Univerzitet u Sarajevu
Elektrotehnički fakultet
Sarajevo

DSAI
DATA SCIENCE & ARTIFICIAL INTELLIGENCE

Netflix Recommendation

Project Guide

Course: Algorithms and Data Structures 2

Professor: Doc. dr. Sead Delalić

Student: Emin Hadžiabdić

Academic Year: 2025/26

Student ID: 19960

Sarajevo, December 2th 2025

Contents

1. Dataset design and choice

Need two CSVs:

1. Ratings / interactions (core for graph) with columns:
- userId: integer user identifier
- movieId: integer movie identifier
- rating: numeric rating, e.g. 0.5–5.0
- timestamp: when rating was created
2. Movies metadata (for display and creativity) with columns:
- movieId: same IDs as ratings file
- title: string movie title
- genres: string with pipe-separated or comma-separated genres

These two tables will be used to:
- Build edges (from ratings)
- Show names/genres in UI (from movies)
- Add "original features" like genre filtering or genre-aware scoring

Dataset Links:
- https://www.kaggle.com/datasets/rishitjavia/netflix-movie-rating-dataset?select=Netflix_Dataset_Movie.csv
- https://www.kaggle.com/datasets/bipulnath98/movie-recommendation-dataset
- https://www.kaggle.com/datasets/rounakbanik/the-movies-dataset?select=movies_metadata.csv

2. Potential Algorithms

Work in a bipartite graph: users on one side, movies on the other; edges mean “user rated movie (liked)”.

1. Common Neighbors (CN) / 2‑hop paths
    - Interpret recommendation as link prediction: user–movie edge likely if they share many 2‑hop neighbors.
    - For user u and movie m, count all nodes reachable in 2 hops from both (i.e., common movies and/or users in the 2‑step neighborhood, depending how you define it in bipartite).
2. Jaccard Similarity on neighborhoods
    - Similar to CN but normalized.
    - For user u and movie m, score = $|\Gamma 2(u) \cap \Gamma 2(m)| / |\Gamma 2(u) \cup \Gamma 2(m)|$, where $\Gamma 2$ = 2‑hop neighbors.
3. Random Walk with Restart (RWR) or personalized PageRank
    - Start a random walk at user u; at each step:
        - with probability $1-\alpha$, move along a random edge
        - with probability α, jump back to user u
    - The stationary probability for each movie is its relevance score to user u.

Chosen Algorithm

I choose Jaccard on 2‑hop neighborhoods because:

- It uses graph structure directly
- It balances popularity and specificity
- It is easier to explain and debug than RWR but more robust than raw common neighbors.

4. Full graph construction steps (Python + NetworkX)

I will use pandas and networkx.

1. Load data
    - Read ratings.csv and movies.csv with pandas.
    - Filter to ratings >= 3.5 (or 4.0) to keep only positive “like” interactions.
2. Optionally downsample
    - Select a subset of users with enough ratings (e.g., at least 10 likes).
    - Filter ratings and movies accordingly.
3. Create bipartite graph

    Core ideas:
    - Use nx.Graph() (undirected)
    - Tag nodes with type: bipartite='user' or 'movie'
4. Steps:
    - Initialize graph: G = nx.Graph()
    - Add user nodes:
        - For each distinct userId, G.add_node(f"u_{userId}", bipartite="user")
    - Add movie nodes:
        - For each distinct movieId, G.add_node(f"m_{movieId}", bipartite="movie", title=..., genres=...)
    - Add edges:
        - For each rating row with rating >= threshold:
            - G.add_edge(f"u_{userId}", f"m_{movieId}", weight=rating)
5. Validate graph
    - Check counts: len(user_nodes), len(movie_nodes), G.number_of_edges()
    - Make sure edges only go user→movie (no user→user, movie→movie).

6. Store helper mappings
   - user_id_to_node, node_to_user_id
   - movie_id_to_node, node_to_movie_id

     Useful for going from integer IDs in the dataset to node names in the graph and back for UI.

5. Two concrete scoring methods

Scoring will be defined as a function: score(user_node, movie_node, G).

Method A: Jaccard on 2-hop neighborhoods (primary)

1. For a user node u, define Γ2(u), as:
   - All nodes at distance exactly 2 from u in the bipartite graph (this will be other users and movies).
2. For a movie node m, define Γ2(m) similarly.
3. Compute:
   - intersection_size = len(Gamma2(u) ∩ Gamma2(m))
   - union_size = len(Gamma2(u) ∪ Gamma2(m))
   - score = intersection_size / union_size (if union_size > 0 else 0)

Implement 2-hop neighbors by:

- Using nx.single_source_shortest_path_length(G, source=u, cutoff=2) and collecting nodes at distance 2.

Method B: Random Walk with Restart (secondary)

For a given user u:

1. Initialize probability vector p(0):
    - p(0)(u)=1, 0 for other nodes.
2. Iteratively update:
    - p(t+1)=αp(0)+(1−α)ATD−1p(t)
    - Where A is adjacency matrix, D degree matrix; α is restart probability (e.g., 0.15).
3. After convergence (or fixed number of iterations), use p(node) for movie nodes as scores.

Why I chose Jaccard as main score

- Simple to code and explain; requires only neighborhood sets and intersections.
- Handles popularity somewhat (because of normalization by union); avoids recommending only the most popular movies.
- Works very well on smaller graphs where interpretability is more important than raw accuracy.

6. High-level recommendation pipeline

For a selected user u:

1. Get all movies already rated/liked by u (neighbors of u in G).
2. Candidate movies = all movie nodes minus already-seen movies.
3. For each candidate movie m:
    - Compute score_Jaccard(u, m) and optionally score_CN(u, m) or score_RWR(u)[m].
4. Sort candidates by chosen main score (Jaccard).
5. Return top-N, e.g., top 10.

Also compute supporting stats:

- Number of common 2-hop neighbors
- Average rating among similar users (for use of ratings in scoring)

7. Web app architecture (Flask + HTML)

Use Flask in Python for a minimal but proper web interface.

Backend structure

- app.py:
  - Load data and build graph *once* at startup (global variables).
  - Define routes:
    - / (GET): show a page with:
      - Dropdown of user IDs
      - Options: number of recommendations, maybe genre filter
    - /recommend (POST):
      - Read selected user ID (and filters) from form
      - Run recommendation pipeline
      - Render a result template with recommended titles and scores
- templates/:
  - index.html: contains form and shows results in a table.

Frontend (HTML) basics

- A form with:
  - <select name="user_id"> listing user IDs or user display names
  - <input type="number" name="top_n" value="10">
  - Optional:
    - <select name="genre_filter"> with genres (All, Action, Comedy, …)

- Results:
    - Simple HTML table with columns:
        - Rank
        - Movie title
        - Score (Jaccard)
        - (Optional) Number of supporters / common neighbors
        - Genres

How backend talks to graph

In /recommend:

- Convert user_id from form to node "u_{id}"
- Call get_recommendations(user_node, top_n, genre_filter)
- That function:
    - Implements step 5 pipeline
    - Returns list of dicts:
        - {title, score, genres, support_count, movieId}

7. Graph visualization component

Use NetworkX + matplotlib or Plotly to draw a local ego-graph around the selected user.

Steps:

1. For selected user u, get:
    - Direct neighbors (movies already liked)
    - A few similar users (e.g., those who share many movies with u)
    - Top-K recommended movies
2. Build a subgraph with just these nodes and edges.
3. Draw:
    - Users = circles, one color (e.g., blue)

    - Movies user has seen = squares, another color (grey)
    - Recommended movies = squares, highlight color (orange or red), node size proportional to score.
4. Export as:
    - PNG or SVG generated in Python, saved to static/ directory
    - The /recommend handler regenerates this local visualization image for the requested user and passes the image URL to the template.

8. Original features

All of these are implementable with your dataset design:

1. Genre-aware score bonus
    - Compute the user's favorite genres from their liked movies.
    - When scoring movie m, after Jaccard score *s*:
        - If m's genres overlap with top user genres, multiply by 1.1 or add a small bonus.
2. Explanation text per recommendation
    - For each recommended movie, pick top 2–3 "supporter" users (those contributing to the intersection / common neighbors).
    - Show explanation like:
        - "Users 5, 12, 19 liked the same movies as you (Inception, Interstellar) and also liked this movie."
3. Comparison of two methods in UI
    - For each recommendation in the result table, show:
        - Column for Jaccard score (main)
        - Column for Common Neighbors (baseline)
    - Maybe color cells where methods disagree a lot.
4. Slider to choose "minimum rating" threshold
5. Switch between "use ratings as weights" vs. ignore weights
6. Small evaluation mode: hide a few of the user's movies and see if the algorithm can recover them as top recommendations (simple hit-rate).