Movie Recommendation project

Yebouet Cedrick-armel, Bédouret Baptiste

June 29, 2024

1 Introduction

In this project we did a movie recommendation system to predict the rating of a given movie depending on a given user.

2 Generation of data

In this project, we used *neo4j AuraDB* because it is much more powerful in computation time than *Neo4j Desktop*. We used the smallest movielens dataset containing 100k ratings and 3,600 tag applications applied to 9,000 movies by 600 users. 4 files were given:

- The first file was **movies.csv** which contains movie information. Each line of this file after the header row represents one movie, and has the following format: movieId,title,genres.
- The second file was **ratings.csv** the most important one. It contains all ratings. Each line of this file after the header row represents one rating of one movie by one user, and has the following format: userId,movieId,rating,timestamp. The lines within this file are ordered first by userId, then, within user, by movieId. Ratings are made on a 5-star scale, with half-star increments (0.5 stars - 5.0 stars). Timestamps represent seconds since midnight Coordinated Universal Time (UTC) of January 1, 1970.
- Identifiers that can be used to link to other sources of movie data are contained in the file **links.csv**. Each line of this file after the header row represents one movie, and has the following format: movieId, imdbId, tmdbId.
- Last file is **tags.csv** which is composed of tags. Each line of this file after the header row represents one tag applied to one movie by one user, and has the following format: userId, movieId, tag, timestamp. Tags are user-generated metadata about movies. Each tag is typically a single word or short phrase. The meaning, value, and purpose of a particular tag is determined by each user.

Then, we proceed to import the 4 files in *AuraDB* and we created the nodes and the relationships between them. The most important thing here is to build the node labels *Movie* and *User*. Then, we created User-Movie rating relationship *RATED*. See figure 1.

Figure 1: Nodes types and relationships

From the starting, nodes of type *User* had property *userId* and nodes of type *Movie* had properties *tile*, *movieId*, *tile* and *genres*. Then we created a label *Genre*, the Genre-Movie relationship *IS_GENRE_OF* and deleted the property *genres* from nodes of type *Movie* [A.1].

3 Process and visualizations

Predicting user ratings for movies is a classic problem in recommendation systems, and there are several approaches to tackle it. A common technique is **collaborative filtering**, which can be implemented in Neo4j using the Cypher query language.

3.1 Find similar users

First of all, **User-Based Collaborative Filtering** is a technique used to predict the items that a user might like on the basis of ratings given to that item by the other users who have similar taste with that of the target user. Indeed, in the code we created a recommendation system which is used to find movie recommendations by identifying movies that users who have rated a specific movie have also rated. The recommendations are sorted by the number of users who have rated both the initial movie and the recommended movie, with the top 25 recommendations displayed. This is a basic form of collaborative filtering for movie recommendations [A.2]. We display the result in figure 2 for this query and we get:

recommendation	usersWhoAlsoWatched
356	154
296	141
318	137
260	134
480	132
2571	123
780	122
593	121
110	117
1210	116
1196	113
150	111
589	110
364	107
1198	107
588	107
1270	106
648	106
32	104
1580	103
4306	102
2959	102
608	101
47	99
2858	99

Figure 2: Basic recommendation system

Now, let's consider how similar 2 users are using their movie ratings [A.3]. For this example, we took $userId = 7$ and $userId = 2$. The result is in figure 3 :

movieId	user1Rating	user2Rating
3578	1.5	4.0
8798	4.5	3.5
48516	1.0	4.0

Figure 3: Ratings of 2 users for similar movies

Thus, we decided to use **cosine similarity**. The cosine similarity of two users will tell us how similar two users preferences for movies are. Users with a high cosine similarity will have similar preferences.

$$\text{Cosine Similarity}(\mathbf{A}, \mathbf{B}) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|}$$

Cosine similarity is a common metric used in collaborative filtering to measure the similarity between two users. Cosine similarity ranges between -1 and 1, where -1 is perfectly dissimilar and 1 is perfectly

similar. In this step, we created a User-User relationship called *SIMILARITY* which compute the similarity between 2 users based on their cosine similarities [A.4]. In the figure 4, down below represents a little example of top 25 relationship similarities for a group of users.

```
MATCH p=()-[:SIMILARITY]->() RETURN p LIMIT 25
```

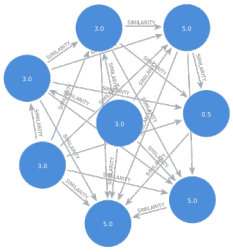

Figure 4: 25 similarities between 8 users

Thus, for *userId 7* and *userId 2* the similarity between them is quite good because the cosine similarity is equal to 0.798. In the following figure 5, we displayed 5 users which have the highest similarities with userId 1. The user in the middle of the image is *userId 1*.

```
MATCH (u1:User {userId: 1})-[s:SIMILARITY]-(u2:User)
WHERE s.cosine > 0.8
RETURN u1,s,u2 LIMIT 5
```

Figure 5: Top 5 users highest similarities with *userId 1*

3.2 Predictive ratings

To calculate a predicted rating for a specific movie for the target user based on the ratings of similar users, we followed a collaborative filtering approach. We calculated the weighted average of ratings given by similar users for the specific movie. The weighted average is based on the similarity between the target user and similar users. For example for $userId = 1$ and $movieId = 3$, we have a predicted rating of 3.245 [A.5].

3.3 Model evaluation

We evaluated the system by comparing actual and predicted ratings using the **Root Mean Square Error (RMSE)** metric. RMSE is a common evaluation metric for recommendation systems that measures the average prediction error. To evaluate it, we created a set of test samples with 10 users and 50 movies with actual user ratings (it's the ratings which the users gave to the movies). Afterwards, we used our collaborative filtering algorithm to predict ratings for the movies in the test set. Finally, we compared the predicted ratings with the actual ratings by calculating the RMSE [A.7]. It can be written as follow:

$$\sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

where y is the actual ratings and $\hat{y}$ is the predicted ratings. This metric measure the accuracy of our predictions.

4 Results

Finally, we found that on average, when using Cosine Similarity collaborative filtering method to predict ratings, we make an error of 0.93. The average prediction error, in the context of a discrete rating scale from 0 to 5 with increments of 0.5, appears to be acceptable. We can say that on average predictions are quite close of the actual ratings. Our predictions deviate from the actual ratings by slightly less than one rating point.

We tried the same thing using Pearson Similarity to take into account the fact that different users could have different mean ratings. However, performance was poor with RMSE of 21.72 [A.9].

A Cypher Codes

A.1 Set Genre label and IS_GENRE_OF relationship

```
MATCH (n:Movie) WITH n, split(n.genres, '|') AS genreList
UNWIND genreList AS genreName
MERGE (genre:Genre {name: genreName})
MERGE (genre)-[:IS_GENRE_OF]->(n)
REMOVE n.genres
```

Listing 1: Set Genre label and IS_GENRE_OF relationship

A.2 Recommendation system

```
MATCH (m:Movie {movieId: 1})<-[:RATED]-(u:User)-[:RATED]->(rec:Movie)
RETURN rec.movieId AS recommendation, COUNT(*) AS usersWhoAlsoWatched
ORDER BY usersWhoAlsoWatched
DESC LIMIT 25
```

Listing 2: Set recommendation system

A.3 Ratings of 2 users for similar movies

```
MATCH (user1:User {userId: 7})-[r1:RATED]->(movie:Movie)<-[r2:RATED]-(user2:User {userId: 2})
RETURN movie.movieId AS movieId, r1.rating AS user1Rating, r2.rating AS user2Rating
```

Listing 3: Ratings of 2 users for similar movies

A.4 Set SIMILARITY relationship and compute cosine similarity property

```
MATCH (u1:User)-[x:RATED]->(m:Movie)<-[y:RATED]-(u2:User)
WITH u1, u2, count(m) AS N,
sum(x.rating * y.rating) AS xyDotProduct,
collect(x.rating) AS xRatings,
collect(y.rating) as yRatings
WHERE N > 10
WITH u1, u2, xyDotProduct,
sqrt(reduce(xDot = 0.0, a IN xRatings | xDot + a^2)) AS xLength,
sqrt(reduce(yDot = 0.0, b IN yRatings | yDot + b^2)) AS yLength
MERGE (u1)-[s:SIMILARITY]-(u2)
SET s.cosine = xyDotProduct / (xLength * yLength)
```

Listing 4: Set SIMILARITY relationship and compute cosine similarity property

A.5 Compute the prediction of how an user will rate a given movie

```
// set input
MATCH (iu:User {userId: 1})
MATCH (m:Movie {movieId: 3})

MATCH (iu)-[iur:RATED]->(m)
MATCH (iu)-[s:SIMILARITY]-(u:User)

// Find the ratings of similar users for the specific movie
MATCH (u)-[r:RATED]->(m)

// Calculate the predicted rating for the target user
WITH iu, m, r.rating AS similarUserRating,
s.cosine AS similarity,
iur.rating AS actualRating
WITH iu, m, similarUserRating, actualRating, similarity,
(similarity * similarUserRating) AS weightedRating
RETURN iu.userId AS userId,
m.movieId AS movieId,
round(SUM(weightedRating) / SUM(similarity),3) AS predRating,
actualRating
```

Listing 5: Compute the prediction of how an user will rate a given movie

A.6 Compute predicted ratings given a set of users and movies

```
//iu for input user (predictions for 10 users for 50 movies max)
MATCH (iu:User) WITH iu LIMIT 10
MATCH (m:Movie) WITH m LIMIT 50
MATCH (iu)-[iur:RATED]->(m)
MATCH (iu)-[s:SIMILARITY]-(u:User)

// Find the ratings of similar users for the specific movies
MATCH (u)-[r:RATED]->(m)

// Calculate the predicted rating for the target users
WITH iu, m, r.rating AS similarUserRating,
s.cosine AS similarity,
iur.rating AS actualRating
WITH iu, m, similarUserRating, actualRating, similarity,
(similarity * similarUserRating) AS weightedRating
RETURN iu.userId AS userId,
m.movieId AS movieId,
round(SUM(weightedRating) / SUM(similarity),3) AS predRating,
actualRating
ORDER BY predRating DESC
```

Listing 6: Compute predicted ratings given a set of users and movies

A.7 Compute RMSE on predictions given a set of users and movies

```
//iu for input user (predictions for 10 users for 50 movies max)
MATCH (iu:User) WITH iu LIMIT 10
MATCH (m:Movie) WITH m LIMIT 50
MATCH (iu)-[iur:RATED]->(m)
MATCH (iu)-[s:SIMILARITY]-(u:User)

// Find the ratings of similar users for the specific movies
MATCH (u)-[r:RATED]->(m)

// Calculate the predicted ratings for the target users
WITH iu, m, r.rating AS similarUserRating,
s.cosine AS similarity,
iur.rating AS actualRating
WITH iu, m, similarUserRating, actualRating, similarity,
(similarity * similarUserRating) AS weightedRating
WITH iu.userId AS userId,
m.movieId AS movieId,
round(SUM(weightedRating) / SUM(similarity),3) AS predRating,
actualRating
//Compute and return RMSE
RETURN sqrt(avg((predRating - actualRating)^2)) AS RMSE
```

Listing 7: Compute RMSE on predictions given a set of users and movies

A.8 Set SIMILARITY's pearson property

```
MATCH (u1:User)-[r1:RATED]->(m:Movie)
WITH u1, avg(r1.rating) AS r1_mean

MATCH (u2:User)-[r2:RATED]->(m:Movie)
WITH u1, r1_mean, u2, avg(r2.rating) AS r2_mean

MATCH (u1)-[x:RATED]->(m:Movie)<-[y:RATED]-(u2)
WITH u1, r1_mean, u2, r2_mean,
collect(x.rating) AS xRatings,
collect(y.rating) AS yRatings,
count(m) AS N
WHERE N > 10
UNWIND xRatings AS xR
UNWIND yRatings AS yR
WITH u1, u2, sum((xR - r1_mean) * (yR - r2_mean)) AS nom,
sqrt(sum((xR - r1_mean)^2) * sum((yR - r2_mean)^2)) AS denom
MERGE (u1)-[s:SIMILARITY]-(u2)
SET s.pearson=nom/denom
```

Listing 8: Set SIMILARITY's pearson property

A.9 Compute RMSE on predictions given a set of users and movies (Pearson Similarity)

```
//iu for input user (predictions for 10 users for 50 movies max)
MATCH (iu:User) WITH iu LIMIT 10
MATCH (m:Movie) WITH m LIMIT 50
MATCH (iu)-[iur:RATED]->(m)
MATCH (iu)-[s:SIMILARITY]-(u:User)

// Find the ratings of similar users for the specific movies
MATCH (u)-[r:RATED]->(m)

// Calculate the predicted rating for the target users
WITH iu, m, r.rating AS similarUserRating,
s.pearson AS similarity,
iur.rating AS actualRating
WITH iu, m, similarUserRating, actualRating, similarity,
(similarity * similarUserRating) AS weightedRating
RETURN iu.userId AS userId,
m.movieId AS movieId,
round(SUM(weightedRating) / SUM(similarity),3) AS predRating,
actualRating
ORDER BY predRating DESC
```

Listing 9: Compute RMSE on predictions given a set of users and movies (Pearson Similarity)