Project Report: Graphic Database Movie Recommendation System

Caroline Zhang, Sijin Lyu, Tristan Cooper

Introduction

Modern streaming platforms offer vast libraries of films and television series. Traditional recommendation systems have largely relied on metadata-based filtering, collaborative filtering, or hybrid models, often failing to capture the deeper contextual relationships between movies, genres, and user interactions. Many earlier approaches, such as Chandan-U's graph-based recommendation system (2016) [chandan u(2016)] and RaveenaK96's movie recommendation system (2025) [mov(2025)], have explored the potential of graph-based methodologies. However, a major limitation of their implementations is their inability to effectively capture the complex dependencies among films and user preferences, as well as their lack of customizability.

To address these challenges, we introduce an advanced graph-based movie recommendation system using Neo4j. Unlike conventional SQL-based relational models or previous graph-based approaches, our system constructs a highly customizable and context-aware knowledge graph that integrates relational data from the MovieLens 25M and IMDb Non-Commercial Datasets. This paper presents a new graph-based recommendation system that optimizes query performance, enhances contextual relevance, and improves customizability.

Dataset Description

This project integrates data from two major sources: the MovieLens 25M Dataset [GroupLens(2019)] and the IMDb Non-Commercial Datasets [IMDb(2023)]. These datasets are combined to create a comprehensive relational dataset, which is subsequently transformed into a graph database to support analytical tasks in the field of movie recommendation systems.

MovieLens 25M Dataset

The primary dataset used in this project is the MovieLens 25M Dataset, publicly available from GroupLens Research at the University of Minnesota [GroupLens(2019)]. This dataset captures detailed user interactions collected from the MovieLens recommendation service between January 9, 1995, and November 21, 2019. It comprises the following components:

1. `Ratings`: A total of 25,000,095 individual user ratings for 62,423 movies, contributed by 162,541 distinct users. Ratings are provided on a 5-star scale with half-star increments.
2. `Tags`: A collection of 1,093,360 free-text tags applied by users to movies, reflecting personalized annotations and categorizations.
3. `Movie Metadata`: Information about movie titles, release years, and genre classifications for each film.
4. `Links`: Identifiers that map MovieLens movie entries to external databases, such as IMDb and TMDb, enabling cross-referencing of movie metadata.
5. `Tag Genome`: A precomputed dense tag-movie matrix that links movies to tags based on relevance scores. These scores are derived using machine learning methods applied to user-generated content, including textual reviews, tags, and ratings.

IMDb Comprehensive Dataset (Kaggle)

To enrich the dataset with additional metadata, we utilized the full IMDb dataset, available via Kaggle [OctopusTeam(2024)]. This dataset is a pre-processed version of the IMDb Non-Commercial Datasets [IMDb(2023)] and includes detailed metadata such as movie IDs, titles, genres, runtime, release dates, user ratings, and the number of votes.

IMDb Non-Commercial Datasets

While the Kaggle dataset provides extensive movie metadata, it lacks comprehensive details about directors. To address this limitation, we incorporated data from the IMDb Non-Commercial Datasets. The following files were used to enhance the dataset with director-related information:

- **title.crew.tsv.gz**:
  - `tconst`: A unique alphanumeric identifier for movie titles.
  - `directors`: An array of director identifiers (`nconst`).
  - `writers`: An array of writer identifiers (`nconst`).
- **name.basics.tsv.gz**: Provides detailed personnel information, including:
  - `nconst`: A unique alphanumeric identifier for each individual (e.g., director, writer, actor).
  - `primaryName`: The credited name of the person.
  - `birthYear` and `deathYear`: The years of birth and death (if applicable).
  - `primaryProfession`: The main professions of the individual.
  - `knownForTitles`: Prominent titles associated with the individual.

Methodology

We aim to construct a structured relational dataset using graph-based modeling to create a unified dataset. In the relational database, SQL queries were used to join tables such as *ratings*, *movies*, *tags*, *genome_scores*, *imdb*, and *title_crew*, facilitating the extraction of user preferences, movie metadata, and tag relevance scores.
Next, in the graph database, Neo4j was employed to model relationships between users, movies, genres, directors, tags, and keywords. This approach captures complex semantic relationships, such as shared genres or director preferences, enhancing the system's ability to understand user interactions and movie characteristics.

Technical Implementation

To organize and process the data effectively, a relational database schema was designed using tables derived from the MovieLens 25M Dataset and the IMDb Non-Commercial Datasets. From MovieLens, we created six tables: *ratings*, *movies*, *tags*, *links*, *genome_tags*, and *genome_scores* to store user information, rating timestamps, and review tags. From the IMDb Non-Commercial Datasets, we created three tables: *imdb*, *title_crew*, and *name_basics* to store movie metadata and director information.

Foreign key constraints were applied to ensure data integrity across related tables. Additionally, to optimize query performance, indexes were created on frequently queried columns, such as user IDs (userId), movie IDs (movieId), and IMDb IDs (imdbId).

SQL Data Filtering and Aggregation

Since the original dataset is large and Neo4j has processing limitations, we decided to preprocess the data, limiting the total number of rows to $50,000$ to ensure the dataset remained both meaningful and manageable. This was achieved through a series of filtering and aggregation steps.

The process began by selecting the top $5,000$ movies based on the number of votes (numVotes) from the IMDb dataset. The primary key used for joining tables is stored in the *links* table with *tag_id* and *movie_id*. We first joined the *imdb* and *links* tables and ordered the results in descending order of numVotes. Next, users who had rated at least 10 movies from this subset were retained to ensure that only active users with sufficient interaction data were included. This was accomplished using a `GROUP BY` clause and a `HAVING` condition, as shown in Listing 1.

Listing 1: Selecting Top 5000 Movies

```sql
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;

SELECT r.userId
FROM ratings r
JOIN TopMovies tm ON r.movieId = tm.movieId
GROUP BY r.userId
HAVING COUNT(r.movieId) >= 10;
```

To focus on the most significant user-generated content, only tags with a relevance score of at least 0.5 were retained. Finally, the dataset was enriched with director information by joining the filtered data with the *movie_directors* table. Missing director names were replaced with the placeholder `"Unknown"` to prevent errors. The following snippet illustrates this integration:

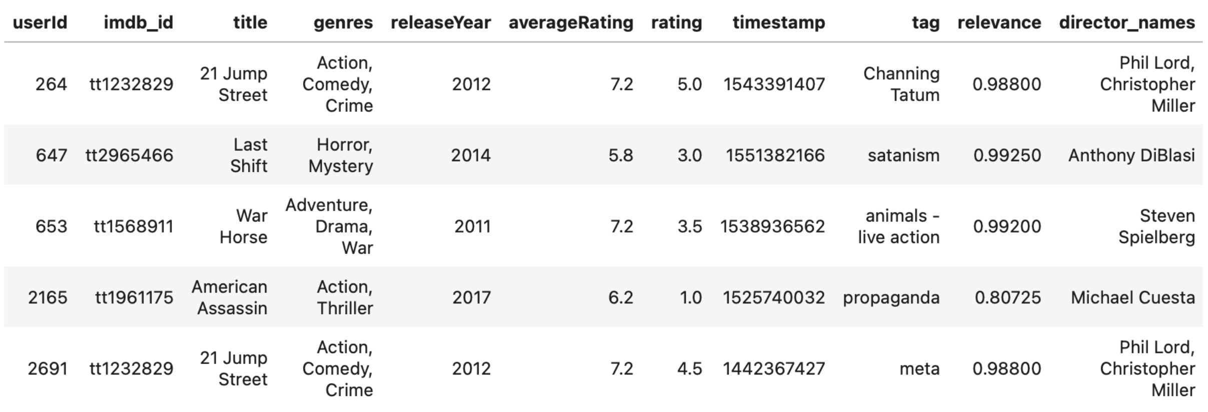
Listing 2: Enriching with Director Information

```sql
SELECT
    f.userId,
    f.imdb_id,
    f.title,
    f.genres,
    f.releaseYear,
    f.averageRating,
    f.rating,
    f.timestamp,
    f.tag,
    f.relevance,
    COALESCE(d.director_names, 'Unknown') AS director_names
FROM FilteredData f
LEFT JOIN movie_directors d ON f.imdb_id = d.tconst;
```

The final dataset consists of 49,500 rows, containing the following columns: userId, imdb_id, title, genres, releaseYear, averageRating, rating, timestamp, tag, relevance, and director_names. Figure 1 displays the top few rows of this dataset.

Graph Database (Neo4j) Transformation

We transformed the relational data into a graph database using Neo4j. Our model consists of six node types and seven relationship types to capture complex semantic relationships.



userId	imdb_id	title	genres	releaseYear	averageRating	rating	timestamp	tag	relevance	director_names
264	tt1232829	21 Jump Street	Action, Comedy, Crime	2012	7.2	5.0	1543391407	Channing Tatum	0.98800	Phil Lord, Christopher Miller
647	tt2965466	Last Shift	Horror, Mystery	2014	5.8	3.0	1551382166	satanism	0.99250	Anthony DiBlasi
653	tt1568911	War Horse	Adventure, Drama, War	2011	7.2	3.5	1538936562	animals - live action	0.99200	Steven Spielberg
2165	tt1961175	American Assassin	Action, Thriller	2017	6.2	1.0	1525740032	propaganda	0.80725	Michael Cuesta
2691	tt1232829	21 Jump Street	Action, Comedy, Crime	2012	7.2	4.5	1442367427	meta	0.98800	Phil Lord, Christopher Miller

Figure 1: Sample dataset of the final relational dataset

To optimize query performance in Neo4j, indexes were created on key properties. The integration of data into the Neo4j graph database involved extracting and transforming structured and semi-structured data into nodes and relationships. For example, movie genres, stored as pipe-separated strings (e.g., ”Action|Sci-Fi|Adventure”), were split into individual genre names. Each genre was then mapped to a *Genre* node, and a relationship of type *OF_GENRE* was established between the movie and its genres. The following Cypher query demonstrates this process:

Listing 3: Handling Genres

```
WITH u, m, row, split(row[3], '|') AS genres
UNWIND genres AS genre_name
MERGE (g:Genre {name: trim(genre_name)})
MERGE (m)-[:OF_GENRE]->(g)
```

We also split director names, stored as comma-separated strings (e.g., ”Christopher Nolan, Denis Villeneuve”), into individual names. Each director was mapped to a *Director* node, and a relationship of type *DIRECTED_BY* was established between the movie and its directors. Similarly, user-generated tags, such as ”mind-bending” or ”Oscar Nominee,” were extracted and mapped to *Tag* nodes. A relationship of type *TAGGED* was established between the user and the movie, capturing the timestamp of the tagging event. Additionally, a relationship of type *HAS_TAG* was created between the movie and the tag.

Tags with a relevance score of at least 0.5 were mapped to *Keyword* nodes. A relationship of type *RELATED_TO* was established between the movie and the keyword. The following Cypher query demonstrates this process:

Listing 4: Handling Keywords

```
WITH u, m, row
WHERE row[8] IS NOT NULL AND row[9] IS NOT NULL AND toFloat(row[9]) >= 0.5
MERGE (k:Keyword {name: trim(row[8])})
MERGE (m)-[:RELATED_TO {relevance: toFloat(row[9])}]->(k)
```

Table 1 provides an overview of the result nodes and relationships in the Neo4j graph database.

Table 1: Nodes and Relationships in the Neo4j Graph Database

Node Type	**Relationship Type**	**Description**
User	*RATED* (User → Movie)	Represents user ratings for movies, including the rating value and timestamp.
User	*TAGGED* (User → Movie)	Captures user-generated tags assigned to movies.
Movie	*DIRECTED_BY* (Movie → Director)	Links movies to their directors.
Movie	*OF_GENRE* (Movie → Genre)	Links movies to their associated genres.
Movie	*HAS_TAG* (Movie → Tag)	Links movies to user-generated tags.
Movie	*RELATED_TO* (Movie → Keyword)	Links movies to keywords derived from the tag genome relevance score.
User	*INTERESTED_IN* (User → Keyword)	Captures user interests based on tagging patterns and interactions with keywords.
Genre	—	Represents movie genres (e.g., Sci-Fi, Drama, Action).
Director	—	Represents directors, uniquely identified by nconst.
Tag	—	Represents user-generated movie tags.
Keyword	—	Represents keywords derived from tags, capturing semantic themes (e.g., ”time travel”, ”virtual reality”).

Experiment

We first conducted a demo to verify the structure of the graph database using the following Cypher query:

Listing 5: Retrieving user preferences from Neo4j

```
MATCH (u:User {userId: '6038'})-[r:RATED]->(m:Movie)
RETURN u, r, m
ORDER BY r.rating DESC
LIMIT 10;
```

The results are shown in Fig. 2.

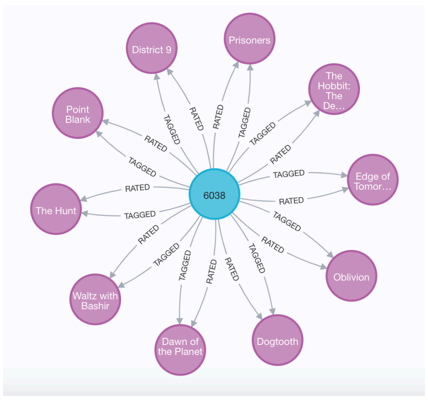


Figure 2: Sample graphical result for user 6038

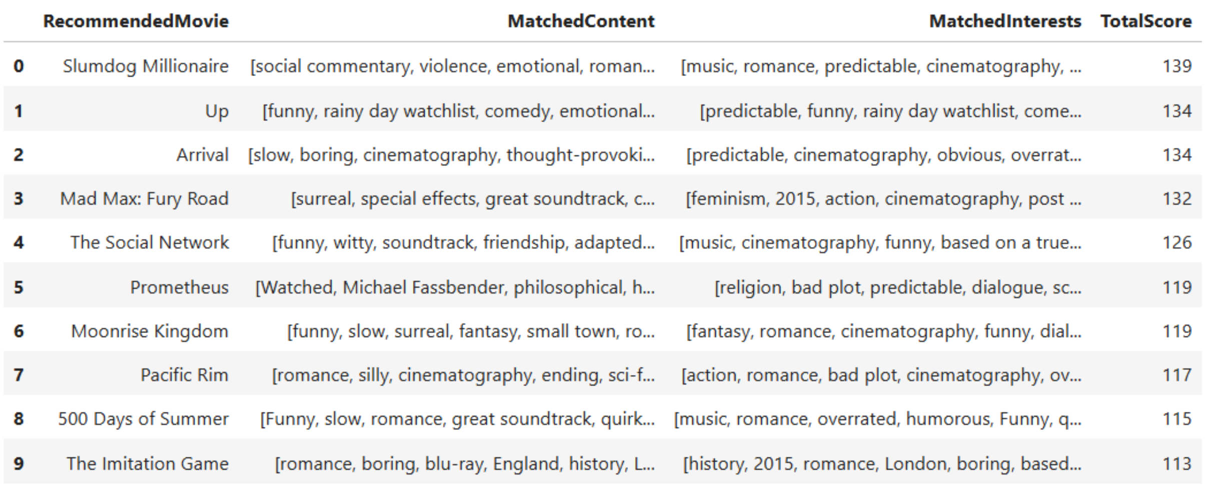
We also conducted several experimental demos to evaluate the effectiveness of the recommendation system. The experiments focused on three key use cases: collaborative and content-based filtering, identifying underrated movies, and prioritizing Oscar-nominated movies. Each use case was designed to address specific user needs and preferences, considering real-world scenarios.

1. Collaborative and Content-Based Filtering

One use case would be let the system recommends movies that the user has not yet seen by combining collaborative filtering and content-based filtering. First, it identifies users with similar tastes by finding those who have highly rated many of the same movies as the target user. Next, it gathers movies that these similar users have highly rated but the target user has not watched. Finally, it prioritizes these candidate movies based on how closely their genres, directors, tags, and keywords match the target user's established interests.

By using both user behavior (collaborative filtering) and movie metadata (content-based filtering), the system generates highly relevant recommendations. For example, if a user frequently watches and rates science fiction movies directed by Christopher Nolan, the system suggests similar movies that correspond with these preferences.

Specifically, we implement a hybrid approach combining collaborative and content-based filtering in Neo4j using Cypher queries. First, we identify a user's highly rated movies (`rating >= 4.0`) and find the top five most similar users based on shared preferences. Next, we extract movies these similar users have rated highly but that the target user has not yet seen. To refine recommendations, we apply content-based filtering by matching candidate movies with previously liked ones via shared attributes such as genres, directors, and tags. This forms a *content similarity score*. Additionally, we incorporate user interests by linking candidates to keywords the user has interacted with, computing an *interest alignment score*. The final recommendation score is computed as the sum of these scores, ranking movies in descending order. The result of user 6038 shown in Fig 3.



	RecommendedMovie	MatchedContent	MatchedInterests	TotalScore
0	Slumdog Millionaire	[social commentary, violence, emotional, roman...	[music, romance, predictable, cinematography, ...	139
1	Up	[funny, rainy day watchlist, comedy, emotional...	[predictable, funny, rainy day watchlist, come...	134
2	Arrival	[slow, boring, cinematography, thought-provoki...	[predictable, cinematography, obvious, overrat...	134
3	Mad Max: Fury Road	[surreal, special effects, great soundtrack, c...	[feminism, 2015, action, cinematography, post ...	132
4	The Social Network	[funny, witty, soundtrack, friendship, adapted...	[music, cinematography, funny, based on a true...	126
5	Prometheus	[Watched, Michael Fassbender, philosophical, h...	[religion, bad plot, predictable, dialogue, sc...	119
6	Moonrise Kingdom	[funny, slow, surreal, fantasy, small town, ro...	[fantasy, romance, cinematography, funny, dial...	119
7	Pacific Rim	[romance, silly, cinematography, ending, sci-f...	[action, romance, bad plot, cinematography, ov...	117
8	500 Days of Summer	[Funny, slow, romance, great soundtrack, quirk...	[music, romance, overrated, humorous, Funny, q...	115
9	The Imitation Game	[romance, boring, blu-ray, England, history, L...	[history, 2015, romance, London, boring, based...	113

Figure 3: Sample output of user 6038 by recommendation query

2. Identifying Underrated Movies

The system can also identifies movies that the user has rated significantly higher than their general audience rating. For instance, if a movie typically receives an average rating of 3.0 stars but the user rates it as 4.5 stars, the system infers that the user perceives this movie as underrated.

This approach helps uncover hidden gems that corresponding with the user's unique tastes. By comparing the user's ratings with the average ratings from the general audience, the system effectively highlights underrated movies. For example, if a user rates a lesser-known indie film significantly higher than its average rating, the system recommends similar indie movies that the user might enjoy.

To identify movies that a user perceives as underrated, we analyze their ratings in relation to adjusted average scores. First, we retrieve movies rated by the user and convert the global average rating from a 10-point scale to a 5-point scale. Next, we explicitly round the converted rating to the nearest 0.5 increment for consistency. We then filter movies where the user's rating is at least 1.0 point higher than this rounded average, highlighting significant deviations. Finally, we rank these movies by rating difference, returning those the user considers notably better than general consensus. This approach refines personalized recommendations by emphasizing unique preferences.

3. Prioritizing Oscar-Nominated Movies

Moreover, the system can recommend movies that have been nominated for or won prestigious awards such as the Oscars. By checking for tags like ”Oscar Nominee” or ”Best Picture,” the system assigns a bonus score to these movies, ensuring they are ranked higher in the recommendations.

First, we calculate a *content-based match score* by identifying movies that share key attributes (*genres, directors, tags, or related entities*) with highly rated movies ($\texttt{rating} \geq 4.0$) while ensuring they are unseen by the user. The number of shared attributes forms the initial *match score*. Next, we refine recommendations with an *interest-based match score* by incorporating keywords the user has previously interacted with. The count of such keywords contributes to the *interest score*. To highlight critically acclaimed films, we check for *Oscar-related tags* such as ”Oscar,” ”Nominee,” or ”Best Picture,” assigning a fixed *Oscar bonus* if a match is found. Finally, we compute the *final recommendation score* as the sum of the content match, interest match, and Oscar bonus scores. Movies are ranked in descending order, ensuring high-relevance recommendations tailored to both the user's preferences and critical recognition.

Discussion

We observed that using a graph database significantly improves the performance of the recommendation system. For a single user, querying the graph database takes approximately 300–400 milliseconds, compared to 900 milliseconds to 2 seconds for a traditional SQL database. This performance advantage is due to Neo4j's ability to efficiently traverse relationships, such as user preferences, movie genres, and director associations, without requiring complex joins. Thus, the graph-based approach is particularly well-suited for recommendation systems.

Additionally, our database supports a highly customizable system that can dynamically adapt to various recommendation strategies, as demonstrated in our demo. Users can fine-tune recommendation parameters, such as adjusting the weighting of content-based versus interest-based filtering, modifying the threshold for highly rated movies, or prioritizing award-winning films. Our demo also showcases real-time adjustments, allowing users to explore how changes in their tagging behavior or rating history impact recommendations.

Limitations

Despite its advantages, the system has some limitations that need to be addressed. First, to ensure efficient performance, the system currently pre-selects the top 5,000 movies based on the number of votes. While this approach works well for smaller datasets, scaling to larger datasets may require additional optimizations. Another challenge is bias in recommendations. The system may reinforce existing biases by favoring popular movies over lesser-known ones.

To address these challenges, we propose several future improvements. First, implementing real-time recommendation updates based on user interactions, such as clicks or watch history, would allow the system to dynamically adapt to user preferences. Second, expanding the dataset to include additional features, such as actors and movie abstracts, could provide richer and more diverse recommendations. Finally, introducing advanced evaluation metrics would help assess the quality of recommendations and further refine the system's performance.

Conclusion

This project demonstrates the effectiveness of a graph-based movie recommendation system that integrates content-based, interest-based, and award-based filtering. By using Neo4j, our model builds relationships between users, movies, genres, directors, and tags. The system can efficiently identify similar users, analyzes shared content attributes, and incorporates user interests to refine suggestions. While the system performs well for smaller datasets and provides highly relevant recommendations, there are opportunities for improvement in scalability, bias mitigation, and feature enrichment. By addressing these limitations and incorporating additional data sources, the system can be further enhanced to provide even more accurate and diverse recommendations.

Acknowledgment

We are really grateful to Prof. Amarnath Gupta, University of California, San Diego for his continuous support, encouragement, and insightful ideas in this class. Many of ideas in this review are proposed and taught by him during the class. I also genuinely appreciate support from TA Zixi Chen, who help us throughout the quarter and answering our questions.

References

[mov(2025)] 2025. https://raveenak96.github.io/movies/

[chandan u(2016)] 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. https://github.com/chandan-u/graph-based-recommendation-system/tree/master

[GroupLens(2019)] GroupLens. 2019. MovieLens 25M Dataset. https://grouplens.org/datasets/movielens/25m/

[IMDb(2023)] IMDb. 2023. IMDb Non-Commercial Datasets. https://developer.imdb.com/non-commercial-datasets/

[OctopusTeam(2024)] OctopusTeam. 2024. Full IMDb Dataset (1M+). https://www.kaggle.com/datasets/octopusteam/full-imdb-dataset