Pixie-Inspired Random Walk Algorithms for Recommendations

1. Introduction to Graph-Based Recommendations

Pixie-inspired random walk algorithms are a modern, graph-based approach to recommendation that moves beyond simple user or item similarity. Instead of relying on direct, one-to-one comparisons, these systems model user–item interactions as a large, interconnected graph and simulate how users might "wander" through it. By starting a random walk from a specific user or movie, the algorithm discovers other movies that are "close" in the graph, not just in terms of direct similarity but also through multi-hop connections (e.g., user → movie → similar user → another movie).

This method is powerful because it can uncover diverse and serendipitous recommendations that traditional collaborative filtering might miss. It excels at balancing popularity with personalization, as the walk is guided by the underlying structure of user preferences. In this project, a Pixie-inspired model was implemented to explore these benefits on the MovieLens 100K dataset.

2. Implementation Details

The implementation followed a three-step process: data preparation and graph construction, building the weighted random walk mechanism, and creating recommendation functions for both user- and movie-based queries.

Step 1: Graph Construction

A bipartite graph was constructed to represent the relationships between users and movies.

1. **Data Preparation**: The ratings and movies datasets were merged to associate movie titles with ratings. If any user rated the same movie multiple times, these ratings were aggregated into a single mean rating.
2. **Rating Normalization**: To remove user bias (e.g., users who consistently rate higher or lower than others), each user's ratings were normalized by subtracting their personal average rating. This centres each user's ratings around zero, ensuring that the graph reflects relative preference rather than absolute scores. (ratings['rating'] = ratings.groupby('user_id')['rating'].transform(lambda x: x - x.mean()))
3. **Adjacency List Creation**: A Python dictionary was used to create an adjacency list, which is an efficient way to store a graph. For each rating, a bidirectional edge was created: the movie was added to the user's list of neighbors, and the user was added to the movie's list of neighbors. This creates a graph where users only connect to movies and vice-versa.

Step 2: Weighted Random Walk Mechanism

The "walk" itself is a probabilistic process. Instead of moving to any neighbor with equal probability, this implementation uses a **degree-weighted** approach.

1. **weighted_pick(neighbors) function**: This helper function was created to handle the biased random selection.
2. **Probability Calculation**: For a given node, it calculates the "degree" (number of connections) of each of its neighbors.
3. **Biased Selection**: It then selects the next node to visit with a probability proportional to its degree. This means popular movies or highly active users are more likely to be visited, which mimics real-world discovery patterns while still allowing for chance encounters with niche items.

Step 3: Recommendation Functions

Two main functions were implemented, demonstrating the flexibility of the graph-based approach:

1. **weighted_pixie_recommend(user_id, walk_length=15, num=5)**:
    - **Starts at a user node**.
    - Performs a random walk for a given walk_length.
    - Tracks how many times each **movie node** is visited.
    - Returns the top num most frequently visited movies.
2. **weighted_pixie_recommend_movie(movie_name, walk_length=10, num=5)**:
    - **Starts at a movie node** (found by its title).
    - Performs the same weighted random walk.
    - Returns the top num most visited movies, which are interpreted as "similar" or "related" movies.

By implementing both, the system can answer two different but equally important questions: "What should this user watch next?" and "What's a good movie to watch after this one?"

3. Results and Insights

The Pixie-inspired algorithm produced fascinating and diverse recommendations, highlighting its strengths over simpler collaborative filtering methods.

- **User-Based Random Walk (User 1, 15 steps)**: The recommendations for User 1 included a varied mix of genres and decades, such as *Arsenic and Old Lace (1944)*, *Chinatown (1974)*, and *Clueless (1995)*. This demonstrates the algorithm's ability to traverse different parts of the graph and surface serendipitous recommendations that a direct similarity search might not find.
- **Movie-Based Random Walk ("Jurassic Park (1993)", 10 steps)**: The recommendations included *In & Out (1997)*, *The Bridge on the River Kwai (1957)*, and *Die Hard: With a Vengeance (1995)*. This is a much more eclectic list than the one produced by item-based collaborative filtering (which recommended similar action blockbusters). It shows that the random walk found movies connected not just by genre, but by the diverse and sometimes unpredictable paths of audience taste.

Why This Approach is Effective:

- **Discovery Power**: The multi-hop nature of random walks allows the system to discover non-obvious connections between items. It can recommend a classic drama to a fan of modern action movies if there's a "path" of users connecting them.
- **Handles Sparsity Well**: Even if two users have no movies in common, they can still be connected in the graph through a series of intermediate users and movies.
- **Balances Popularity and Personalization**: The degree-weighted selection ensures that popular, high-quality items are more likely to be recommended (a desirable trait), but the "random" aspect of the walk means that less popular, niche items can still be discovered.

4. Conclusion

The implementation of a Pixie-inspired random walk algorithm was highly successful. It demonstrates a powerful and flexible alternative to traditional collaborative filtering. By modelling the user-movie ecosystem as a graph, it captures the complex, interconnected nature of user preferences far more effectively than simple similarity metrics. The results show that this approach can generate recommendations that are not only relevant but also diverse and surprising, which is a key factor in creating an engaging user experience. While it introduces a level of randomness, its ability to explore the "long tail" of the catalogue and uncover hidden gems makes it an invaluable technique for modern recommendation systems.