Movie Recommendation System: Building and Validating a Predictive Rating System Using Neo4j and the MovieLens Dataset

Paola Maria Lepore
Federica Sfeir

12^{th} November 2025

This report presents the design, implementation, and evaluation of a Movie Recommendation System developed using the MovieLens dataset and the Neo4j graph database. Recommendation systems play a central role in modern personalized content delivery across fields such as entertainment, e-commerce, and social media.

This project adopts a graph-based architecture to implement the system, leveraging the rich structure and semantic meaning of relationships between users, movies, and genres. By representing the MovieLens dataset as an interconnected network of nodes (users, movies, genres) and edges (ratings, belonging to a genre), the system can capture complex multi-hop dependencies and provides a transparent framework for understanding the recommendation procedure. By leveraging Cypher queries and graph-based similarity computations, the model infers user affinities and predicts ratings for unseen movies, combining insights from both collaborative filtering and content-based features such as genre preferences.

The performance of the proposed system is then evaluated by comparing predicted ratings with actual user evaluations using standard metrics such as RMSE and MAE. This work aims to show the potential and robustness of Neo4j in building recommendation systems, combining accuracy with explainability and scalability.

1 Exploratory Data Analysis (EDA) and Graph Structure Exploration

After importing the `movies.csv` and `ratings.csv` datasets into Neo4j, we performed a series of Cypher queries to analyze the structure and content of the graph database.

1.1 Data Ingestion and Schema Definition

The first step of the analysis consists of importing the dataset containing movie ratings, which consists of two .csv files, *movies.csv* and *ratings.csv*, obtained from the Grouplens repository, `https://grouplens.org/datasets/movielens/latest`. These files respectively contain metadata about movies (including identifiers, titles, and genres) and user ratings. Before loading data, unique constraints are created on the `:Movie` and `:User` nodes to ensure data integrity and prevent duplicate entries. These constraints define `m.id` and `u.id` as unique identifiers for movies and users, respectively.

```
CREATE CONSTRAINT FOR (m:Movie) REQUIRE m.id IS UNIQUE;
CREATE CONSTRAINT FOR (u:User) REQUIRE u.id IS UNIQUE;
```

The *movies.csv* file is first imported. For each row, a `:Movie` node is created, and its associated genres are parsed and standardized to uppercase format. The genres are then linked to the corresponding movie nodes through `:HAS_GENRE` relationships, capturing multi-genre associations.

```
LOAD CSV WITH HEADERS FROM  file:///movies.csv  AS line WITH line,
SPLIT(line.genres,  | ) AS Genres CREATE (m:Movie { id:
TOINTEGER(line.`movieId`), title: line.`title` }) WITH Genres, m UNWIND
RANGE(0, SIZE(Genres)-1) as i MERGE (g:Genre {name: toUpper(Genres[i])})
CREATE (m)-[r:HAS_GENRE {position:i+1}]->(g);
```

Subsequently, the *ratings.csv* file is processed to create `:User` nodes and `:RATED` relationships that connect users to movies, annotated with the numeric rating values.

```
LOAD CSV WITH HEADERS FROM  file:///ratings.csv  AS line WITH line MATCH
(m:Movie { id: TOINTEGER(line.`movieId`) }) MERGE (u:User { id:
TOINTEGER(line.`userId`) }) CREATE (u)-[r:RATED
{rating:TOFLOAT(line.`rating`)}]->(m);
```

Once the import is complete, the foundational step for the recommendation system involved defining the precise graph schema based on the input CSV files: the model focuses exclusively on user interactions and content categorization. The structure of the graph is formally defined below, outlining the Entities (Nodes) and Key Relationships that form the basis of the system.

Main Entities (Nodes):

- :User — Represents the unique users who provide ratings.
- :Movie — Represents the film content.
- :Genre — Represents thematic categories of the films.

Principal Relationships:

- `(:User)-[:RATED {rating: float}]->(:Movie)`: This relationship captures the core user interaction, with the rating property being the central metric for predictive algorithms.
- `(:Movie)-[:HAS_GENRE {position: int}]->(:Genre)`: This link connects content to its categories, where the optional position property is derived from the genre list order in the source data.

The overall architecture of the graph can be conceptually represented by the following logical path:

```
(:User) - [:RATED \{rating\}] -> (:Movie) - [:HAS\_GENRE \{position\}] -> (:Genre)
```

To visually confirm the implemented schema, the `CALL db.schema.visualization()` procedure was executed in the Neo4j Browser. The resulting visualization, displayed in Figure 1, confirms the successful mapping of the conceptual model into the physical graph database.

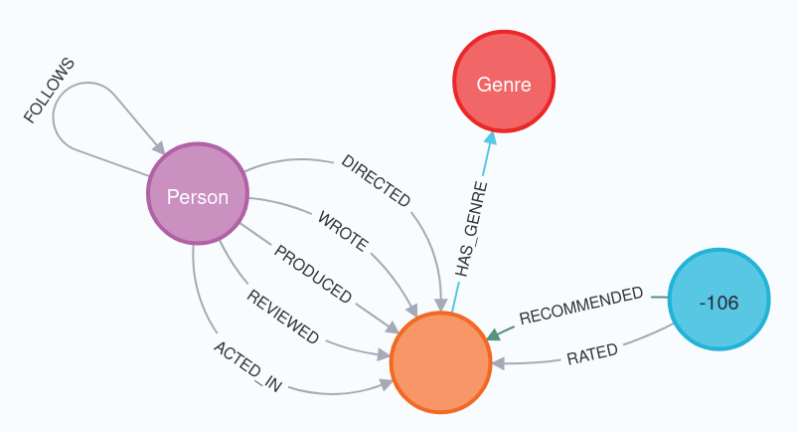


Figure 1: Visualization of the complete database scheme generated by the Neo4j.

The visualization also displays residual components, such as the `:Person` node and related relationships (e.g., DIRECTED, ACTED_IN, FOLLOWS, etc.). However, these entities and relationship types were discarded in this model as the recommendation system's core logic is based exclusively on approaches that do not require actor or director information.

The first step in EDA is to determine the fundamental dimensions of the dataset, in Table 1, providing an essential understanding of the graph's size and complexity. The dataset comprises 671 unique users and 9163 unique movies, interconnected by 100004 rating interactions. These figures establish the scale of the system. Crucially, the overall User-Item matrix presents a high sparsity of 98.37% (computed as $1 - \frac{100004}{671 \times 9763}$). However, the resulting average interaction density (approximately 149 ratings per user and 10.9 ratings per movie) still indicates a manageable and favorable interaction space for training collaborative filtering models.

Table 1: Summary of Initial Structural Metrics Queries

Metric	Query	Result
Total Number of Users	`MATCH (u:User) RETURN count(u)`	671
Total Number of Movies	`MATCH (m:Movie) RETURN count(m)`	9163
Total Number of Ratings	`MATCH ()-[r:RATED]->() RETURN count(r)`	100004
Total Number of Genres	`MATCH (g:Genre) RETURN count(g)`	20

1.2 Most Rated Movies Analysis

The initial exploratory query identifies the most rated movies in the dataset by counting the number of incoming `RATED` relationships for each `Movie` node. This measure reflects the popularity and visibility of a movie within the user–item graph, as movies with higher numbers of ratings are more likely to be well-known and widely viewed. The results indicate that a small group of classic and highly popular titles dominate the ranking.

```
MATCH (m:Movie)<-[r:RATED]-(:User)
RETURN m.title AS Movie, count(r) AS NumRatings
ORDER BY NumRatings DESC
LIMIT 10;
```

Table 2: Top 10 Most Rated Movies

Movie	NumRatings
Forrest Gump (1994)	341
Pulp Fiction (1994)	324
The Shawshank Redemption (1994)	311
The Silence of the Lambs (1991)	304
Star Wars: Episode IV – A New Hope (1977)	291
Jurassic Park (1993)	274
The Matrix (1999)	259
Toy Story (1995)	248
Schindler's List (1993)	244
Terminator 2: Judgment Day (1991)	237

1.3 Most Active Users Analysis

The second query identifies the most active users in the dataset by counting the number of outgoing `RATED` relationships for each `User` node. This measure quantifies user engagement and provides insight into the distribution of participation within the platform.

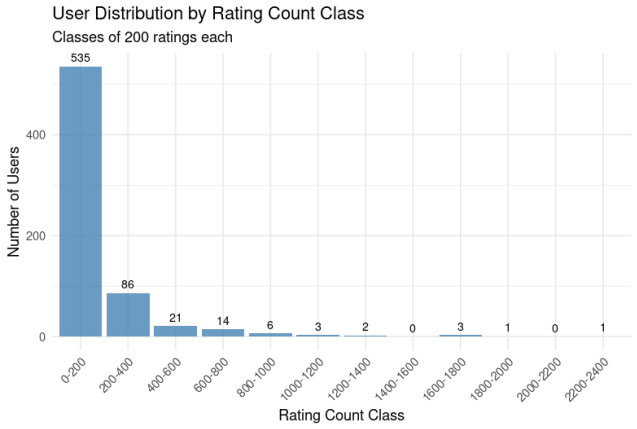
```
MATCH (u:User)-[r:RATED]->(:Movie)
RETURN u.id AS UserID, count(r) AS NumRatings
ORDER BY NumRatings DESC
LIMIT 10;
```

The results in Table 2a reveal that a relatively small subset of users contribute a disproportionately large number of ratings. For instance, the most active user has provided more than 2,300 ratings, whereas the tenth most active user has slightly above 1,000. The plot in Figure 2b illustrates that the vast majority of users fall into the low-activity class, confirming the acute sparsity of the dataset. From a modeling standpoint, these highly active users are valuable, as their extensive rating histories allow for more accurate similarity computations and user-based collaborative filtering.

Overall, the activity distribution suggests a realistic behavioral pattern consistent with real-world rating platforms: most users interact sporadically, while a small group of power users engage extensively, shaping the collective trends of the system.

User ID	Rating Count
47	2391
564	1868
624	1735
15	1700
73	1610
452	1340
468	1291
380	1063
311	1019
30	1011

(a) Top 10 Most Active Users



(b) Distribution of users by number of ratings classes.

1.4 Average Ratings Analysis

The following analysis examines both the individual movie ratings and the aggregated ratings by genre, providing an overview of user appreciation trends in the dataset. The results in Table 3 show that the highest-rated movies all share a perfect average score of 5.0; however, these films have been rated by only a very limited number of users (between 2 and 4). This observation highlights a common issue in recommender datasets: movies with fewer ratings may appear at the top due to small-sample bias.

```
MATCH (u:User)-[r:RATED]->(m:Movie)
RETURN m.title AS Movie,
       round(avg(r.rating), 2) AS AvgRating,
       count(r) AS NumRatings
ORDER BY AvgRating DESC, NumRatings DESC
LIMIT 10;
```

Table 3: Top 10 Movies by Average Rating

Movie	AvgRating	NumRatings
A Face in the Crowd (1957)	5.00	4
Red Firecracker, Green Firecracker (Pao Da Shuai Yan) (1994)	5.00	3
'night Mother (1986)	5.00	3
A Passage to India (1984)	5.00	2
Love Me If You Dare (Jeux d'enfants) (2003)	5.00	2
Maya Lin: A Strong Clear Vision (1994)	5.00	2
It's Such a Beautiful Day (2012)	5.00	2
The Unvanquished (Aparajito) (1957)	5.00	2
Step Into Liquid (2002)	5.00	2
Taste of Cherry (Ta'm e guilass) (1997)	5.00	2

When the analysis is extended to the genre level (Table 4), a more stable trend emerges. Genres such as **Film-Noir**, **War**, and **Documentary** consistently achieve the highest average ratings, suggesting that these categories tend to attract a more discerning and appreciative audience. In contrast, popular genres such as **Drama** and **Crime** have slightly lower average ratings but a significantly higher number of reviews, reflecting their widespread popularity and broader audience base. The overall mean ratings across genres remain relatively close, indicating that user satisfaction is fairly consistent across different types of content, though certain artistic or specialized genres may achieve higher critical acclaim.

```
MATCH (:User)-[r:RATED]->(m:Movie)-[:HAS_GENRE]->(g:Genre)
RETURN g.name AS Genre,
       round(avg(r.rating), 2) AS AvgRating,
       count(r) AS NumRatings
ORDER BY AvgRating DESC;
```

Table 4: Average Rating and Number of Ratings by Genre

Genre	AvgRating	NumRatings
FILM-NOIR	3.96	1140
WAR	3.82	5025
DOCUMENTARY	3.81	1564
(NO GENRES LISTED)	3.78	18
DRAMA	3.68	44752
CRIME	3.68	16266
MYSTERY	3.68	7625
ANIMATION	3.64	6171
MUSICAL	3.60	4722
IMAX	3.57	3156

1.5 Analysis of Highly Rated and Popular Movies

Subsequently, we extended our exploratory analysis to identify the movies that combine both a high level of appreciation and substantial audience engagement. This step focuses on those titles that achieved not only excellent ratings but also a significant number of user evaluations. This dual criterion allows us to highlight films that are both critically acclaimed and statistically reliable.

To ensure the robustness of the results, a minimum threshold of fifty ratings was imposed, thereby reducing small-sample bias and providing a more representative measure of user consensus.

```
MATCH (u:User)-[r:RATED]->(m:Movie)
WITH m, avg(r.rating) AS AvgRating, count(r) AS NumRatings
WHERE NumRatings > 50
RETURN m.title AS Movie, round(AvgRating,2) AS AvgRating, NumRatings
ORDER BY AvgRating DESC, NumRatings DESC
LIMIT 10;
```

The results, presented in Table 5, reveal that several enduring classics dominate the top positions. *The Shawshank Redemption* (1994) and *The Godfather* (1972) both lead the ranking with an average rating of 4.49, followed closely by *The Godfather: Part II* (1974) and *The Maltese Falcon* (1941). These findings underline a clear audience preference for timeless dramas and crime films that have achieved both critical and popular success across decades.

The inclusion of other titles such as *Rear Window* (1954), *12 Angry Men* (1957), and *City of God* (2002) further demonstrates that users tend to value strong storytelling, moral depth, and cinematic craftsmanship. The relatively narrow range of average ratings (4.3–4.5) suggests a high level of agreement among viewers regarding the quality of these films. This consensus indicates that, once the small-sample bias is removed, the graph exhibits strong collective preference signals for highly-rated content.

Table 5: Top 10 Highly Rated and Popular Movies

Movie	AvgRating	NumRatings
The Shawshank Redemption (1994)	4.49	311
The Godfather (1972)	4.49	200
The Godfather: Part II (1974)	4.39	135
The Maltese Falcon (1941)	4.39	62
The Usual Suspects (1995)	4.37	201
Chinatown (1974)	4.34	76
Rear Window (1954)	4.32	92
Schindler's List (1993)	4.30	244
12 Angry Men (1957)	4.30	74
City of God (Cidade de Deus) (2002)	4.30	69

1.6 Analysis of Movie Distribution by Genre

Following the analysis of highly rated and popular titles, the examination of the structural balance of the dataset by counting the number of movies associated with each genre is done. This step provides an overview of the content

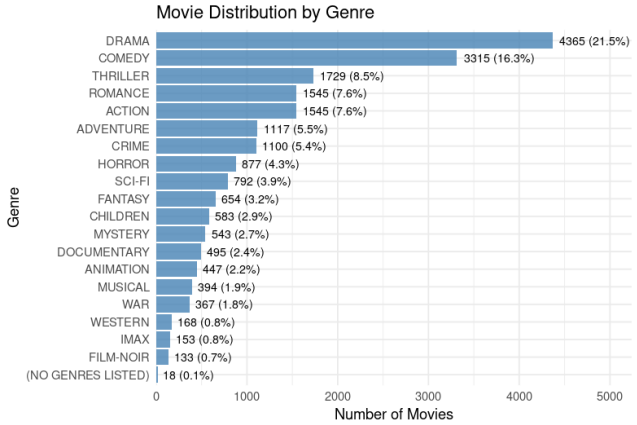
diversity within the collection and helps to identify which genres dominate the dataset, which, in turn, may influence user interaction patterns and recommendation outcomes.

```
MATCH (g:Genre)<-[:HAS_GENRE]-(m:Movie)
RETURN g.name AS Genre, count(m) AS NumMovies
ORDER BY NumMovies DESC;
```

As shown in Table 3a, the distribution of movies is highly uneven across genres. **Drama** and **Comedy** clearly dominate the dataset, accounting for a substantial proportion of all entries (4,365 and 3,315 titles, respectively). These two genres together represent the core of the database, reflecting both the prevalence of such films in general production and their broad appeal among audiences. Mid-tier genres such as **Thriller**, **Romance**, and **Action** also hold a significant presence, while more specialized categories such as **Fantasy**, **Sci-Fi**, and especially **Horror** are less represented numerically but remain relevant for capturing specific user preferences. The visual representation in Figure 3b of this distribution clearly highlights the long-tail characteristic of the dataset, where a few genres account for the majority of the movies, a crucial factor to consider when designing Content-Based strategies.

Genre	NumMovies
DRAMA	4365
COMEDY	3315
THRILLER	1729
ROMANCE	1545
ACTION	1545
ADVENTURE	1117
CRIME	1100
HORROR	877
SCI-FI	792
FANTASY	654

(a) Number of Movies per first 10 Genres



(b) Movie distribution per Genre

The genre distribution highlights a natural bias toward mainstream categories, which may affect the variety of recommendations produced by collaborative or content-based filtering models. Nevertheless, the presence of a wide range of genres, from **Drama** and **Action** to **Documentary** and **Fantasy**, ensures that the dataset maintains an adequate level of thematic diversity necessary for robust and generalizable recommendation performance.

The exploratory data analysis confirmed that the MovieLens dataset was successfully modeled as a graph structure suitable for Conten-Based and Collaborative Filtering Approaches. Key findings include the high, yet manageable, sparsity of the User-Item matrix, the identification of a small cohort of highly engaged users critical for model training, and the strong collective preference signals found within the highly-rated, popular content. The uneven but diverse content distribution by genre gives both a challenge and an opportunity. These structural properties provide a solid foundation for the implementation of the prediction algorithm detailed in the following sections.

2 Content-Based Approach: The Item Profile Implementation

The Content-Based Approach (CB) leverages the intrinsic characteristics of the movies to generate personalized suggestions. The core idea is simple: recommend movies whose genre profiles are highly similar to the user's previously liked content. Using the tripartite graph structure ($Movie \rightarrow Genre$), the similarity between items can be calculated using the Jaccard Coefficient, focusing exclusively on the shared genre attributes.

Step 0: Two parameters are defined to ensure the reproducibility of the queries, specifying a reference movie and a maximum number of results to get in output:

```
:param title => Fargo (1996) ;
:param limit => 10;
```

Step 1: In the first step, the system computes the Jaccard Similarity Coefficient. It is a measure of similarity between two sets, based on the ratio between the intersection and union of the sets themselves. Basically, this metric normalizes the count of shared genres by the total number of unique genres involved in both movies. Given two sets, A and B, the Jaccard similarity is:

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

where $A \cap B$ is the number of shared movies in the two sets and $A \cup B$ is the total number of unique elements of the two sets. This ratio produces a similarity score between 0 and 1, where:

- $J(A, B) = 1$ means that the movies share exactly the same genres, having identical genre profiles.
- $J(A, B) = 0$ means they don't share any genre.

By ranking movies according to this score, the system can identify the movies that are most similar to a given title in a more accurate and balanced way than the simple counting approach.

```
// Find the target movie and collect its genres
MATCH (m1:Movie {title: $title})-[:HAS_GENRE]->(g:Genre)
WITH m1, COLLECT(DISTINCT g.name) AS g1, SIZE(COLLECT(DISTINCT g.name)) AS n1

//Find all other movies and their genres for comparison
MATCH (m2:Movie)-[:HAS_GENRE]->(g2:Genre)
WHERE m2 <> m1
WITH m1, m2, g1, n1, COLLECT(DISTINCT g2.name) AS g2list, SIZE(COLLECT(DISTINCT g2.name)
    ) AS n2

// Calculate Jaccard similarity components
WITH m2,
     SIZE([x IN g1 WHERE x IN g2list]) AS inter,
     (n1 + n2 - SIZE([x IN g1 WHERE x IN g2list])) AS unionSize
// Filter out cases where union size is zero
WHERE unionSize > 0

// Return results with similarity scoring
RETURN m2.id AS movieId, m2.title AS title,
       inter AS commonGenres,
       // jaccard = inter/unionSize
       toFloat(inter)/toFloat(unionSize) AS jaccard
ORDER BY jaccard DESC, commonGenres DESC, title
LIMIT $limit;
```

The above query collects genres of the reference movie and compare them with the genres of all other movies to compute the Jaccard similarity, returning the movie titles sorted by Jaccard coefficient in decreasing order.

Table 6: Recommended movies for "Fargo (1996)" according to Jaccard Similarity

ID	Movie Title	Common Genres	Jaccard Coefficient
6003	Confessions of a Dangerous Mind (2002)	4	1.00
900	Cul-de-sac (1966)	4	1.00
1034	Freeway (1996)	4	1.00
7669	In Bruges (2008)	4	1.00
75813	The Informant! (2009)	4	1.00
73266	Leaves of Grass (2009)	4	1.00
296	Man Bites Dog (C'est arrivé près de chez vous) (1992)	4	1.00
27674	11:14 (2003)	4	0.80
5027	Another 48 Hrs. (1990)	4	0.80
296	Pulp Fiction (1994)	4	0.80

These results in Table 6 reveal a significant content similarity between the reference film, "Fargo (1996)", and the recommended candidates, quantified via the Jaccard Coefficient. The most notable finding is the achievement of a perfect Jaccard score of 1.0 by seven films, such as *"Confessions of a Dangerous Mind (2002)"* and *"Cul-de-sac (1966)"*, which denotes an exact theoretic identity between their respective genre sets and that of the target film. This equivalence of genre set is supported by the consistent value of four common genres observed across all results. The Jaccard index serves as a normalization metric that validates the proportion of similarity: a Jaccard of **1.0**, while maintaining an intersection of four elements, implies the size of the union of the sets is also four. Conversely, films exhibiting a Jaccard score of **0.8**, while preserving the same four genre intersection, prove a lower proportional similarity. This means the combined genre set contains five distinct genres, indicating that their genre profiles contain a single, unshared genre relative to the source film. The Jaccard methodology thus ensures that recommendations are based on the proportional coherence of the genre profile, rather than merely its absolute size.

Step 2: The second step extends the model from a general movie-to-movie similarity framework to a user-centered recommendation approach. Instead of simply ranking similar movies, the system now estimates how a specific user would rate a given movie by analyzing their historical ratings. Each movie previously rated by the user contributes to the prediction according to its similarity with the target movie. This procedure transforms the content-based model from a generic similarity engine into a personalized recommendation system customized to individual user preferences. The implementation is conducted extracting the set of genres associated with the target movie and compares it with the genres of each movie previously rated by the user. As in step 1, the similarity is computed through the Jaccard coefficient, so each past rating contributes to the prediction proportionally to this similarity value, resulting in a weighted average of the user's past rating, i.e.:

$$\hat{r} = \frac{\sum_i J(target, i) \cdot r_i}{\sum_i J(target, i)}$$

where $J(target, i)$ is the Jaccard similarity between the target movie and the movie i while r_i is the corresponding rating. First the parameters for the target user and movie for which the rating is to be predicted are defined. **User 1** was selected to serve as the reference user for generating personalized recommendations, consistent across all Content-Based steps. This choice allows for a controlled analysis of the model's fundamental behavior and ensures the reproducibility of the prediction and recommendation lists, regardless of the underlying data sparsity.

```
:param userId => 1;
:param movieId => 2;
```

The following query computes the predicted rating by first retrieving the genres of the target movie and comparing them with the genres of all movies previously rated by the user to calculate the Jaccard similarity.

```
CALL {
  MATCH (target:Movie {id: $movieId})-[:HAS_GENRE]->(g:Genre)
  WITH target, COLLECT(DISTINCT g.name) AS targetGenres, SIZE(COLLECT(DISTINCT g.name))
      AS targetCount

  MATCH (u:User {id: $userId})-[ur:RATED]->(other:Movie)
  WHERE other.id <> $movieId
  MATCH (other)-[:HAS_GENRE]->(og:Genre)
  WITH target, targetGenres, targetCount, ur.rating AS userRating,
       COLLECT(DISTINCT og.name) AS otherGenres, SIZE(COLLECT(DISTINCT og.name)) AS
           otherCount

  WITH
    // Weighted sum of user ratings with weights equal to Jaccard similarity between
        target movie and other already rated movies genres computation
    SUM( ( toFloat(SIZE([x IN targetGenres WHERE x IN otherGenres]))
         / toFloat(targetCount + otherCount - SIZE([x IN targetGenres WHERE x IN
             otherGenres])) )
        * userRating ) AS weightedSum,

    // Sum of all weights used in weighted average computation
    SUM( toFloat(SIZE([x IN targetGenres WHERE x IN otherGenres]))
         / toFloat(targetCount + otherCount - SIZE([x IN targetGenres WHERE x IN
             otherGenres])) ) AS totalWeight
```

```
  RETURN CASE WHEN totalWeight IS NULL OR totalWeight = 0 THEN NULL ELSE weightedSum /
      totalWeight END AS predicted,
          coalesce(totalWeight, 0) AS total_weight
}

// Fallback mechanism if predicted is NULL
WITH predicted, total_weight
// User average
OPTIONAL MATCH (u:User {id: $userId})-[ur:RATED]->()
WITH predicted, total_weight, AVG(ur.rating) AS user_avg
// Movie average
OPTIONAL MATCH ()-[mr:RATED]->(m:Movie {id: $movieId})
WITH predicted, total_weight, user_avg, AVG(mr.rating) AS movie_avg
// Global average
OPTIONAL MATCH ()-[gr:RATED]->()
WITH predicted, total_weight, user_avg, movie_avg, AVG(gr.rating) AS global_avg

RETURN COALESCE(predicted, user_avg, movie_avg, global_avg, 3.0) AS predicted_rating,
       total_weight;
```

Here `total_weight` measures the sum of similarities used in the rating computation and provides information about the estimate robustness: the higher the values, the more reliable the prediction. To ensure that the system always produces a rating prediction, a fallback mechanism is applied when the content-based computation is not possible or not reliable (e.g., when `total_weight` is zero, meaning no overlapping genres with the user's past rated movies). In this case, the system sequentially resorts to: the average rating of the user (`user_avg`), the average rating of the movie (`movie_avg`), the global average across all ratings (`global_avg`), and finally a default value (e.g., 3.0) if no other data is available. This guarantees that a predicted rating is always returned, while `total_weight` continues to provide an indication of how much the content-based component contributes to the final prediction.

Using the content-based model, the predicted rating for user 1 and *"Jumanji (1995)"* movie (id = 2) is **2.41** with a `total_weight` of **1.24**. This result indicates that the system predicts the user would likely rate the movie slightly below the midpoint of the rating scale. The `total_weight` value reflects the confidence of the content-based prediction: since it is relatively low (1.24), it suggests that the overlap between the target movie's genres and the genres of movies previously rated by the user is limited. Therefore, the prediction relies on a small number of contributing ratings, making it less robust.

Step 3: In the third step the evaluation of the obtained predictions is conducted selecting a fixed-size sample set of real user ratings to compute a predicted rating for each pair (user, movie) using the content-based procedure described before. To ensure a manageable computational load and maintain consistent evaluation times, the sample size was limited to 1 000 (user–movie) rating pairs. This subset provides a representative sample of the dataset while allowing for efficient computation of prediction accuracy metrics. The two metrics are RMSE (Root Mean Squared Error) and MAE (Mean Absolute Error). Both measure the difference between the predicted ratings $\hat{r}_i$ and the actual rating r_i over a set of N, here 1 000 predictions, but they differ in how they treat large errors:

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(r_i - \hat{r}_i)^2} \qquad MAE = \frac{1}{N}\sum_{i=1}^{N}|r_i - \hat{r}_i|$$

The first gives higher weights to large errors, i.e. the outliers, while the second provides a straightforward interpretation since it is measured on the same scale of the ratings.

```
// Select a sample set with 1 000 real ratings
MATCH (u:User)-[r:RATED]->(m:Movie)
WITH u, m, r.rating AS real_rating
LIMIT 1000

// Compute predicted ratings
CALL {
    WITH u, m
```

```
    // Retrieve the genres and count for the target movie (m)
    MATCH (target:Movie {id: m.id})-[:HAS_GENRE]->(g:Genre)
    WITH target, COLLECT(DISTINCT g.name) AS targetGenres, SIZE(COLLECT(DISTINCT g.name)
        ) AS targetCount

    // Retrieve all other movies rated by the user (u)
    MATCH (u)-[ur:RATED]->(other:Movie)
    WHERE other.id <> target.id
    MATCH (other)-[:HAS_GENRE]->(og:Genre)
    WITH target, targetGenres, targetCount, ur.rating AS userRating, COLLECT(DISTINCT og
        .name) AS otherGenres, SIZE(COLLECT(DISTINCT og.name)) AS otherCount

    // Compute Weighted Sum and Total Weight (Jaccard * Rating)
    WITH SUM( (toFloat(SIZE([x IN targetGenres WHERE x IN otherGenres])) / toFloat(
        targetCount + otherCount - SIZE([x IN targetGenres WHERE x IN otherGenres]))) *
        userRating ) AS weightedSum,
          SUM( toFloat(SIZE([x IN targetGenres WHERE x IN otherGenres])) / toFloat(
              targetCount + otherCount - SIZE([x IN targetGenres WHERE x IN otherGenres])
              ) ) AS totalWeight

    // Return predicted rating (Weighted Sum / Total Weight)
    RETURN CASE WHEN totalWeight = 0 THEN NULL ELSE weightedSum / totalWeight END AS
        predicted_rating
}

// Fallback if predicted_rating is null
WITH u, m, real_rating, predicted_rating
OPTIONAL MATCH (u)-[ur:RATED]->()
WITH u, m, real_rating, predicted_rating, AVG(ur.rating) AS user_avg
OPTIONAL MATCH ()-[mr:RATED]->(m)
WITH u, m, real_rating, predicted_rating, user_avg, AVG(mr.rating) AS movie_avg
OPTIONAL MATCH ()-[gr:RATED]->()
WITH u, m, real_rating, predicted_rating, user_avg, movie_avg, AVG(gr.rating) AS
    global_avg

WITH u, m, real_rating,
     COALESCE(predicted_rating, user_avg, movie_avg, global_avg, 3.0) AS final_pred

// RMSE and MAE computation on sample set
WITH collect({real: real_rating, pred: final_pred}) AS results
UNWIND results AS r
WITH r.real AS real, r.pred AS pred
RETURN sqrt(AVG((real - pred)*(real - pred))) AS RMSE,
       AVG(ABS(real - pred)) AS MAE;
```

The query computes the two metrics over a sample of size 1 000 and the computed values summarize the overall predictive performance of the content-based model combination with the fallback logic. The RMSE is **1.41** and the MAE is **1.21**: they indicate that, on average, the model's prediction deviate from the actual ratings by about one point on the rating scale (1-5). For a simple content-based model that relies solely on genre those values are large but expected. The system captures general user preferences but could be improved by integrating additional features such as directors or actors.

Step 4: Finally, the top 10 movie recommendations are generated for the reference user (id = 1) using the content-based approach. Instead of starting from a target movie, the system now analyzes all movies previously rated by the user and estimates which unseen movies the user is likely to enjoy. The query takes the user id and the number of desired recommended movies as input parameters.

```
:param userId => 1;
:param limit => 10;
```

For each candidate movie, the predicted rating is computed as a weighted average of the user's past ratings,

with weights corresponding to the squared Jaccard similarity between the genres of previously rated movies and the candidate movie. The query also returns total_weight as an indicator of prediction reliability.

```
// Retrieve movies already rated by the user and their associated genres
MATCH (u:User {id: $userId})-[r:RATED]->(m1:Movie)-[:HAS_GENRE]->(g:Genre)
WITH u, r.rating AS userRating, m1,
     COLLECT(DISTINCT g.name) AS g1,
     SIZE(COLLECT(DISTINCT g.name)) AS n1

// Select all movies that have not yet been rated by the user
MATCH (m2:Movie)-[:HAS_GENRE]->(g2:Genre)
WHERE NOT EXISTS((u)-[:RATED]->(m2)) AND m2 <> m1
WITH m2, g1, n1, COLLECT(DISTINCT g2.name) AS g2list,
     SIZE(COLLECT(DISTINCT g2.name)) AS n2, userRating

// Compute the Jaccard similarity between genres of rated movies and candidate movies
WITH m2, userRating,
     SIZE([x IN g1 WHERE x IN g2list]) AS inter,
     (n1 + n2 - SIZE([x IN g1 WHERE x IN g2list])) AS unionSize
WHERE inter > 0
WITH m2,
     // Weighted sum using squared similarity as weight
     SUM((toFloat(inter) / toFloat(unionSize))^2 * userRating) AS weightedSum,
     // Total weight (sum of squared similarities)
     SUM((toFloat(inter) / toFloat(unionSize))^2) AS weights

// Compute the predicted rating as the weighted average of similar movies' ratings
WITH m2,
     CASE
         WHEN weights = 0 THEN NULL
         ELSE weightedSum / weights
     END AS predictedRating,
     weights AS total_weight

// Normalize the predicted rating between 1 and 5 and return top recommendations
RETURN
     m2.id AS movieId,
     m2.title AS RecommendedMovie,
     CASE
         WHEN predictedRating > 5 THEN 5
         WHEN predictedRating < 1 THEN 1
         ELSE ROUND(predictedRating, 2)
     END AS predictedRating,
     total_weight
ORDER BY predictedRating DESC, RecommendedMovie
LIMIT $limit;
```

The output provides a ranked list of the top recommended movies for the user, including the predicted rating and the total weight for each movie. As shown in Table 7, all movies exhibit a uniform predicted rating of **3.5** and an identical total weight of **0.0625**. This result is a direct result of User 1's sparse profile, having only **20** total ratings. Given the limited historical data, the average rating of the user is likely 3.5, and all candidate movies share an identical low squared Jaccard similarity (0.0625) with the user's rated items. This scenario highlights a key limitation of the Content-Based approach: while it successfully generates recommendations even from minimal input (the cold start problem), the lack of diverse input can lead to homogenized predictions suggestions.

Table 7: Top 10 Recommended movie for User 1

MovieID	Recommended Movie	Predicted Rating	Total Weight
3206	Against All Odds (1984)	3.5	0.0625
2708	Autumn Tale, An (Conte d'automne) (1998)	3.5	0.0625
7320	Dirty Dancing: Havana Nights (2004)	3.5	0.0625
4503	Everybody's All-American (1988)	3.5	0.0625
4024	House of Mirth, The (2000)	3.5	0.0625
1137	Hustler White (1996)	3.5	0.0625
638	Jack and Sarah (1995)	3.5	0.0625
1475	Kama Sutra: A Tale of Love (1996)	3.5	0.0625
1493	Love and Other Catastrophes (1996)	3.5	0.0625
2340	Meet Joe Black (1998)	3.5	0.0625

Despite the uniform predicted ratings, this list provides a personalized set of suggestions for the user, demonstrating how the system can generate customized recommendations even when the similarity weights are low or evenly distributed.

While the content-based approach effectively recommends movies with similar genres to those previously liked by the user, it is limited to the available content features and cannot capture patterns across users with similar tastes. To overcome this limitation and improve prediction accuracy, the system can be extended to collaborative filtering approach, which leverages the preferences of similar users to generate recommendations beyond what content alone can suggest.

3 Collaborative Filtering: Leveraging User Similarities

In this section, we introduce and implement a user–user Collaborative Filtering (CF) model on the user–movie graph. Collaborative Filtering is one of the most established and intuitive paradigms for recommendation systems, based on the assumption that users who exhibited similar preferences in the past will also share similar tastes in the future. Unlike content-based approaches, which rely on item features or metadata, collaborative filtering operates exclusively on patterns of user behavior — in this case, the explicit movie ratings stored as relationships in the graph.

Each user is represented as a vector of ratings across the movies they have evaluated. The similarity between users is determined by comparing their rating vectors through the cosine similarity measure, which quantifies the extent to which their preferences are correlated. Once the most similar users (called *neighbors*) are identified, the system predicts the target user's potential rating for an unseen movie by aggregating the ratings provided by those neighbors, weighted by their degree of similarity to the target. The final prediction therefore represents a weighted consensus among the most relevant peers in the rating space.

Formally, for a target user u and a candidate movie m, the predicted rating $\hat{r}_{u,m}$ is calculated as a deviation from the user's mean rating $\bar{r}_u$, adjusted by the average tendency of similar users to rate that movie above or below their own means:

$$\hat{r}_{u,m} = \bar{r}_u + \frac{\sum_{v \in N_k(u)} s(u,v)\,[r_{v,m} - \bar{r}_v]}{\sum_{v \in N_k(u)} |s(u,v)|}.$$

Here, $s(u,v)$ represents the similarity between users u and v, and $N_k(u)$ denotes the set of the top-k most similar users to u. The model therefore balances personalization (through each user's rating habits) and collective agreement (through aggregated neighbor opinions).

In practical terms, the implementation reproduces this reasoning directly in the graph: edges of type `RATED` store numerical ratings and connect users to movies, while the model traverses the graph to identify overlapping rating histories, compute similarities, and predict unseen ratings in a fully graph-native manner.

Step 0: Before executing the recommendation workflow, a set of parameters is defined to control the model's behavior and computational scope.

```
:param userId => 1;
```

```
:param sampleSize => 1000;
:param topN => 30;
```

Step 1: In this step, a random subset of user–movie rating pairs is extracted from the dataset to serve as the working sample for subsequent computations. The query selects relationships of type `RATED` connecting users (`u`) and movies (`m`), while storing each rating value as `actual_rating`. A random number is then generated for each pair using the `rand()` function, which ensures that the sampling process is unbiased.

```
MATCH (u:User)-[r:RATED]->(m:Movie)
WITH u, m, r.rating AS actual_rating, rand() AS random
ORDER BY random
LIMIT $sampleSize
```

Step 2: Similarity Computation and Rating Prediction. The goal is to identify users with overlapping movie preferences, quantify their similarity, and use their ratings to predict how the target user would rate unseen movies.

The first subquery identifies pairs of users (`u`, `u2`) who have rated at least one movie in common. For each user pair, two aligned vectors of ratings (`v1` and `v2`) are constructed, representing their evaluations of the same set of movies. Restricting the selection to pairs with more than one shared item (`size(v1) > 1`) ensures that the similarity calculation is statistically meaningful.

```
MATCH (u)-[r1:RATED]->(common:Movie)<-[r2:RATED]-(u2:User)
WHERE u <> u2
WITH u, m, u2, actual_rating,
     collect(toFloat(r1.rating)) AS v1,
     collect(toFloat(r2.rating)) AS v2
WHERE size(v1) > 1
```

The cosine similarity between each pair of users is then computed by measuring the angle between their rating vectors. This metric captures how closely aligned two users' preferences are, independent of rating scale differences. A similarity score close to 1 indicates highly correlated tastes, while values near 0 imply little or no agreement.

```
WITH u, m, u2, actual_rating,
     reduce(dot = 0.0, i IN range(0, size(v1)-1) | dot + (v1[i]*v2[i])) AS dotProduct,
     sqrt(reduce(s = 0.0, x IN v1 | s + (x*x))) AS norm1,
     sqrt(reduce(s = 0.0, x IN v2 | s + (x*x))) AS norm2
WITH u, m, u2, actual_rating,
     CASE WHEN norm1=0 OR norm2=0 THEN 0 ELSE dotProduct/(norm1*norm2) END AS similarity
```

After computing all pairwise similarities, only the top 30 most similar users are retained as the target user's neighbors. This limitation (`topN = 30`) controls computational complexity.

```
ORDER BY similarity DESC
WITH u, m, collect({u2: u2, sim: similarity})[..30] AS neighbors, actual_rating
```

For each neighbor, their mean rating across all movies is computed. This normalization accounts for differences in individual rating scales (e.g., users who tend to rate systematically higher or lower). It allows subsequent calculations to focus on deviations from each user's typical rating behavior.

```
UNWIND neighbors AS n
WITH u, m, n.u2 AS neighbor, n.sim AS sim, actual_rating
MATCH (neighbor)-[r:RATED]->()
WITH u, m, neighbor, sim, actual_rating, avg(toFloat(r.rating)) AS meanNeighbor
```

Similarly, the target user's mean rating is calculated to center the prediction formula around the user's individual baseline.

```
MATCH (u)-[rU:RATED]->()
WITH u, m, neighbor, sim, meanNeighbor, avg(toFloat(rU.rating)) AS meanUser,
    actual_rating
```

The model then aggregates all ratings from the neighbors for each candidate movie. Each neighbor's deviation from their own mean rating is weighted by the similarity to the target user, producing a weighted sum that estimates how much the target user is likely to deviate from their own mean rating for that movie. The denominator (`simSum`) normalizes the contribution of all similarities.

```
MATCH (neighbor)-[r3:RATED]->(m)
WITH u, m, actual_rating, meanUser,
     sum(sim * (toFloat(r3.rating) - meanNeighbor)) AS weightedSum,
     sum(abs(sim)) AS simSum
```

The raw predicted rating is thus computed as the target user's mean rating plus the weighted deviation inferred from similar users. This corresponds directly to the standard user-based collaborative filtering formula.

```
WITH u, m, actual_rating,
     CASE
         WHEN simSum = 0 THEN null
         ELSE meanUser + (weightedSum / simSum)
     END AS raw_predicted
```

Finally, all predicted ratings are clamped to the valid rating interval between 1 and 5. This ensures numerical consistency with the rating scale used in the dataset and avoids unrealistic predictions that fall outside the permissible range.

```
WITH u, m, actual_rating,
     CASE
         WHEN raw_predicted IS NULL THEN NULL
         WHEN raw_predicted < 1 THEN 1
         WHEN raw_predicted > 5 THEN 5
         ELSE raw_predicted
     END AS predicted_rating
```

Step 3: This step implements a fallback strategy to handle cases where a predicted rating could not be computed reliably due to the absence of sufficient neighbor information or shared ratings. In such cases, average values at different aggregation levels — user, movie, and global — are used as proxy estimates to ensure that every user–movie pair receives a meaningful prediction.

```
// Compute user averages
MATCH (u)-[rU:RATED]->()
WITH u, avg(toFloat(rU.rating)) AS user_avg, m, predicted_rating, actual_rating

// Compute movie averages
MATCH (m)<-[rM:RATED]-()
WITH u, m, user_avg, avg(toFloat(rM.rating)) AS movie_avg, predicted_rating,
    actual_rating

// Compute global average
MATCH (:User)-[rG:RATED]->()
WITH u, m, user_avg, movie_avg, avg(toFloat(rG.rating)) AS global_avg, predicted_rating,
     actual_rating

// Combine all (predicted, user, movie, global)
WITH u, m, actual_rating,
     coalesce(predicted_rating, user_avg, movie_avg, global_avg, 3.0) AS final_pred
```

Step 4: In this step we quantitatively assess the predictive accuracy of the collaborative filtering model by computing two widely adopted error metrics: the Root Mean Squared Error (RMSE) and the Mean Absolute Error (MAE). These metrics provide complementary insights into the performance of the rating-prediction subsystem.

```
WITH collect({real: actual_rating, pred: final_pred}) AS results
UNWIND results AS r
WITH sqrt(avg((r.pred - r.real)^2)) AS RMSE,
     avg(abs(r.pred - r.real)) AS MAE
RETURN
    round(RMSE,3) AS RMSE,
    round(MAE,3)  AS MAE;
```

In this approach a RMSE of **0.747** and a MAE of **0.582** were obtained. These results indicate a relatively low average error in predicted ratings: on average the model's predicted rating differs from the actual rating by

approximately 0.58 units on the original 1–5 scale (MAE), and the RMSE suggests that large deviations remain infrequent, since RMSE < 1, the error magnitude is well under one full rating point.

The evaluation results demonstrate that the Collaborative Filtering (CF) model achieves substantially higher predictive accuracy than the Content-Based approach. With a RMSE of **0.747** and a MAE of **0.582**, the CF model produces ratings that are, on average, closer to the actual user evaluations than those predicted by the Content-Based model (**RMSE = 1.41**, **MAE = 1.21**).

This improvement confirms that leveraging user–user similarity relationships allows the CF system to better capture latent preference patterns, resulting in more reliable and personalized predictions.

Step 5: In this step, the model applies the similarity-based mechanisms developed and validated in the previous stages to infer which movies the user is most likely to appreciate. While earlier steps primarily aimed to evaluate predictive accuracy (e.g., through RMSE and MAE computation), this phase operationalizes the recommendation process by ranking unseen movies according to predicted preference strength and confidence.

```
//  Find similar users (neighbors)
MATCH (u:User {id: $userId})-[r1:RATED]->(common:Movie)<-[r2:RATED]-(u2:User)
WHERE u <> u2
WITH u, u2,
     collect(toFloat(r1.rating)) AS v1,
     collect(toFloat(r2.rating)) AS v2
WHERE size(v1) > 1

//  Compute cosine similarity
WITH u, u2,
     reduce(dot = 0.0, i IN range(0, size(v1)-1) | dot + (v1[i]*v2[i])) AS dotProduct,
     sqrt(reduce(s = 0.0, x IN v1 | s + (x*x))) AS norm1,
     sqrt(reduce(s = 0.0, x IN v2 | s + (x*x))) AS norm2
WITH u, u2,
     CASE WHEN norm1=0 OR norm2=0 THEN 0 ELSE dotProduct/(norm1*norm2) END AS similarity
ORDER BY similarity DESC
LIMIT 30

//  Compute mean ratings for target user and neighbors
MATCH (u)-[rU:RATED]->()
WITH u, avg(toFloat(rU.rating)) AS meanUser, collect({u2:u2, sim:similarity}) AS sims

// Predict ratings for movies not yet rated by the user
UNWIND sims AS s
WITH u, meanUser, s.u2 AS neighbor, s.sim AS sim
MATCH (neighbor)-[r3:RATED]->(rec:Movie)
WHERE NOT (u)-[:RATED]->(rec)
WITH rec, meanUser,
     sum(sim) AS sumSim,
     sum(sim * (toFloat(r3.rating) - meanUser)) AS weightedSum

//  Normalize and attenuate weights
WITH rec, meanUser, weightedSum, sumSim,
     (1 - exp(-sumSim / 20.0)) AS attenuation,
     sumSim AS confidence

//  Compute raw predicted rating and blend with global average
MATCH (:User)-[rAll:RATED]->()
WITH rec, meanUser, weightedSum, sumSim, attenuation, confidence,
     avg(toFloat(rAll.rating)) AS global_avg
WITH rec, confidence,
     CASE
         WHEN sumSim = 0 THEN NULL
         ELSE (
             // 70 % user-based prediction + 30 % global average
             0.7 * (meanUser + ((weightedSum / sumSim) * attenuation)) + 0.3 *
                 global_avg
```

```
        )
    END AS raw_predicted

// Clamp final predicted rating to [1, 5]
WITH rec, confidence,
    CASE
        WHEN raw_predicted IS NULL THEN NULL
        WHEN raw_predicted < 1 THEN 1
        WHEN raw_predicted > 5 THEN 5
        ELSE raw_predicted
    END AS PredictedRating

RETURN
    rec.title AS RecommendedMovie,
    round(PredictedRating, 2) AS PredictedRating,
    round(confidence, 3) AS ConfidenceScore
ORDER BY PredictedRating DESC, ConfidenceScore DESC
LIMIT $topN;
```

The resulting movies are sorted in descending order of their predicted rating and associated confidence score, which represents the degree of consensus among similar users. Higher confidence scores indicate stronger agreement within the neighborhood, and therefore more reliable recommendations.

Table 8: Top-10 Movie Recommendations using Collaborative Filtering

Rank	Recommended Movie	Predicted Rating	Confidence Score
1	Sling Blade (1996)	4.53	79.655
2	Wallace & Gromit: The Wrong Trousers (1993)	4.51	139.356
3	The Name of the Rose (Der Name der Rose) (1986)	4.48	59.724
4	Yhe Usual Suspects (1995)	4.41	139.231
5	Rear Window (1954)	4.36	139.454
6	Harold and Maude (1971)	4.36	79.610
7	Moonstruck (1987)	4.33	39.941
8	Modern Times (1936)	4.33	39.931
9	Diva (1981)	4.33	39.925
10	American Graffiti (1973)	4.33	39.868

4 System Validation: Experimenting with a New User

To evaluate the effectiveness of the developed recommendation models, both the collaborative and content-based filtering approaches were applied to a newly created user, identified as **User 672**. This user was introduced into the system with a predefined set of ratings across fifteen movies, selected to represent a mix of genres and quality levels (ranging from highly rated classics such as *Pulp Fiction (1994)* and *Jurassic Park (1993)* to poorly received titles like *The Room (2003)* and *Catwoman (2004)*). By doing so, the system could infer the user's genre and preference profile, enabling both models to generate recommendations based on their respective methodologies.

The evaluation was conducted by comparing the *predicted* and *actual* ratings for a sample of user–movie pairs, assessing the consistency of the predictions.

Step 1: First the new user is added creating a new node as user 672.

```
CREATE(u:User {id:672})
RETURN u;
```

Step 2: Then, the new user provides initial ratings for a selected set of movies.

```
MATCH (u:User {id:672})
MATCH (m:Movie)
WHERE m.title IN [
    Forrest Gump (1994) ,
```

```
      Toy Story (1995) ,
      Life in a Day (2011) ,
      Pulp Fiction (1994) ,
      Jurassic Park (1993) ,
      The Godfather (1972) ,
      Groundhog Day (1993) ,
      Saw (2004) ,
      Mean Girls (2004) ,
      Catwoman (2004) ,
      The Room (2003) ,
      Battlefield Earth (2000) ,
      Gigli (2003) ,
      Jack and Jill (2011)
]
WITH u, m,
      CASE m.title
          WHEN  Forrest Gump (1994)  THEN 4
          WHEN  Toy Story (1995)  THEN 4
          WHEN  Life in a Day (2011)  THEN 3
          WHEN  Pulp Fiction (1994)  THEN 5
          WHEN  Jurassic Park (1993)  THEN 5
          WHEN  The Godfather (1972)  THEN 4
          WHEN  Groundhog Day (1993)  THEN 2
          WHEN  Saw (2004)  THEN 1
          WHEN  Mean Girls (2004)  THEN 2
          WHEN  Catwoman (2004)  THEN 1
          WHEN  The Room (2003)  THEN 1
          WHEN  Battlefield Earth (2000)  THEN 2
          WHEN  Gigli (2003)  THEN 1
          WHEN  Jack and Jill (2011)  THEN 2
      END AS rating
MERGE (u)-[:RATED {rating: rating}]->(m);
```

Figure 4 shows the visualization of User 672 and its rated movies. The central node represents the new user, while the surrounding nodes correspond to the movies rated by this user. The edges indicate the ratings given, allowing a clear overview of the user's preferences across different genres.

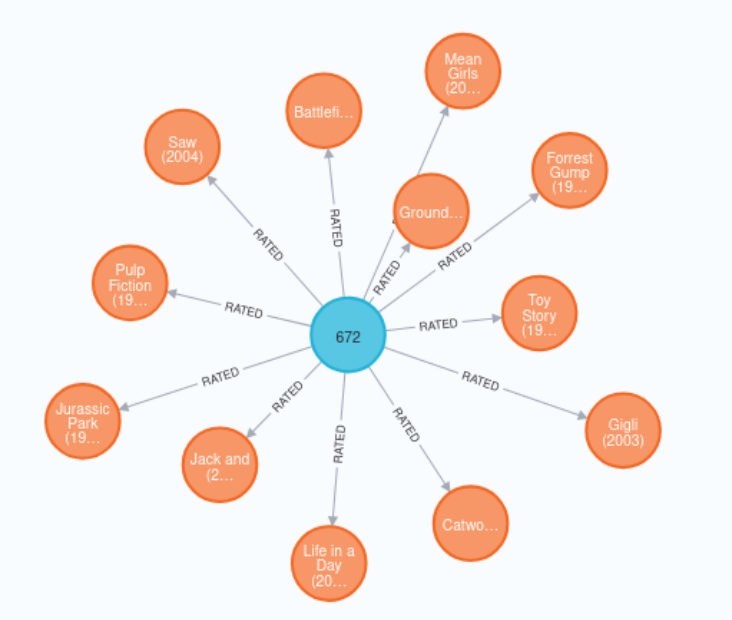


Figure 4: Visualization of the new User 672.

Step 3: First, we were interested in content-based recommendations for user 672.

```
// Retrieve movies already rated by the user and their associated genres
MATCH (u:User {id: 672})-[r:RATED]->(m1:Movie)
MATCH (m1)-[:HAS_GENRE]->(g:Genre)
WITH u, r.rating AS userRating, m1,
     COLLECT(DISTINCT g.name) AS g1,
     SIZE(COLLECT(DISTINCT g.name)) AS n1

// Select all movies that have not yet been rated by the user
MATCH (m2:Movie)-[:HAS_GENRE]->(g2:Genre)
WHERE NOT EXISTS((u)-[:RATED]->(m2)) AND m2 <> m1
WITH m2, g1, n1, COLLECT(DISTINCT g2.name) AS g2list,
     SIZE(COLLECT(DISTINCT g2.name)) AS n2, userRating

// Compute the Jaccard similarity between genres of rated movies and candidate movies
WITH m2, userRating,
     SIZE([x IN g1 WHERE x IN g2list]) AS inter,
     (n1 + n2 - SIZE([x IN g1 WHERE x IN g2list])) AS unionSize
WHERE inter > 0
WITH m2,
     // Weighted sum using squared similarity as weight
     SUM((toFloat(inter) / toFloat(unionSize))^2 * userRating) AS weightedSum,
     // Total weight (sum of squared similarities)
     SUM((toFloat(inter) / toFloat(unionSize))^2) AS weights

// Compute the predicted rating as the weighted average of similar movies' ratings
WITH m2,
     CASE
         WHEN weights = 0 THEN NULL
         ELSE weightedSum / weights
     END AS predictedRating

// Save as property in Movie
SET m2.predictedCB = predictedRating

// Normalize the predicted rating between 1 and 5 and return top recommendations
RETURN
     m2.id AS movieId,
     m2.title AS RecommendedMovie,
     CASE
         WHEN predictedRating > 5 THEN 5
         WHEN predictedRating < 1 THEN 1
         ELSE ROUND(predictedRating, 2)
     END AS predictedRating
ORDER BY predictedRating DESC, RecommendedMovie
LIMIT 10;
```

The prediction results for User 672 mirror the behavior observed with User 1: all candidate movies received an identical predicted rating of 4.61. This uniformity stems from the user's sparse rating history, which limits the diversity of the genre profile used for comparison in the Content-Based approach, thus preventing the model from generating nuanced, differentiated suggestions.

Table 9: Recommended Movies for user 672 using Content-Based Approach

movieId	Recommended Movie	Predicted Rating
2537	Beyond the Poseidon Adventure (1979)	4.61
3166	Brenda Starr (1989)	4.61
5357	Iron Will (1994)	4.61
941	The Mark of Zorro (1940)	4.61
8654	Prince Valiant (1954)	4.61
26013	Rodan (Sora no daikaijû Radon) (1956)	4.61
3207	The Snows of Kilimanjaro (1952)	4.61
5864	Tarzan, the Ape Man (1981)	4.61
60343	Wee Willie Winkie (1937)	4.61
5361	White Fang (1991)	4.61

Step 4: Then, we were interested in Collaborative filtering recommendations for user 672. The first critical step in the Collaborative Filtering process is identifying the user's neighborhood by calculating similarity. To validate this foundation, the Cosine Similarity metric was computed and materialized into a `:SIMILAR_TO` relationship for high-correlation users ($Similarity > 0.8$).

```
// Find user 672 and all the other users who rated the same movie as 672
MATCH (u1:User {id: 672})-[r1:RATED]->(m:Movie)<-[r2:RATED]-(u2:User)
WHERE u1 <> u2
// For each pair (u1, u2) collect their ratings on the same movies
WITH u1, u2,
     collect(toFloat(r1.rating)) AS v1,
     collect(toFloat(r2.rating)) AS v2
// Consider only pairs with at least 3 common movies
WHERE size(v1) > 2
// Compute cosine similarity between the two rating vectors
WITH u1, u2,
     reduce(dot = 0.0, i IN range(0, size(v1)-1) | dot + (v1[i]*v2[i])) AS dotProduct,
     sqrt(reduce(s = 0.0, x IN v1 | s + x^2)) AS norm1,
     sqrt(reduce(s = 0.0, x IN v2 | s + x^2)) AS norm2
// Cosine similarity: dotProduct / (norm1 * norm2)
WITH u1, u2,
     CASE WHEN norm1 = 0 OR norm2 = 0 THEN 0 ELSE dotProduct/(norm1*norm2) END AS
         similarity
// Filter by significant similarities (similarity > 0.8)
WHERE similarity > 0.8
// Create a similarity relationship between users
MERGE (u1)-[:SIMILAR_TO {similarity: similarity}]->(u2)
RETURN u1, u2, similarity
ORDER BY similarity DESC
LIMIT 20;
```

Figure 5 serves as the visual proof of concept that the Cosine Similarity metric successfully identified and persisted the strongest neighbor relationships, establishing the weighted network used by the subsequent prediction query.

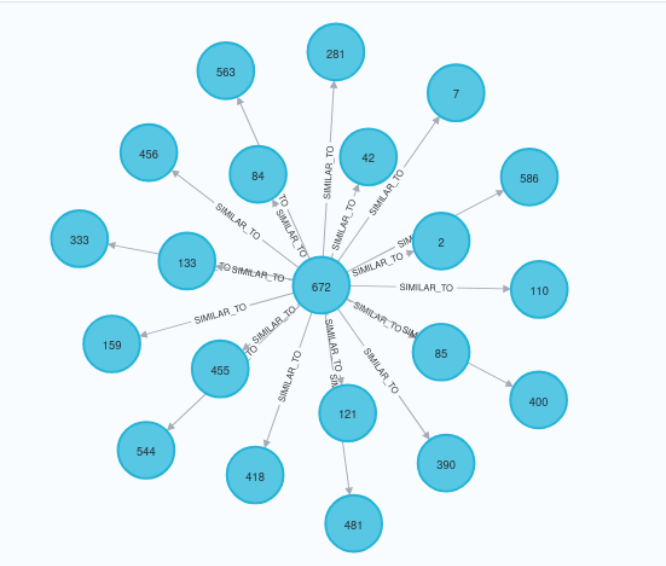


Figure 5: Similarity Graph for High-Correlation Users

The full prediction process then leverages the computed similarity and neighborhood structure to calculate the final predicted ratings.

```
// Find similar users (neighbors)
MATCH (u1:User {id: $userId})-[r1:RATED]->(common:Movie)<-[r2:RATED]-(u2:User)
WHERE u1 <> u2
WITH u1, u2,
     collect(toFloat(r1.rating)) AS v1,
     collect(toFloat(r2.rating)) AS v2
WHERE size(v1) > 1

// Compute cosine similarity
WITH u1, u2,
     reduce(dot = 0.0, i IN range(0, size(v1)-1) | dot + (v1[i] * v2[i])) AS dotProduct,
     sqrt(reduce(s1 = 0.0, i IN range(0, size(v1)-1) | s1 + (v1[i]^2))) AS norm1,
     sqrt(reduce(s2 = 0.0, i IN range(0, size(v2)-1) | s2 + (v2[i]^2))) AS norm2
WITH u1, u2,
     CASE WHEN norm1 = 0 OR norm2 = 0 THEN 0 ELSE dotProduct / (norm1 * norm2) END AS
         sim
ORDER BY sim DESC
LIMIT 30

// Compute mean ratings for target user and neighbors
MATCH (u1)-[rU:RATED]->()
WITH u1, avg(toFloat(rU.rating)) AS meanUser, collect({u2: u2, sim: sim}) AS sims

// Predict ratings for movies not yet rated by the user
UNWIND sims AS s
MATCH (neighbor)-[r3:RATED]->(rec:Movie)
WHERE id(neighbor) = id(s.u2) AND NOT (u1)-[:RATED]->(rec)
WITH u1, meanUser, rec, s.sim AS sim, r3.rating AS neighborRating
WITH u1, meanUser, rec,
     sum(sim) AS sumSim,
     sum(sim * (toFloat(neighborRating) - meanUser)) AS weightedSum

// Normalize and attenuate weights
WITH u1, rec, meanUser, weightedSum, sumSim,
     (1 - exp(-sumSim / 20.0)) AS attenuation,
     sumSim AS confidence
```

```
// Compute predicted rating blending with global average
MATCH (:User)-[rAll:RATED]->()
WITH u1, rec, meanUser, weightedSum, sumSim, attenuation, confidence,
     avg(toFloat(rAll.rating)) AS global_avg
WITH rec, confidence,
     CASE
         WHEN sumSim = 0 THEN NULL
         ELSE (
             0.7 * (meanUser + ((weightedSum / sumSim) * attenuation))
             + 0.3 * global_avg
         )
     END AS raw_predicted

// Clamp predicted rating to [1,5]
WITH rec, confidence,
     CASE
         WHEN raw_predicted IS NULL THEN NULL
         WHEN raw_predicted < 1 THEN 1
         WHEN raw_predicted > 5 THEN 5
         ELSE raw_predicted
     END AS PredictedRating

// Save predicted rating for collaborative filtering
SET rec.predictedCF = PredictedRating

// Return top recommendations
RETURN
     rec.title AS RecommendedMovie,
     round(PredictedRating, 2) AS PredictedRating,
     round(confidence, 3) AS ConfidenceScore
ORDER BY PredictedRating DESC, ConfidenceScore DESC
LIMIT 10;
```

Table 10: Recommended Movies for user 672 using Collaborative Filtering

Rank	Recommended Movie	Predicted Rating	Confidence Score
1	Wallace & Gromit: A Close Shave (1995)	4.48	59.994
2	Reservoir Dogs (1992)	4.40	72.000
3	Goodfellas (1990)	4.35	71.980
4	Dr. Strangelove or: How I Learned to Stop Worrying and Love the Bomb (1964)	4.35	59.980
5	Taxi Driver (1976)	4.35	59.980
6	Fight Club (1999)	4.35	59.964
7	Citizen Kane (1941)	4.34	47.984
8	Raiders of the Lost Ark (Indiana Jones and the Raiders of the Lost Ark) (1981)	4.29	167.926
9	Fargo (1996)	4.29	155.964
10	A Clockwork Orange (1971)	4.29	83.964

The recommendations generated by the Collaborative Filtering model in Table 10 show a marked improvement in differentiation and quality compared to the Content-Based results.

4.1 Qualitative Comparison of Recommendations

Tables 9 and 10 present the Top-10 recommended movies for user 672 obtained through the content-based and collaborative filtering approaches, respectively. The two lists highlight distinct recommendation behaviors that reflect the intrinsic differences between the underlying models.

Content-Based Filtering (Table 9) focuses exclusively on the similarity of movie genres previously rated by the user. The resulting recommendations, such as *Beyond the Poseidon Adventure (1979)*, *The Mark of Zorro (1940)*, and *White Fang (1991)*, are strongly homogeneous in terms of thematic content—mainly adventure, action, and classic cinema. All predicted ratings are equal (4.61), indicating that every candidate movie shares a nearly identical level of genre similarity with the set of rated movies. Consequently, the weighted average calculation simplifies, causing the predicted rating to converge towards the user's overall weighted mean rating, thereby confirming the model's inability to differentiate items when input data is sparse. This shows the model's precision in capturing explicit preferences, but also its limitation in diversity, since it cannot introduce new types of content.

Collaborative Filtering (Table 10), instead, recommends titles such as *Wallace & Gromit: A Close Shave (1995)*, *Reservoir Dogs (1992)*, and *Fight Club (1999)*. These films differ more widely in genre - spanning animation, drama, and crime — because the algorithm leverages the opinions of similar users rather than shared attributes. Predicted ratings range from 4.29 to 4.48, indicating greater variability and finer differentiation in the strength of recommendations. The presence of the confidence score further quantifies how strongly the model trusts each prediction, adding interpretability to the results.

To further examine the relationship between the content-based and collaborative filtering models, a random sample of ten movies not previously rated by user 672 was selected. For these titles, both models had generated predicted ratings, allowing a direct comparison of their output. The goal of this analysis was to assess the level of consistency between the two recommendation strategies on the same set of items, rather than focusing on their respective Top 10 lists, which contained no overlap.

The following query was used to randomly select the movies and retrieve the corresponding predictions:

```
MATCH (u:User {id: 672})
MATCH (m:Movie)
WHERE NOT (u)-[:RATED]->(m)
  AND m.predictedCB IS NOT NULL
  AND m.predictedCF IS NOT NULL
WITH m ORDER BY rand() LIMIT 10
RETURN
    m.title AS Movie,
    round(m.predictedCB, 2) AS ContentBased_Rating,
    round(m.predictedCF, 2) AS Collaborative_Rating,
    abs(m.predictedCB - m.predictedCF) AS Difference
ORDER BY Difference DESC;
```

Table 11: Random set comparison of predicted ratings for user 672 using Content-Based and Collaborative Filtering

Rank	Movie	Content-Based Rating	Collaborative Rating	Difference
1	American Psycho (2000)	1.70	3.34	1.64
2	Benny & Joon (1993)	2.16	3.35	1.19
3	The Boys from Brazil (1978)	2.51	3.35	0.84
4	Empire of Passion (1978)	2.80	3.35	0.55
5	A Passage to India (1984)	3.88	3.67	0.21
6	Cry Freedom (1987)	3.50	3.67	0.17
7	Raising Victor Vargas (2002)	3.03	3.19	0.16
8	Manderlay (2005)	3.50	3.35	0.15
9	Mortal Kombat (1995)	2.73	2.85	0.12
10	Out of Africa (1985)	3.27	3.35	0.08

The results in Table 11 show that, for a random sample of ten unrated movies, the predicted ratings produced by the two models differ only moderately. While some titles such as *American Psycho (2000)* or *Benny & Joon (1993)* exhibit noticeable discrepancies (above 1.0 point), most predictions fall within a small margin of variation (below 0.3).

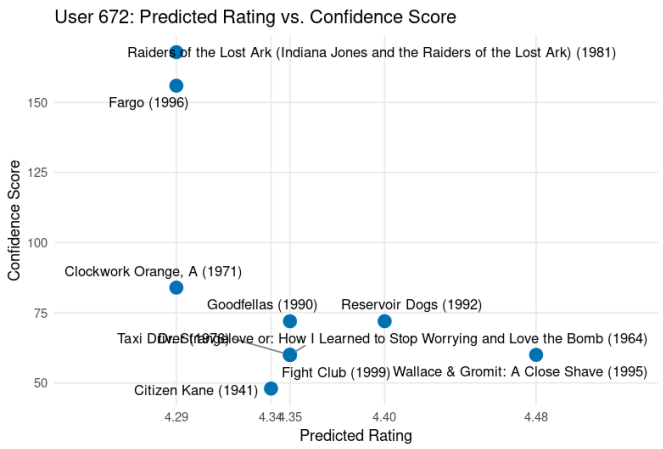


Figure 6: Relationship between Predicted Rating and Confidence Score for User 672 Recommendations.

Figure 6 visually confirms the successful differentiation provided by the Confidence Score. The plot highlights a clear distinction between the expected user preference (Predicted Rating) and the statistical robustness of that prediction (Confidence Score). Films such as *Raiders of the Lost Ark (1981)* and *Fargo (1996)* achieve the highest confidence levels, signaling a strong consensus among similar users, making them the most reliable recommendations. Conversely, titles with the maximum predicted rating, such as *Wallace & Gromit: A Close Shave (1995)*, show a moderate confidence score, which suggests that the high preference relies on a smaller or less correlated set of neighbors. This differentiation confirms that the Confidence Score is a vital tool for assessing the risk and interpretability of the Collaborative Filtering predictions.

Overall, this comparison confirms that both approaches capture complementary aspects of user preferences. The content-based model tends to provide stable and explainable recommendations aligned with known interests, while the collaborative method introduces broader diversity by incorporating patterns from similar users.

5 Synthesis and Insights from the Recommendation System

Throughout this analysis, two main recommendation approaches were implemented and evaluated: Content-Based Filtering and Collaborative Filtering, using a Neo4j graph database.

The content-based approach leveraged movie genres to estimate user preferences. Predictions were generated using a weighted Jaccard similarity between the genres of previously rated movies and candidate movies. A fallback mechanism, considering user, movie, and global averages, ensured robust predictions even in sparse data scenarios. Evaluation on a sample of ratings indicated moderate accuracy with some margins for improvement, particularly in capturing nuanced preferences beyond genre similarity. The model achieved a RMSE of 1.41 and a MAE of 1.21, reflecting a significant average deviation from actual user ratings.

The collaborative filtering approach utilized user–user similarities based on common ratings, computing predicted ratings through a weighted average of neighbors' deviations from their mean ratings. Recommendations were enriched with a confidence score reflecting the reliability of each prediction. This method captured latent patterns in user behavior, providing more diverse and finely differentiated suggestions compared to the content-based approach. Quantitatively, the CF model showed significantly superior predictive accuracy, achieving a RMSE of 0.747 and a MAE of 0.582. This improvement confirms the ability of the CF model to model user preferences more effectively.

A new user, User 672, was introduced to validate both models. The content-based model provided highly coherent but homogeneous recommendations, a predictable limitation when dealing with sparse user profiles, while the collaborative filtering model offered more varied suggestions with meaningful confidence differentiation. This comparison highlights the complementary strengths of the two approaches: content-based excels at precision when explicit preferences are known, whereas collaborative filtering introduces diversity and exploration by leveraging community knowledge.

Overall, the project demonstrates the effectiveness of combining attribute-based and user-based information to build a robust recommendation system. The successful implementation within the Neo4j graph environment, leveraging its native traversal and aggregation capabilities, provides a scalable and efficient framework.