STATISTICS FOR SMART DATA

NoSQL Project report
Movie recommendation with Neo4j

Hugo Brehier
Florentin Coeurdoux

July 22, 2020

Abstract

For this project our goal is to build and compare different recommendation algorithm based on the movielens dataset. Starting from a given user, we recommend the movies he is most likely to like. To compare the score the algorithm, starting from a given user and a given movie, we will calculates the predicted rating that the user would give to this movie. Then, we will evaluate systems by comparing actual and predicted ratings for a sample of users and movies.

Contents

1 INTRODUCTION

Neo4j is a native graph database built and designed for connected data. Especially considering recommendations increasing prevalence, many have found the underlying graph model helpful in supporting rich contextual recommendations that are understood intuitively and developed rapidly.

There are several reasons why graphs lend themselves so well to recommendations:

- **Explainability** : Graph models are easy to understand and visualize for non-technical users. Using graph traversals and pattern matching with Cypher make graph-based recommendations easier to understand and dissect than black-box statistical approaches.

- **Rapid Development** : Requirements change rapidly, and models need to adapt to fit these requirements. Neo4j is scheme-less, and you can refactor your graph to easily consider new data or different access patterns.

- **Personalization** : Contextualization: Neo4j is often used as a storage layer that connects data from multiple silos. Having quick access to everything you know about a user or product allows you to make more relevant recommendations.

- **Graph Algorithms** : Neo4j graph algorithms library lets you run similarity, community detection, and centrality algorithms directly on the database to enrich your graph and enhance your recommendations.

For our project we have used the movielens dataset. The dataset takes the form of 2 CSV files originating from the Grouplens site (grouplens.org/datasets/movielens/latest/).

This dataset is composed of tree types of nodes :

- User with attribute id
- Movie with attributes tittle and id
- Genre with attributes name

Those nodes are connected to each other by two types of relation ship :

- HAS_GENRE from movie to genre
- RATED from user to movie

We choose to implement tree types of algorithm from the most simple to the most complex.

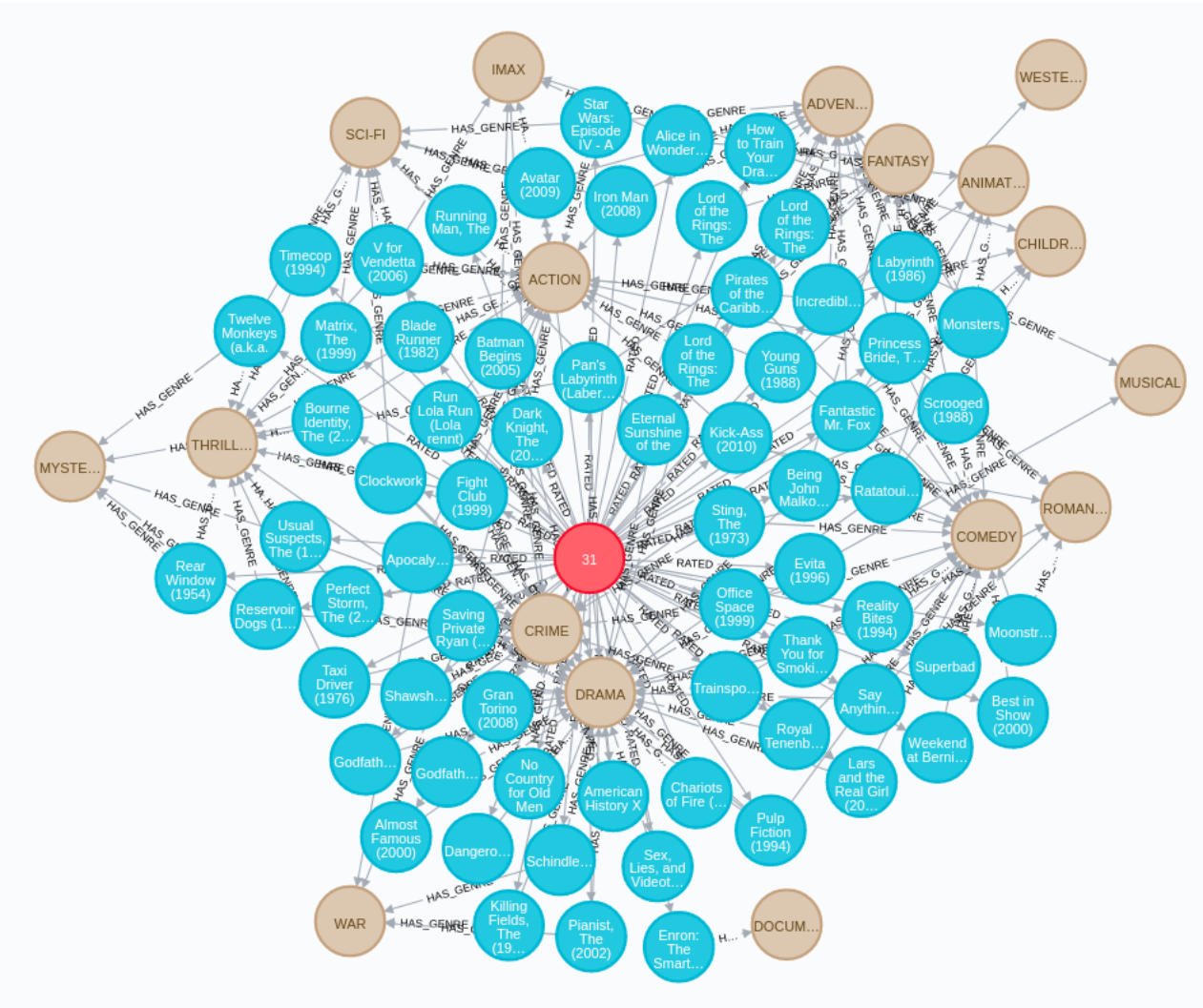

Figure 1.1: The MovieLens dataset

2 CONTENT-BASED APPROACH

The content-based approach system is, here, only based on the movie property 'genre'. It does not take user info into account; we recommend a movie to a user based on movies similarities. Since movies can have several genres, we recommend movies with most common genres. Thus, movies are associated one to another, independently of users.

MATCH (m:Movie title: "Fargo
(1996)")-[:HAS_GENRE]->(g:Genre)<-[:HAS_GENRE]-(rec:Movie)
WITH rec, COLLECT(g.name) AS genres, COUNT(*) AS commonGenres
RETURN rec.title, genres, commonGenres
ORDER BY commonGenres DESC

3 COLLABORATIVE-FILTERING APPROACH

Collaborative-filtering is a way to recommend new content to a user based on other users likings. If similar users, based on ratings, like another movie the user hasn't seen,it will be recommended to him. This method is based on a metric of similarity. Often used is the cosine similarity between non-null ratings of two users. Let **A** and **B** be the vectors of ratings given by two users A and B. Then their cosine similarity is defined as:

$$sim(A,B) = \cos\theta = \frac{\mathbf{A}\cdot\mathbf{B}}{\|\mathbf{A}\|\cdot\|\mathbf{B}\|}$$

In the recommendation system implemented below, another similarity is used.
Let $S_{A,B}$ be the subset of m movies belonging to M such that users A and B give similar ratings $r_A(m)$ and $r_B(m)$:

$$S_{A,B} = card\{m \in M : |r_A(m) - r_B(m)| \leq 1\}, r_A(m), r_A(m) \neq \emptyset$$

Then, the similarity of B to A is simply this measure controlled by the number of ratings of A :

$$sim_A(B) = \frac{S_{A,B}}{card\{m \in M : r_A(m) \neq \emptyset\}}$$

Note : $sim_A(B) \neq sim_B(A)$, since we normalize by the number of movies rated by one of the two users.

We compute a predicted rating for each movie a given user (here the user of id 1) has already rated, based on the average rating of the k (here 10) most similar users. If K is the set of k most similar users then the predicted rating of movie m given by A is:

$$pred(r_A(m)) = \frac{\sum_{i\in K} r_i(m)}{\sum_{i\in K} \mathbf{1}_{\{i:r_i(m)\neq\emptyset\}}}$$

Finally, the Mean Square Error of n predicted ratings over actual ratings is returned:

$$MSE = n^{-1}(\mathbf{pred} - \mathbf{actual})^T(\mathbf{pred} - \mathbf{actual})$$

The implementation is as following:

```
MATCH (m:Movie)<-[r:RATED]-(:User id:1)
with count(m) as cm
MATCH (u2:User)-[r2:RATED]->(m1:Movie)<-[r1:RATED]-(u1:User id:1)
WHERE (NOT u2=u1) AND (abs(r2.rating - r1.rating) <= 1)
WITH u1, u2, tofloat(count(m1))/cm as sim
order by sim DESC limit 10
with collect(u2.id) as ids
MATCH (u2:User)-[r2:RATED]->(m1:Movie)<-[r1:RATED]-(u1:User id:1)
where u2.id in ids
with m1,r1, avg(r2.rating) as pred
return sum((r1.rating-pred)*(r1.rating-pred))/count(pred)
```

Here are different values of MSE for different k with user 1 :

k neighbors	2	5	10	20
MSE	1.01	1.57	1.57	1.62

We have an minimum of MSE around $k = 2$ neighbors. If we repeat the process for, say, user 42, the optimum is at $k = 10$ neighbors.

4 CLUSTER-FILTERING APPROACH

Here we combine two approach, the collaborative-filtering system and the clustering approach. To build this system we had to export data from Neo4j to a statistical software. We choose to use R to build our clustering model. By using the package RNeo4j we manage to connect to our graph database, use the query as an R objet, and the data out of the query as a dataframe.
Once the data frame obtained and format we could launch a clustering algorithm. We choose to use the k-mean algorithm because of the sparsity of the data. Then we wrote in a CSV each user id and its corresponding cluster. Next, we imported this CSV is Neo4j and set the clusters as user attribute.

R code to build the clustering model :

```
library(RNeo4j)
library(dplyr)
library(tidyr)

graph = startGraph("http://localhost:7474/db/data/",
                   username="username", password="password")
query = "
MATCH (u:User)-[x:RATED]->(m:Movie)-[h:HAS_GENRE]->(g:Genre)
RETURN u.id AS user, m.title AS movie, AVG(x.rating) AS avg_rating
"
data = cypher(graph, query)

head(data, 5)
data = spread(data, movie, avg_rating, fill=0)
data[1:5, 1:5]

data$cluster = kmeans(data[-1], 50)$cluster
data.out = data[c("user", "cluster")]
head(data.out, 5)

write.csv(data.out, "cluster.csv", row.names = F)
```

Cypher code to import the CSV in Neo4j

```
LOAD CSV WITH HEADERS
FROM "file:///cluster.csv" AS row
MATCH (u:User id: toInt(row.user) )
MERGE (c:Cluster id: row.cluster )
```

CREATE (u)-[:USER_CLUSTER]->(c)

Recommandation Cypher code

```
MATCH
(u:Userid:31)-[r:RATED]->(m:Movie)<-[s:RATED]-(u1:User)-[o:RATED]->(rm:Movie),
(u)-[:USER _CLUSTER]->(c:Cluster), (u1)-[:USER _CLUSTER]->(c)
WHERE r.rating>3 and s.rating>3 and o.rating>3 and not exists ((u)-[:RATED]-(rm))
RETURN rm.title, avg(o.rating), count(o), (avg(o.rating)*count(o)) as scoring
ORDER BY scoring desc limit 5
```

Here we obtain :

"rm.title"	"avg(o.rating)"	"count(o)"	"scoring"
"Forrest Gump (1994)"	4.406332453825855	379	1669.999999999999
"Star Wars: Episode V - The Empire Strikes Back (1980)"	4.520569620253166	316	1428.5000000000005
"Silence of the Lambs, The (1991)"	4.270270270270268	296	1263.9999999999995
"Raiders of the Lost Ark (Indiana Jones and the Raiders of the Lost Ark) (1981)"	4.389860139860141	286	1255.5000000000002
"Inception (2010)"	4.5871369294605815	241	1105.5000000000002

To evaluate the system, we remove a well rated movie for a given user and then see if this movie appears to the recommendation system
As the result, most of the time the movie that we deleted from the database doesn't appear. This is due to the fact that no one in the same cluster had seen it before. This kind of model may be interesting for large database as he only compute scores for use in the same cluster (so it reduce the number of calculus).