Movie Recommendation System Report

Zane Hutchens
20 Nov. 2025

Introduction

Recommendation systems have been a major research subject since their inception in the early 1990s, due to their extensible applications in business, education, medicine, entertainment, and more. Companies such as Spotify may use them to recommend new songs or podcasts to users, whereas instructors may use them to recommend personalized assignments or readings depending on a student's standing. In this report, I explore three major but simple recommender systems: User-user collaborative filtering, item-item collaborative filtering, and pixie-inspired weighted random walk graph-based recommendation.

Dataset

The dataset used in this paper is the widely-used benchmark dataset, MovieLens 100k. It has 100 thousand ratings from 1682 movies and 943 users. Within the overall dataset, there are three separate files: one for ratings, one for movies, and one for users. Starting with the rating set, there are 100000 ratings, with columns user_id, movie_id, rating, and timestamp. There was no preprocessing done on this dataset as no rows had missing values or were duplicated. However, the timestamp feature was not particularly useful for the listed recommendation systems and was not utilized. The movies dataset has columns movie_id, title, and release_date. This set did have missing values in the release date column, so that row is removed leaving us with 1681 movies. Again, the release_date feature is not particularly useful for the explore recommendation systems, so it is ignored. Finally, the users dataset has columns user_id, age, gender, and occupation. It does not have any missing values or duplicated rows, so it is left alone. The features of age, gender, and occupation are not utilized for this exploration.

Methodology

As stated previously, this project explores three types of recommendation systems:

- User-user collaborative filtering
- Item-item collaborative filtering
- Pixie-inspired random walk graph-based recommendations

User-user and item-item both use cosine similarity to measure pairwise similarity between users and items, where they are then sorted by most similar to least similar. Then, we take the top n similar items to generate n recommendations. Since the recommendations are movie ids, they are mapped back to movie titles using the existing movie dataset. Additionally, because the ratings matrix is sparse, in both implementations any missing ratings are 0, suggesting the user does not have a rating for that particular movie.

The pixie-inspired system uses weighted random walks to traverse the user-item graph, where the edge weights are indicated by normalized ratings. Since the edge weights are used to determine the probabilities of traversing to a specific connected node, weights with higher ratings will have a higher probability of being selected. This will lead to areas of the graph being explored that the user is likely to enjoy.

Implementation Details

The implementations of user-user collaborative filtering and item-item collaborative filtering are very similar, with minor differences occurring in each type. Starting with user-user collaborative filtering, the general steps are as follows:

- Start with a provided user_id to find recommendations for
- Find similar users using pairwise cosine similarity between users
- Sort similar users by ratings, excluding the given user_id
- Find similar users ratings using existing user-movie matrix of ratings
- Compute the average rating of the similar users, then sort in descending order
- Using the average ratings, retrieve top n recommendations
- Map those movie_ids to movie titles using the existing movie dataset
- Return dataframe of movie rankings and titled

The core idea of item-item recommendation is the same, with minor differences in item-item similarity:

- Compute item-item similarities using cosine similarity and the transpose of the user-movie rating matrix
    - Transpose is used here to swap columns with rows, as similarity between movies is required

- Given a movie id and number of recommendations to return:
    - Check if movie actually exists within the dataset
    - Find the most similar items from the created item-item similarity matrix
    - Sort similar items in descending order, removing the provided movie id
    - Take top n recommendations from that matrix
    - Map movie ids back to movie names using existing movie dataset
    - Return dataframe of movie rankings and titles

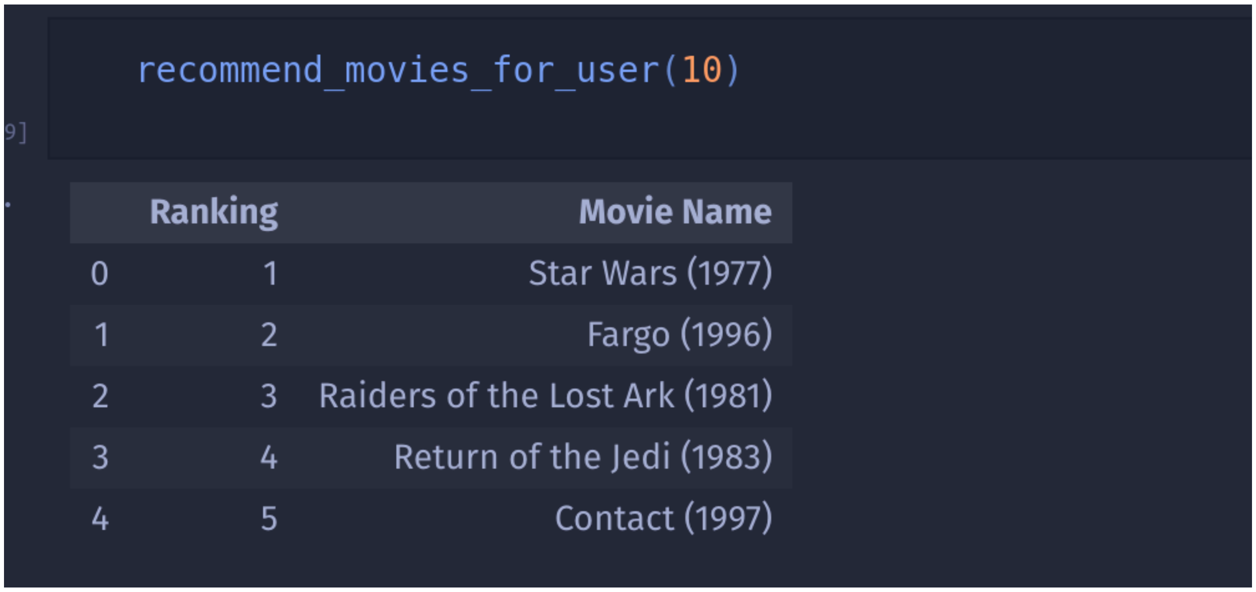
As for the pixie-inspired graph method, the user-item graph is built by merging the movies dataset onto the ratings dataset, providing a matrix with a user_id, movie_id, and rating. The graph is built using this matrix with the following steps:

- For each row in the movie-ratings matrix:
    - If the user id is not in the graph, add as a key with a set of empty nodes
    - If the movie id is not in the graph, add as a key with a set of empty nodes
    - Add the user to the graph with an edge containing the movie id and rating
    - Add the movie to the graph with an edge containing the user id and rating

Weighted random walks are performed using a given number of steps to complete and a user or movie id. A step is only completed if the current node is a movie, as user nodes are not considered when performing graph-based item-item recommendations. For the current node at timestep t, it is marked as visited with value 0, or using the existing visited score and adding one. Then the step count is increased by 1 to note that a movie node was visited. After this, to traverse to a new node the connected nodes and edge weights are considered. Each connected node has a weight (rating) associated with its edge. These weights are used as the probability of selecting the next node at timestep t + 1. However, if the sum of those weights are less than or equal to 0, we choose a new node uniformly. After N steps are reached, a final dictionary of movies and visit counts are returned, ranked from most visited to least visited.

Results and Evaluation

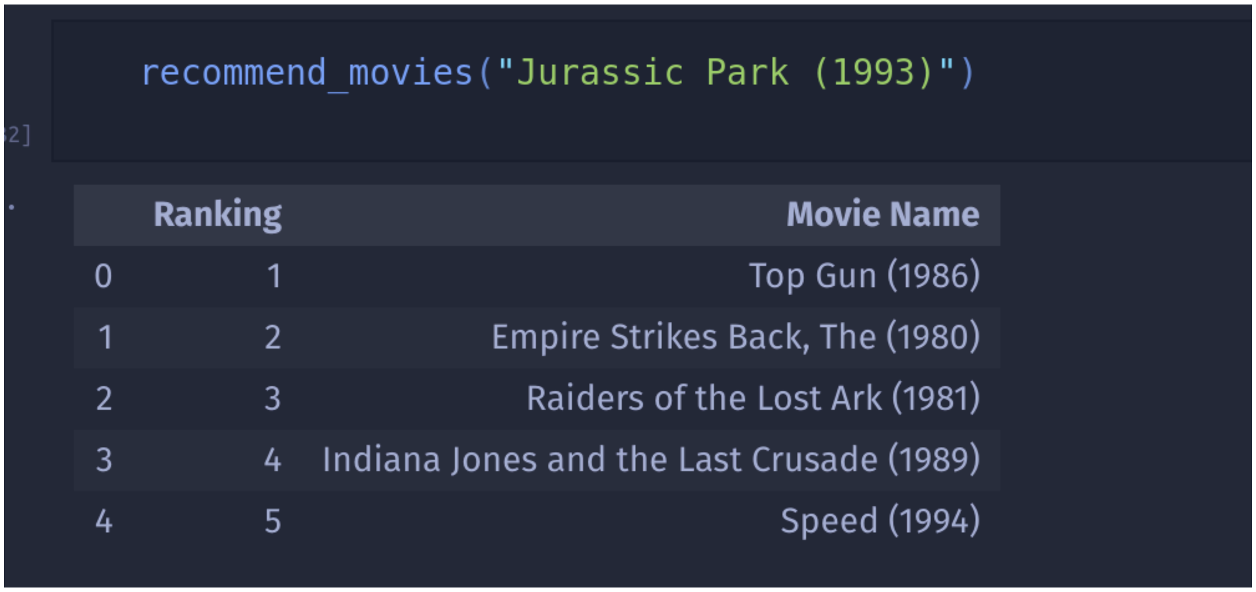
Here are some provided example outputs of the implemented recommendation systems. Starting with user-user collaboration filtering:



```
recommend_movies_for_user(10)
```

	Ranking	Movie Name
0	1	Star Wars (1977)
1	2	Fargo (1996)
2	3	Raiders of the Lost Ark (1981)
3	4	Return of the Jedi (1983)
4	5	Contact (1997)

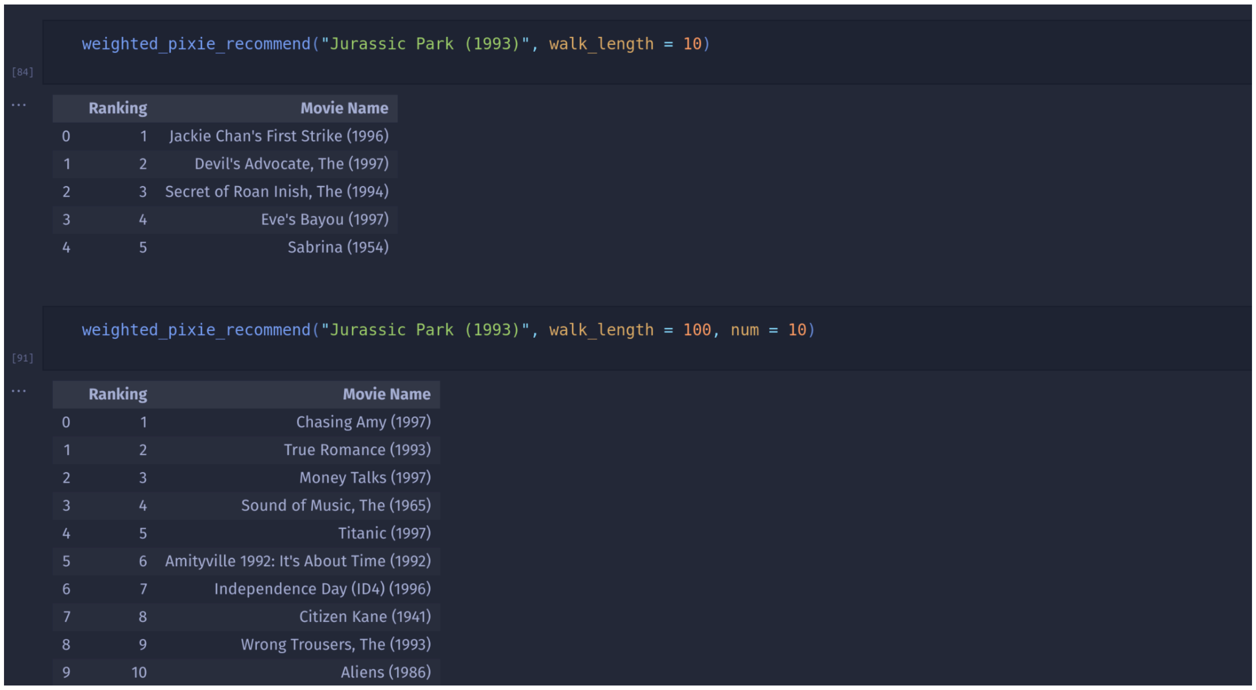
And item-item collaborative filtering:



```
recommend_movies("Jurassic Park (1993)")
```

	Ranking	Movie Name
0	1	Top Gun (1986)
1	2	Empire Strikes Back, The (1980)
2	3	Raiders of the Lost Ark (1981)
3	4	Indiana Jones and the Last Crusade (1989)
4	5	Speed (1994)

Due to the inherent randomness in a weighted random walk algorithm, multiple example outputs are shown with differing parameters. The movie name is kept unchanged to leave as little room for variation as possible.

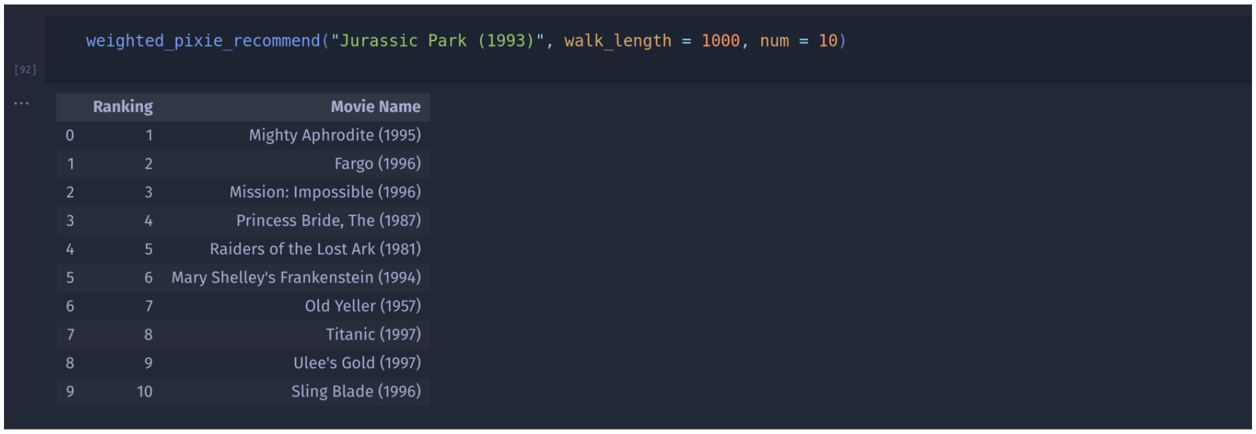


```
[84]
weighted_pixie_recommend("Jurassic Park (1993)", walk_length = 10)
```

	Ranking	Movie Name
0	1	Jackie Chan's First Strike (1996)
1	2	Devil's Advocate, The (1997)
2	3	Secret of Roan Inish, The (1994)
3	4	Eve's Bayou (1997)
4	5	Sabrina (1954)

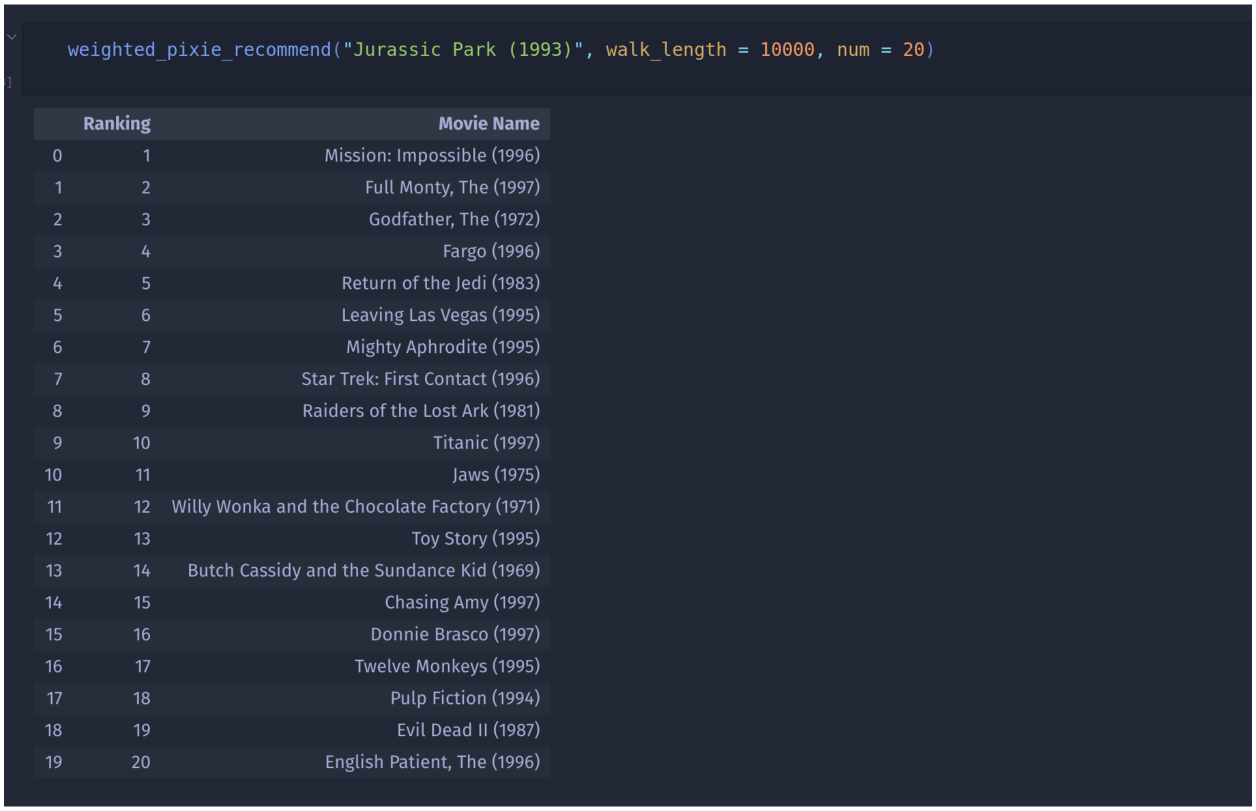
```
[91]
weighted_pixie_recommend("Jurassic Park (1993)", walk_length = 100, num = 10)
```

	Ranking	Movie Name
0	1	Chasing Amy (1997)
1	2	True Romance (1993)
2	3	Money Talks (1997)
3	4	Sound of Music, The (1965)
4	5	Titanic (1997)
5	6	Amityville 1992: It's About Time (1992)
6	7	Independence Day (ID4) (1996)
7	8	Citizen Kane (1941)
8	9	Wrong Trousers, The (1993)
9	10	Aliens (1986)



```
[92]
weighted_pixie_recommend("Jurassic Park (1993)", walk_length = 1000, num = 10)
```

	Ranking	Movie Name
0	1	Mighty Aphrodite (1995)
1	2	Fargo (1996)
2	3	Mission: Impossible (1996)
3	4	Princess Bride, The (1987)
4	5	Raiders of the Lost Ark (1981)
5	6	Mary Shelley's Frankenstein (1994)
6	7	Old Yeller (1957)
7	8	Titanic (1997)
8	9	Ulee's Gold (1997)
9	10	Sling Blade (1996)



```
weighted_pixie_recommend("Jurassic Park (1993)", walk_length = 10000, num = 20)
```

	Ranking	Movie Name
0	1	Mission: Impossible (1996)
1	2	Full Monty, The (1997)
2	3	Godfather, The (1972)
3	4	Fargo (1996)
4	5	Return of the Jedi (1983)
5	6	Leaving Las Vegas (1995)
6	7	Mighty Aphrodite (1995)
7	8	Star Trek: First Contact (1996)
8	9	Raiders of the Lost Ark (1981)
9	10	Titanic (1997)
10	11	Jaws (1975)
11	12	Willy Wonka and the Chocolate Factory (1971)
12	13	Toy Story (1995)
13	14	Butch Cassidy and the Sundance Kid (1969)
14	15	Chasing Amy (1997)
15	16	Donnie Brasco (1997)
16	17	Twelve Monkeys (1995)
17	18	Pulp Fiction (1994)
18	19	Evil Dead II (1987)
19	20	English Patient, The (1996)

There are multiple limitations across all three implementations, with the most crucial being the selected features to use. In a robust implementation, the user's age, gender, and occupation all should be considered, along with the release date of the movie and time of rating.

A person's preferences are not constant, so older ratings may not be a good indicator of a person's preferences months or years in the future. To mitigate this problem, you may assign a weight to rating based on the timestamp or release date, rather than using the pure rating.

Conclusion

This was the most fun assignment I attempted in this course. It was rather satisfying to see something I had only practiced with pen and paper be implemented and used with a standard dataset. I observed that while user-user and item-item implementations always gave the same recommendation on the same input, the weighted random walk system gave different results with the same input, even as the number of steps increased. I think this is most likely due to the inherent randomness of selecting a new node to go to inside the algorithm.

There are certainly limitations in all of the implementations, as mentioned in the results section. Although in this case we have an existing dataset to play with, companies starting out will not have that luxury. A hybrid model of using item-item filtering then swapping to user-user filtering once there are enough ratings may be useful in limiting a cold-start issue.

There are multiple applications of these algorithms, with the most obvious to recommend new items to users. However, I believe that they could be utilized in education to provide personalized sources for students based on their standing in the course. Recommender systems in medicine may also be worthwhile, where they support medical decisions in prescribing drugs or recommending treatments based on previous patient experiences.