Open Data - Project

Javier Moreno Fernandez
UPC Barcelona Tech
javier.moreno.f@est.fib.upc.edu

Victor Sendino Garcia
UPC Barcelona Tech
victor.sendino@est.fib.upc.edu

Meysam Zamani Forooshani
UPC Barcelona Tech
meysam.zamani@est.fib.upc.edu

Friday 19th June, 2020

This document supposes the deliverable of the project subject of Open Data corresponding to the Spring semester of the MIRI master degree during the course 2019/2020. This deliverable is divided according to the statement provided by the lecturer and it's accompanied by the corresponding resources containing the scripts required to reproduce the output of this work.

---

1 Business idea and identification of data sources.

A film recommender system that has parameters to recommend as the duration of the film, age of the grader, gender of the grader, genre of the movie, director, cast included in the movie

1.1 Purpose statement.

Our project is called "Film recommender" and it aims to help users find suitable movies for their film's preferences. To do so we create a recommendation system by merging different data sets that contain information regarding the movies and the users who rate movies. The type of data integration we have chosen is physical data integration by using property graphs (Neo4j). In the following sections, we will explain all the technical details starting from where we obtained our data sets, which is the most important information we are going to extract, how will we manage to load the data, entity resolution and everything else.

1.2 Data sources.

The datasets that we have chosen are:

1. **Movie Industry dataset contains Three decades of movies.**

   https://www.kaggle.com/danielgrijalvas/movies
2. **MovieLens data sets were collected by the GroupLens Research Project (Three datasets):**
   - **u.data** The full data set, 100000 ratings by 943 users on 1682 items (movies).
   - **u.item** Information about the items (movies).
   - **u.user** Demographic information about the users.

https://www.kaggle.com/prajitdatta/movielens-100k-dataset

The first dataset consists of movies and information related to them. The second dataset contains information about users rating movies and their information is extracted from 3 related files named: u.data, u.item and u.user. Each of which can be accessed by its corresponding URL. Preprocessing of the data is one of the main aspects of the work that we will explain below:

First dataset: In the first dataset, we only have 1 source file. First of all, we have removed some columns that we found irrelevant, such as budget or gross. Then, to preprocess it we made a Java project called IMBdPreprocess (`https://github.com/meysam24zamani/Project_MVJ_OD/tree/master/preprocess/IMDbPreprocess-master`). What this code mostly do is removing characters from the rows that may cause conflict and solve entity resolution issues. An example of the final rows is:

- Columbia Pictures Corporation, USA, Rob Reiner, Adventure, StandbyMe, 89, Stephen King, 1986

Being the items (in order) the producer company, the country, the director, the genre, the title, the duration of the film, the writer, and when it was recorded.

Second dataset: In the second dataset, as the files are distributed in 3 files, we made a Java project named MovieLensPreprocess (`https://github.com/meysam24zamani/Project_MVJ_OD/tree/master/preprocess/MovieLensPreprocess-master`) to preprocess all the files and then join them together. Going through each file, the u.user doesn't needed much preprocessing, just remove the ZIP code column. The rows on the file u.item have been preprocessed this way:

- An example of columns in the original one is: 1,Toy Story (1995),01-Jan-1995,0,0,0,1,1,1,0,0,...
- Columns in the Preprocessed one is: 1,ToyStory,1995,Animation,Children

We've removed the date from the title, and changed the binary method to tell the genres that the film has, to a much simpler way. Finally, we've joined the ids of item and user to make a single dataset containing all the data from the 3 sources. An example of a row is shown below:

- Jungle2Jungle, 1997, Children, Comedy, 49, M, writer, 2

which tells the film title, the year produced, the genres that it has, the age, gender, occupation and grade done by the user.

Finally, we have made a code called testentity (`https://github.com/meysam24zamani/Project_MVJ_OD/tree/master/preprocess/testentity-master`) whose functionality is to match the titles from one dataset to the other one. We've made this code to make us sure that we've got enough matches to work with. As we had a good quantity of matches, we keep going to the next section.

1.3 Graph family.

The graph family is selected based on the expectations we have for a given project. We are asked to merge datasets and to query on top of them, so we think that the best choice for us would be using the property graphs. Let's emphasize the main features we expect from the chosen graph family:

We've considered that it's fundamental to have high query performance, since we don't want the users to wait for the response, and this implicitly means that we should go for physical data integration. Furthermore, the benefits from the virtual data integration aren't interesting at all for our purposes. For instance, we don't need to keep the autonomy of the data since our datasets are open. A big disadvantage of the physical data integration is the high cost of data freshness. However, we've considered that the information in the graph may not have big changes during long periods of time and consequently it will be enough with monthly "windows updates". In this scenario data freshness will not be a big issue.

Following the same principles, the graph family we want to select should have a high query performance and must allow physical data integration. Consequently, that's why we have chosen property

graphs. Their strength resides in the expressivity, flexibility and high query performance. Knowledge graphs use the open-world assumption and are used to exchange information, but we are not interested in those features because we will use close world assumption and we will not exchange data.

Finally, even though some database platforms may fit into our approach, we've selected Neo4j, which is greatly optimized for queries on the relationships between data. It's a fast, scalable, reliable and schema-less database.

2 Global schema.

2.1 Integration schema.

To do recommendations, we want to keep information about the films and the reviewers.

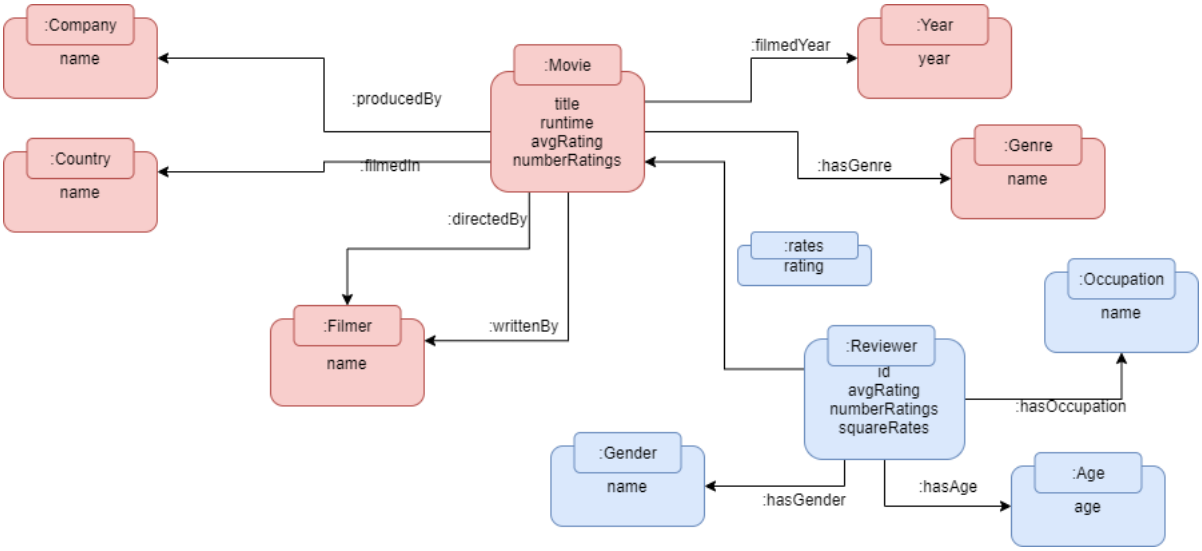

Figure 1: Integration Schema

This is how our integration graph looks like. As we mentioned before, we are using a physical data integration. The different colors are used to identify different sources from where this information has been extracted. In red we have the data that resides in the first dataset (moviedesc.csv), while in blue we have the one from the second (usermovies.csv).

First of all, the movie will have a title and its duration. Besides, it belongs to a certain genre, it is produced by a certain country and company and released in a concrete year. The movie will also have a director and a writer. This information is useful to characterize the films, and determine similarities between them. Genre is probably the most important feature to relate two films. Both director and writer allow us to find specific work from a likable filmer. The company, country, year and duration can be also used to do recommendation to a lesser extent.

Regarding the reviewer, it will have ID, gender, age and occupation. All the features seem important to find similarities between users and recommend them movies. In particular, males and females tend to prefer different movie genres. The age is useful to link reviewers with the movies of their generation, and we can find the appropriate ones for children as well. The recommendation of movies based on the occupation is a bit more diffuse, but in general, it is thought that people with higher intellect is more prone to see complex or alternative movies.

The movies and the reviewers will be linked by the reviews, that contain a rating between 1 and 5. The code for the Integration can be found in:

(`https://github.com/meysam24zamani/Project_MVJ_OD/blob/master/Cypher/code.ipynb`)

2.2 Data flow.

After the preprocessing step, we proceed with two different source CSVs that can be found in (`https://github.com/meysam24zamani/Project_MVJ_OD/tree/master/Final_datasets`)

1. `moviedesc.csv`: Contains all the information about the movie: Title, release year, director, writer, genre, company and country,
2. `usermovies.csv`: It contains all the reviews done by the users. From the reviewers, we have the ID, gender, age and occupation. We also have the movie title, release year and genre. Lastly, we have the movie rating done by the reviewer. It is a number between 0-5.

To begin, we load all the information about movies from the first CSV to our system. The information is split in different nodes because most of it is shared between movies (e.g. the same year or country will have multiple references) Then we create the relationships between them.

Then we load all the information about the reviewers from the second CSV. Again, we have different nodes for the attributes, since most of the nodes will be referenced often. Then we relate them through edges. We won't load the genres from this CSV because they are equivalent to the ones from the first dataset, and many movies are not labeled accurately.

The next step is to relate the movies and reviewers through the ratings. To do that, we read the second CSV again, and we match the movies by title and year, and the reviewer by its ID. Once this process has finished, we have all the nodes and edges integrated into our database.

2.3 Metadata to automate the exploitation process.

To automate the exploitation process we've added metadata both in the movies and the reviewer. For the movies, we've decided to enrich our data by adding as a field the total number of reviews done for that movie, and the average rating for the reviews. For the reviewers, we've followed the same approach, adding the number of reviews done by the user, and the average user rating. Apart from that, we have added the square root of the squared sum of the ratings done by the user (useful to calculate the cosine distance). This pre-calculations will be useful to run faster our recommender system. Besides, we will not have to redo these calculations very often, since our system will not change very frequently, and also because once a film or a user has done/received many reviews, the effect of adding a new one in the average is small.

3 Exploitation idea.

3.1 Purpose statement refined.

Since in our initial project we have added some metadata such as the total number of reviews done for a movie or the average rating for the reviews, now it is possible to retrieve more powerful data from our graphs and do recommendations more accurately. We considered three different approaches for the recommendations:

The code for the recommenders can be found in:

(`https://github.com/meysam24zamani/Project_MVJ_OD/blob/master/Cypher/code.ipynb`)

1. **First recommender: Top 20 of Movies**

   In the first recommender, we used a collaborative recommendation to match the films that have an average rating of 4. In the return we specify to get the count of rates that the films got, this is done so that if a film has only one rating that is a 5, we cannot be sure that for everyone will be a 5-star movie. For the ordering, we use a score metric that takes into account both the grading and the number of reviews. With this, we will find the most popular movies with the best ratings.

2. **Second recommender: Similar Reviewers**

   In the second recommender, we want to suggest films to a user based on the movies he likes. To do that we will use collaborative recommendation. We will estimate the rating that our user would do to the movies using the averages of the most similar users that have rated that movie. For our calculations, we will normalize the ratings. This means that for each review done by a user, we will subtract his average user rating. To begin, we find the 25 closest reviewers to our target user, in terms of movie ratings. The metric used is the cosine distance.

   Once we have this list of similar users, we look for all the movies that our target user has not rated yet. The score for these movies will be the average of the ratings done by the similar users. We will only consider the movies that were rated by at least 3 from our top 25 similar users. The movies will be ordered by a score, where the 80% corresponds to the average of the ratings and 20% corresponds to the number of ratings.

3. **Third recommender: Similar movies**

   In the third recommender, we've implemented a query that recommends the most similar movies to the latest one seen. It takes into account genre, director, year, writer, company and country. The similarity is calculated in terms of a weighted metric that considers as most important the genre and the director. The average rating of the movie is also added to the score, to have a metric that considers both the similarity and the quality of the movie. For instance, for the action movie Aliens directed by James Cameron in 1986, the best recommendation is Terminator 2, another action movie from 1991 directed by James Cameron and with good ratings as well.

3.2 Data analysis conducted.

In our first recommender, the analysis is quite simple since the answer will be the same for every user. The only tricky part is considering the weighing of the score. In our case, we've opted for doing $numberRatings * avgRating$. This way, the number of ratings plays a really important role, but the fact that we are choosing only the movies with more than a 4 as average ensures that we will always return movies with good ratings.

Regarding the second recommender, we've decided to use the 25 closest reviewers. It seems a reasonable number taking into account that we've a total of 945 reviewers. However, some inconsistencies may arise because we are weighing equally the rating for all the users. This means that in the cases where we can't find many similar reviewers, the furthest reviewers in our top 25 might perturb the calculations of the averages a bit. This could be addressed with a more complex calculation by averaging the ratings with the cosine distance.

To avoid some inconsistent scores because of the lack of reviews, we only consider the movies that have been rated by at least 3 reviewers from the closest 25. This value can be raised to have more confident recommendations but in a lesser number. In our final movie score, the 80% corresponds to the average of the ratings by the closest reviewers. The remaining 20% corresponds to the number of ratings. This way, we take into account both the expected rating by our user and the popularity of the movie.

To check that the recommendation system worked properly, we added ourselves in the second dataset as reviewers. The recommended movies are in general coherent, but some of them look a bit random. This happens due to two things. The first is because of the ratings in our dataset. In particular, we've observed that comedies (even the poor ones) tend to be a bit overrated. On the other side, critically acclaimed dramas or thrillers seem a bit underrated. The other situation depends on the method that we use. Sometimes is not easy to find very similar users (it is easier to find similar movies instead), specially in sparse situations like we are facing. And even if do there exist many nuances that complicate the problem.

For the third and last recommender, we look for similarities in movies. We use the genre, the director, the writer, the country, the year and the company in a weighted distance (40% genre, 20% director, 10% writer, 10% country, 10% year and 10% company). We consider that the gender and the director are the most influential in the movie. We only count year if the difference is less than 10 years. Because of the

sparsity of the data, if for instance we choose an action movie like Die Hard, the majority of the best results will be action movies as well. We also use the average rating of the movies, so we recommend both the most similar and better movies.

4 About proof of concept (PoC).

In terms of making a PoC, we have made a Java project that will make use of the recommenders that we defined earlier. This project can be checked in this link: (`https://github.com/meysam24zamani/Project_MVJ_OD/tree/master/PoC_OD`)

We have made this program in IntelliJ IDEA, using the plugin for Neo4j that allows the combination of Java with Neo4j and cypher querys. Connecting to the local database that we have created, the project can extract information from the Neo4j database and show an output to the Java console with the results of the query that we used. The exact information about how to use it will be put as a readme in the Github repository.