing similarities.

Item-Based Collaborative Filtering

Cold Start (Items): Cannot recommend new or rarely rated movies due to lack of rating data.

Popularity Bias: Tends to favor well-known or frequently rated movies, which can limit diversity in recommendations.

Context Ignorance: Does not consider user-specific preferences beyond what they've rated—recommendations are based purely on item similarity.

Static Similarity: Item similarity is precomputed and might not adapt quickly to trending or time-sensitive changes.

Pixie-Inspired Random Walk (Graph-Based)

Computational Complexity: Random walks and graph traversal operations can be memory-intensive and require optimized infrastructure for real-time performance.

Graph Maintenance: Requires careful construction and updating of the graph as new users and items are added or updated.

Cold Start (Graph Dependency): Needs meaningful user-item interaction data to build an effective graph—suffers if the graph is sparse or disconnected.

Interpretability: The reasoning behind recommendations can be less transparent compared to traditional filtering methods, making it harder to explain to end-users.

In this project, we explored three distinct recommendation strategies—User-Based Collaborative Filtering, Item-Based Collaborative Filtering, and Pixie-Inspired Random Walks—using the MovieLens 100K dataset. Each method demonstrated unique advantages and trade-offs in terms of accuracy, scalability, and implementation complexity. While collaborative filtering approaches offered simple, interpretable recommendations based on user or item similarity, they were limited by sparsity and cold-start issues. The Pixie-inspired graph-based model, although more complex, provided richer and more personalized results by leveraging the structure of user-movie interactions and simulating discovery through random walks.

Overall, the project highlights how the choice of recommendation technique depends on the size and nature of the dataset, the desired level of personalization, and the system's performance needs. Combining multiple approaches or adopting hybrid systems can often yield more balanced and effective recommendations in real-world applications.

While the current implementations offer a solid foundation, several improvements could enhance the performance and accuracy of these recommendation models. For collaborative filtering methods, incorporating weighted similarity metrics or matrix factorization techniques like Singular Value Decomposition (SVD) can better capture latent relationships between users and movies. For the Pixie-inspired random walk model, adding edge weights based on normalized ratings or user engagement frequency could make the walks more meaningful. Additionally, integrating demographic data (age, gender, occupation) and temporal context (e.g., trends or time-based preferences) could help personalize results further. Scaling the system using efficient graph libraries or GPU-accelerated computations would also make it more suitable for real-time, large-scale deployment.

The recommendation techniques explored in this project have direct applications in various domains beyond movie platforms. Streaming services like Netflix, Hulu, and Spotify can use these models to personalize content suggestions. E-commerce platforms such as Amazon or eBay can leverage item-based or graph-based methods to recommend products based on user behavior. Social media platforms can use graph-based recommendation to suggest friends, pages, or groups by analyzing interaction networks. Furthermore, educational platforms like Coursera or Udemy can recommend courses using similar techniques based on a user's learning history or interests, enhancing user experience and retention across a wide range of industries.