UC San Diego

Movie Recommendation system using Graph Database

Sijin Lyu, Caroline Zhang, Tristan Cooper

Mar 17th, 2025

Contents

Introduction

Have you ever finished a mind-bending sci-fi thriller and immediately craved another movie with the same level of intrigue and excitement?

We’ve all experienced that moment of indecision—endlessly scrolling through dozens of titles, hoping to stumble upon a gem that truly resonates. Our system aims to eliminate the guesswork by automatically curating the perfect follow-up selection, tailored to your unique cinematic tastes.

Movie Recommendation System

UC San Diego

Why Do We Need This Recommendation System?

- Streaming platforms now host massive libraries, making it easy for viewers to feel overwhelmed
- Personalized recommendations help users discover new favorites that match their interests
- A targeted system ensures viewers spend more time watching and less time searching

Why It's Important?

- Increases user engagement and satisfaction, reducing churn
- Drives higher watch times and showcases a platform's diverse content catalogue
- Differentiates a streaming service in a highly competitive market

What Are We Using?

- **Core Technology:** Neo4j graph database for rapid, relationship-based querying (compared to traditional SQL)
- **Essential Data:** User watch history, ratings, movie metadata (genre, directors, tags), expandable to social media and demographics
- **Goal:** Deliver accurate, fast, and scalable recommendations that adapt to individual user preferences

Dataset Description

Full IMDb Dataset (1M+)

- A comprehensive dataset with over 1 million movies and TV shows.
- **key features**: titles, genres, release years, ratings, episode details, and cast information.
- **Key Data Files:**
  - **title.crew.tsv.gz:** Information on movie **directors and writers**.
  - **name.basics.tsv.gz:** Details of **actors, directors, and other crew members**, including birth years and primary professions

MovieLens 25M Dataset

- A dataset containing **25 million ratings and 1.1 million tag applications** across 62,423 movies.
- Collected from **162,541 users** between January 9, 1995 – November 21, 2019.
- Users rated at least 20 movies, but no demographic data is included.
- **Key Data files**:
  - **ratings.csv:** User ratings on a 5-star scale.
  - **tags.csv:** User-generated movie tags.
  - **movies.csv:** Movie titles and genres.
  - **links.csv:** Movie identifiers (IMDb, TMDb).
  - **genome-scores.csv & genome-tags.csv**: Tag genome for movie-tag relevance.

Methodology

IMDB Dataset

id	title	type	genres	averageRa	numVotes	releaseYear
tt0000009	Miss Jerry	movie	Romance	5.4	223	1894
tt0000147	The Corbe	movie	Document	5.3	555	1897
tt0000502	Bohemios	movie		4	22	1905
tt0000574	The Story (	movie	Action, Ad	6	978	1906
tt0000591	The Prodig	movie	Drama	5.6	31	1907
tt0000615	Robbery U	movie	Drama	4.3	28	1907
tt0000630	Hamlet	movie	Drama	3.2	33	1908
tt0000675	Don Quijot	movie	Drama	4.3	23	1908
tt0000679	The Fairylo	movie	Adventure	5.2	78	1908

Movielens Tag Dataset

userId	movieId	tag	timestamp
3	260	classic	1.44E+09
3	260	sci-fi	1.44E+09
4	1732	dark come	1.57E+09
4	1732	great dialo	1.57E+09
4	7569	so bad it's	1.57E+09
4	44665	unreliable	1.57E+09
4	115569	tense	1.57E+09
4	115713	artificial in	1.57E+09
4	115713	philosophi	1.57E+09

Movielens Ratings Dataset

userId	movieId	rating	timestamp
1	296	5	1.15E+09
1	306	3.5	1.15E+09
1	307	5	1.15E+09
1	665	5	1.15E+09
1	899	3.5	1.15E+09
1	1088	4	1.15E+09
1	1175	3.5	1.15E+09
1	1217	3.5	1.15E+09
1	1237	5	1.15E+09
1	1250	4	1.15E+09
1	1260	3.5	1.15E+09
1	1653	4	1.15E+09
1	2011	2.5	1.15E+09
1	2012	2.5	1.15E+09
1	2068	2.5	1.15E+09
1	[illegible]	[illegible]	[illegible]

Use SQL to combine the datasets

We took a number of steps to filter and aggregate the data. One example:

Selecting top 5000 most popular movies:

```
WITH TopMovies AS (
  -- Select Top 5000 movies based on numVotes
  SELECT l.movieId, i.id AS imdb_id, i.title, i.genres, i.averageRating, i.numVotes, i.releaseYear
  FROM imdb i
  JOIN links l ON i.id = CONCAT('tt', l.imdbId)
  WHERE i.numVotes IS NOT NULL
  ORDER BY i.numVotes DESC
  LIMIT 5000
),
```

Use SQL to combine the datasets

Considering only more active users:

```
UserFilter AS (
    -- Select only users who have rated at least 10 movies from TopMovies
    SELECT r.userId
    FROM ratings r
    JOIN TopMovies tm ON r.movieId = tm.movieId
    GROUP BY r.userId
    HAVING COUNT(r.movieId) >= 10
),
```

Use SQL to combine the datasets

Then the data is aggregated

```
FilteredData AS (
    SELECT DISTINCT ON (r.userId, r.movieId, r.timestamp) -- Keep only one row per timestamp
        r.userId,
        tm.imdb_id,  -- Use IMDb ID instead of movieId
        tm.title,
        tm.genres,
        tm.releaseYear,
        tm.averageRating,
        r.rating,
        r.timestamp,
        t.tag,
        g.relevance
.............
```

Use SQL to combine the datasets

…………..

```
FROM ratings r
   JOIN TopMovies tm ON r.movieId = tm.movieId
   JOIN UserFilter uf ON r.userId = uf.userId
   LEFT JOIN tags t ON r.movieId = t.movieId AND r.userId = t.userId
   LEFT JOIN genome_scores g ON r.movieId = g.movieId
   WHERE g.relevance >= 0.5  -- Only keep relevant tags
   AND t.tag IS NOT NULL      -- Remove NULL tags
   ORDER BY r.userId, r.movieId, r.timestamp, g.relevance DESC  -- Keep the highest relevance tag
)
```

Final SQL Dataset

UC San Diego

Contains userids, IMDBids, Movie Titles, Genres, Release Year, Average Rating, User Rating, timestamp, tags, keyword relevance, and Directors

264	tt1232829	21 Jump St	Action, Co	2012	7.2	5	1.54E+09	Channing T	0.988	Phil Lord, Christopher Miller
647	tt2965466	Last Shift	Horror, My	2014	5.8	3	1.55E+09	satanism	0.9925	Anthony DiBlasi
653	tt1568911	War Horse	Adventure,	2011	7.2	3.5	1.54E+09	animals - li	0.992	Steven Spielberg
2165	tt1961175	American /	Action, Thr	2017	6.2	1	1.53E+09	propagand	0.80725	Michael Cuesta
2691	tt1232829	21 Jump St	Action, Co	2012	7.2	4.5	1.44E+09	meta	0.988	Phil Lord, Christopher Miller
3394	tt4537896	White Boy	Crime, Dra	2018	6.5	3.5	1.56E+09	Matthew M	0.708	Yann Demange
3448	tt1232829	21 Jump St	Action, Co	2012	7.2	3.5	1.5E+09	undercove	0.988	Phil Lord, Christopher Miller
3941	tt1232829	21 Jump St	Action, Co	2012	7.2	3.5	1.36E+09	drugs	0.988	Phil Lord, Christopher Miller
3975	tt1568911	War Horse	Adventure,	2011	7.2	3.5	1.34E+09	Steven Spi	0.992	Steven Spielberg

Restructure into a Graph with Neo4j

We then transformed the data into a graph using Neo4j to better model the relationships between the data. This graph has nodes for users, movies, genres, directors, keywords, and tags. This also lets us map relationships such as:

RATED (User → Movie)

OF_GENRE (Movie → Genre)

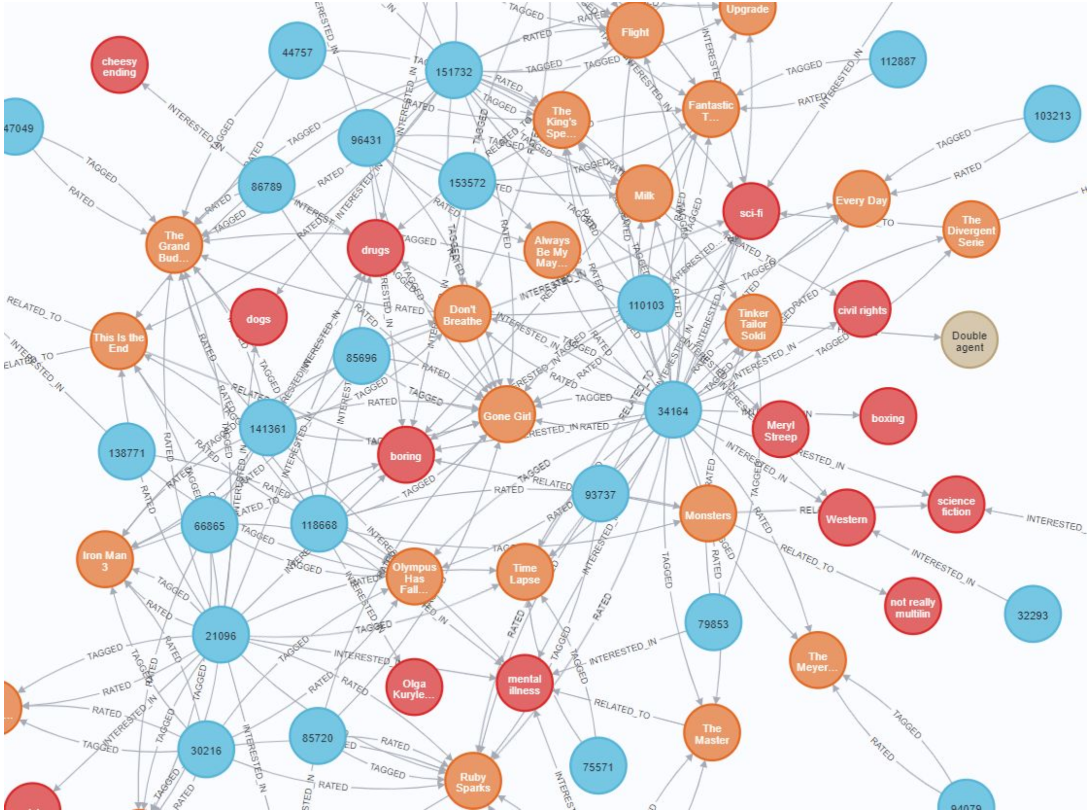
DIRECTED_BY (Movie → Director)

HAS_TAG (Movie → Tag)

RELATED_TO (Tag → Keyword)

INTERESTED_IN (User → Genre/Tag)

Final Graph


cheesy ending
44757
151732
Flight
Upgrade
112887
103213
47049
96431
The King's Spe...
Fantastic T...
86789
153572
Milk
sci-fi
Every Day
The Divergent Serie
The Grand Bud...
drugs
Always Be My May...
dogs
Don't Breathe
110103
Tinker Tailor Soldi
civil rights
Double agent
This Is the End
85696
Gone Girl
34164
Meryl Streep
boxing
138771
141361
boring
93737
science fiction
66865
118668
Monsters
Western
Iron Man 3
Olympus Has Fail...
Time Lapse
not really multilin
32293
21096
79853
The Meyer...
Olga Kuryle...
mental illness
30216
85720
Ruby Sparks
75571
The Master

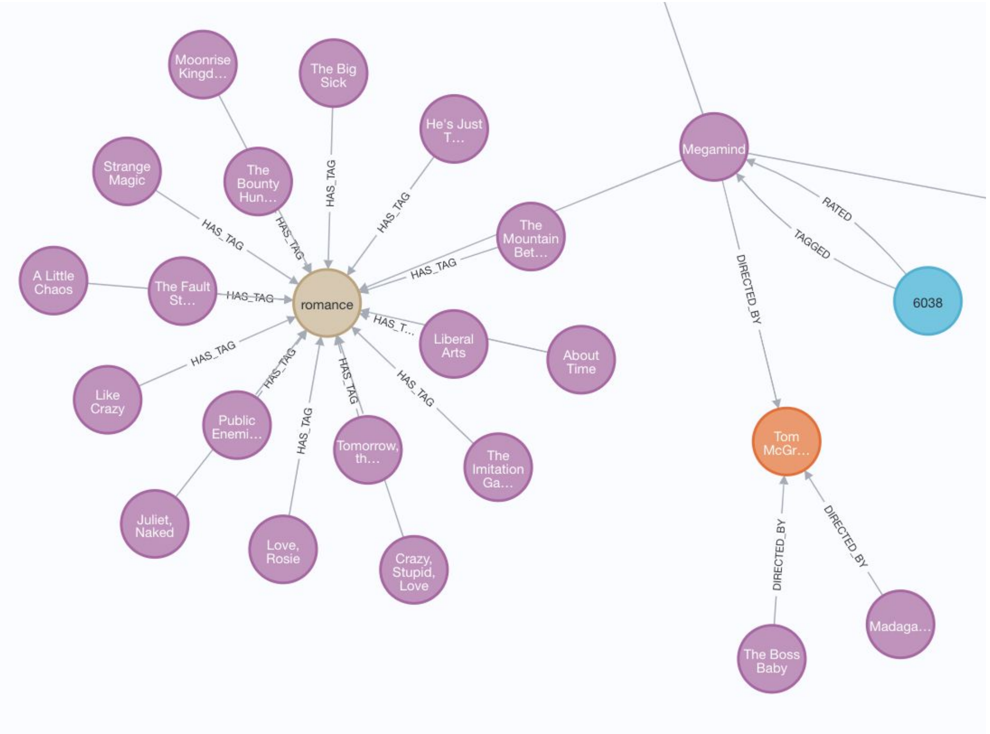
UC San Diego

Demonstration

How does the recommendation work?

Real-world Case

- **Use Case**: A streaming platform like Netflix or Amazon Prime wants to recommend movies to a user based on their viewing history and preferences.
- **How It Works**:
    - The system identifies movies the user has rated highly (e.g., "Inception" and "The Dark Knight").
    - It then finds similar movies based on shared genres (e.g., "Sci-Fi," "Action"), directors (e.g., Christopher Nolan), and tags/keywords (e.g., "Mind-bending," "Superhero").
    - The user receives recommendations like "Interstellar" or "Blade Runner 2049," which align with their interests.



Demo

“Recommend movies that the user has not yet seen by first identifying other users with similar tastes. Specifically, find users who've highly rated many of the same movies as the target user. Next, gather movies these similar users highly rated, but the target user hasn't watched. Finally, prioritize these candidate movies explicitly based on how closely their genres, directors, tags, and keywords match the user's established interests (content-based filtering). Return the most relevant and highly-ranked recommendations in dataset."

Code 1

```
MATCH (u:User {userId: '6038'})-[r1:RATED]->(m:Movie)
WHERE r1.rating >= 4.0
MATCH (u2:User)-[r2:RATED]->(m)
WHERE u2 <> u AND r2.rating >= 4.0
WITH u, u2, COUNT(DISTINCT m) AS similarity
ORDER BY similarity DESC
LIMIT 5

MATCH (u2)-[r3:RATED]->(candidate:Movie)
WHERE r3.rating >= 4.0 AND NOT EXISTS((u)-[:RATED]->(candidate))
WITH DISTINCT u, candidate
OPTIONAL MATCH (u)-[:RATED|TAGGED]->(likedMovie:Movie)
WHERE EXISTS((u)-[:RATED {rating: 4.0}]->(likedMovie)) OR EXISTS((u)-[:TAGGED]->(likedMovie))

MATCH
(likedMovie)-[:OF_GENRE|DIRECTED_BY|HAS_TAG|RELATED_TO]->(commonEntity)<-[:OF_GENRE|DIRECTED_BY|HAS_TAG|RELA
TED_TO]-(candidate)
WITH candidate, COUNT(DISTINCT commonEntity) AS contentScore,
COLLECT(DISTINCT commonEntity.name) AS matchedContent
```

Code 2

```
OPTIONAL MATCH (u)-[:INTERESTED_IN]->(interest:Keyword)<-[:RELATED_TO]-(candidate)
WITH candidate, contentScore, matchedContent,
COUNT(DISTINCT interest) AS interestScore,
COLLECT(DISTINCT interest.name) AS matchedInterests

WITH candidate, matchedContent, matchedInterests,
(contentScore + interestScore) AS finalRecommendationScore

RETURN candidate.title AS RecommendedMovie,
matchedContent AS MatchedContent,
matchedInterests AS MatchedInterests,
finalRecommendationScore AS TotalScore
ORDER BY TotalScore DESC
LIMIT 10;
```

Result

	RecommendedMovie	MatchedContent
1	"Slumdog Millionaire"	["social commentary", "violence", "emotional", "romance", "heartwarming", "great soundtrack", "based on a book", "love story", "cinematography", "visually appealir
2	"Up"	["funny", "rainy day watchlist", "comedy", "emotional", "Adventure, Animation, Comedy", "animation", "feel good movie", "Watched", "animated", "friendship", "borir
3	"Arrival"	["slow", "boring", "cinematography", "thought-provoking", "touching", "aliens", "visually appealing", "strong female lead", "plot twist", "overrated", "plot holes", "pre(
4	"Mad Max: Fury Road"	["surreal", "special effects", "great soundtrack", "cinematography", "sci-fi", "Action, Adventure, Sci-Fi", "visually appealing", "Bechdel Test:Pass", "dystopia", "chase
5	"The Social Network"	["funny", "witty", "soundtrack", "friendship", "adapted from:book", "imdb top 250", "cinematography", "good cast", "dark comedy", "visually appealing", "business",
6	"Prometheus"	["Watched", "Michael Fassbender", "philosophical", "horror", "atmospheric", "unresolved", "sci-fi", "unpredictable", "aliens", "script", "franchise", "genetics", "space

Unrated Movies

UC San Diego

"Identify movies that the user rated significantly higher than their general audience rating

	Movie	RoundedAverageRating	UserRating	RatingDifference
1	"They Came Together"	3.0	5.0	2.0
2	"Get Hard"	3.0	5.0	2.0
3	"21 Jump Street"	3.5	5.0	1.5
4	"That's My Boy"	3.0	4.5	1.5
5	"Unfinished Business"	2.5	4.0	1.5
6	"The Campaign"	3.0	4.5	1.5

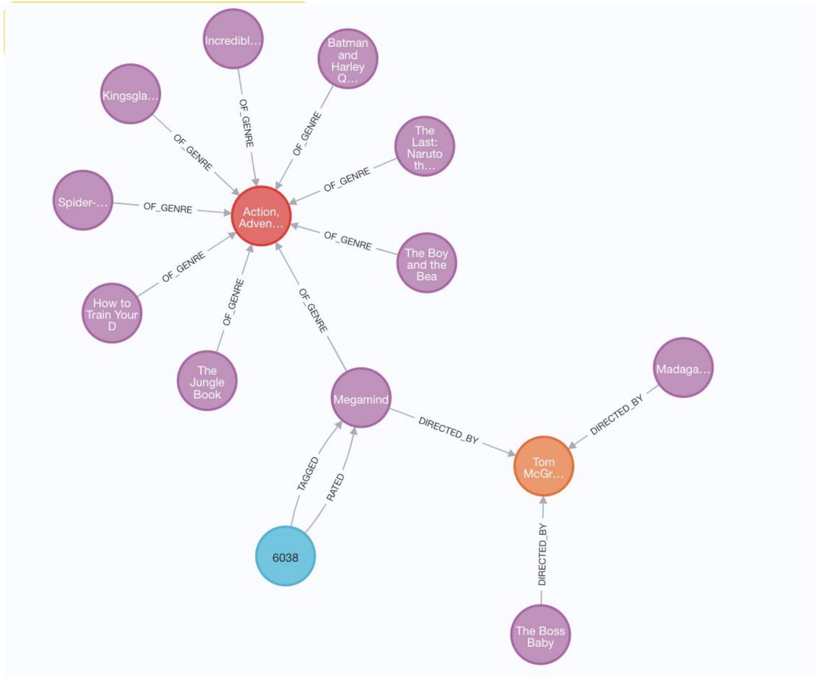
ORDER BY RatingDifference DESC;

Oscar reward

```
// Check tags
OPTIONAL MATCH (recMovie)-[:HAS_TAG]->(oscarTag:Tag)
WHERE oscarTag.name CONTAINS 'Oscar' OR oscarTag.name
CONTAINS 'Nominee' OR oscarTag.name CONTAINS 'Best
Picture'

// Assign a bonus score for Oscar-related tags
WITH recMovie, matchScore, matchedEntities, interestScore,
matchedKeywords,
    CASE WHEN oscarTag IS NOT NULL THEN 10 ELSE 0 END AS
oscarBonus

// Calculate total recommendation score with Oscar bonus
WITH recMovie, matchedEntities, matchedKeywords,
    (matchScore + interestScore + oscarBonus) AS totalScore
```



Discussion

Conclusion

- A Fast solution using Graph database
  - Neo4j (300~400ms) vs SQL (900~2s) for one single user
- Customizable Features
- The system can be extended to incorporate additional data sources (e.g., social network data, user demographics)

Limitations

UC San Diego

- Scalability for Larger Datasets
  - We pre-selected top 5000 movies to put into Neo4j
- Bias in Recommendations
- Future track:
  - Implement real-time recommendation updates based on user interactions (e.g., clicks, watch history).
  - Incorporate datasets with more features:
    - Actors
    - Abstracts

References

1. chandan-u. (2016). GitHub - chandan-u/graph-based-recommendation-system: building a recommendation system using graph search methodologies. We will be comparing these different approaches and closely observe the limitations of each. Retrieved March 19, 2025, from GitHub website: https://github.com/chandan-u/graph-based-recommendation-system/tree/master
2. GroupLens. (2019, December 10). MovieLens 25M Dataset. Retrieved from GroupLens website: https://grouplens.org/datasets/movielens/25m/
3. IMDb. (2023). IMDb Non-Commercial Datasets. Retrieved from Imdb.com website: https://developer.imdb.com/non-commercial-datasets/
4. Movie Recommender System. (2025). Retrieved March 19, 2025, from Github.io website: https://raveenak96.github.io/movies/
5. OctopusTeam. (2024). Full IMDb Dataset (1M+). Retrieved March 19, 2025, from Kaggle.com website: https://www.kaggle.com/datasets/octopusteam/full-imdb-dataset

Thank you!