Open Data - Project

Prat Sicart, Joan
UPC Barcelona Tech
joan.prat.sicart@est.fib.upc.edu

Karriqi, Nejada
UPC Barcelona Tech
nedaaisel@gmail.com

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

Sunday 16th June, 2019

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 2018/19. 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 two data sources.

1.1 Purpose statement.

Our project is called "Barcelona Tourism Boost" and it aims to help tourists identify touristic attractions and the restaurants near to a touristic attraction they are visiting. To do so we create a recommendation system by merging different data sets that contain information regarding the tourist attractions and the restaurants in Barcelona. The type of data integration we have chosen is the 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- Tourist points of interest in the city of Barcelona.
2- List of restaurant equipment in the city of Barcelona.
3- TripAdvisor Restaurants Info for 31 Euro-Cities.
The first dataset consists of touristic attractions in Barcelona and information related to them. The second and the third datasets contain information about restaurants in Barcelona and not only. Each one of them can be accessed by its corresponding URL (Also you can find short description about variables in Tables 1 to 3. be inform that due to the large number of variables,we are just showing the more relevant and Highlight those which we used in our graph data model):
1- Tourist points of interest in Barcelona (Barcelona's City Hall Open Data Service):
https://opendata-ajuntament.barcelona.cat/data/en/dataset/punts-informacio-turistica
2- List of restaurant in Barcelona (Barcelona's City Hall Open Data Service):
https://opendata-ajuntament.barcelona.cat/data/en/dataset/equipament-restaurants

3- TripAdvisor Restaurants Info (Kaggle data scientists repository):
https://www.kaggle.com/damienbeneschi/krakow-ta-restaurans-data-raw

Variables	Description
address	Address of Attraction
address/_label	Label that shows its a field related to Address
city_label	Label that shows its a field related to City
city	it is a variable to show the city which is "Barcelona"
type1	Urban use or type of attraction
type2	Urban use or type of attraction (If it has more than one)
type3	Urban use or type of attraction (If it has more than one)
type4	Urban use or type of attraction (If it has more than one)
url	Website address of the attraction
district	Placement area of the attraction
barri	Neighborhood of the attraction
gmapx	geographical location(longitude)
gmapy	geographical location(latitude)
info	Email address of the attraction
phone	Phone number of the attraction
title	Name of the attraction
description	Short description of each attraction

Table 1: "Tourist points of interest in Barcelona" dataset

Variables	Description
EQUIPAMENT	Name of the restaurant
TIPUS_VIA	Street type
NOM_CARRER	Name of the street
NUM_CARRER_1	Street number (First one)
NUM_CARRER_2	Street number (First one)
CODI_BARRI	Code of neighborhood
NUM_BARRI	Name of the neighborhood
CODI_DISTRICTE	District area code
NOM_DISTRICTE	Name of district
CODI_POSTAL	Postal code of restaurant
POBLACIO	Name of the population area (Barcelona)
LATITUD	geographical location(latitude)
LONGITUD	geographical location(longitude)
TELEFON_NUM	Phone number of the restaurants

Table 2: "List of restaurant equipment in the city of Barcelona" dataset

Variables	Description
Name	Name of the restaurant
City	city location of the restaurant
Ranking	rank of the restaurant among the total number of restaurants in the city
Rating	rate of the restaurant on a scale from 1 to 5
Price Range	price range of the restaurant among 3 categories
Number of Reviews	number of reviews that customers have let to the restaurant
Reviews	2 reviews that are displayed on the restaurants scrolling page of the city
URL_TA	part of the URL of the detailed restaurant page, comes after (www.tripadvisor.com)
ID_TA	identification of the restaurant in the tripadvisor database

Table 3: "TripAdvisor Restaurants Info for 31 Euro-Cities" dataset

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 a 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 , but however, since we've considered that the information in the graph may not have big changes during long periods of time and consequently will be enough "windows updates" each months, we can conclude that this will neither be a big issue for us.

Following the same principles, the graph family we want to select, should have a high query performance and must allow physical data integration, and 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 on those features because we will use close world assumption and we will not exchange data.

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

2 Global schema.

2.1 integration schema.

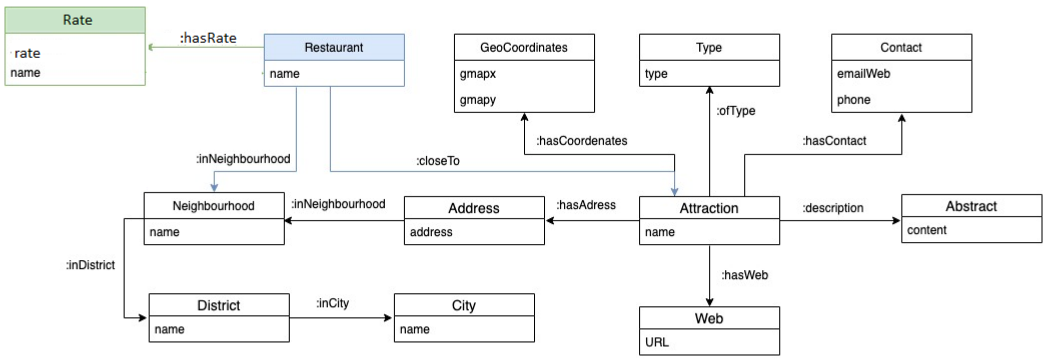

Figure 1: graph database schema

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. The main dataset in our project is ”Touristic point of Interest”. From this dataset we have extracted all the entities that are uncolored such as: Attraction, Web, Address, Type, and so on. This way the tourists have the opportunity to find all the information they need for a certain location. They an also find a contact form, from where they can have information about bookings. We merged this dataset with the restaurants to make a match if tourists after visiting an attraction would like to eat something near there. The way we connected them was through the location address, so if the tourist will search a restaurant near in a given neighborhood, our integrated system will show all the possibilities by showing at the same time the exact location, the rates (we will talk about them), open and closing hours and some more information. As you can see, restaurant entity has two different connection with the other dataset. The reason we did in this way is, since this query will be very frequent the performance will be significantly increased (even though the maintenance cost will be higher) so, if each time we query for the restaurants closer to an attraction we will just need to follow one edge, not three. The rates are extracted from the third dataset of restaurants in 31 European cities. We connected it with the restaurants in Barcelona and we are matching them through the restaurant name. The rates are real data but the users who have eaten on the restaurants will be auto-generated, since as we had imagined, there isn’t open source data regarding users. You will find more information about the users in the following sections (section 2.3).

2.2 Data flow.

As we know, Neo4j only accepts csv formats but not all the datasets we chose are in this format. The dataset containing all the touristic points of interest was in xml format, so we had to convert it into csv. To do so, we used an online converter where all you have to do is to upload the xml and it gives you the csv format. Even though two of our main datasets were downloaded from the same source, they had different special characters for the same words. Most probably the reason why this happend was because the persons who created those datasets used different Unicode Transformation Formats (UFT). In order to fix this problem, we used Atom, a code editor that has a debug command to decode the files. After solving the special characters problem, we used “Load csv” to load the data in Neo4j. The first dataset loaded was “Touristic points of attraction”. We prepared the cypher query to load the data and to create the relationships between nodes as ordinary. The same thing we did for the ”Restaurant equipment Barcelona” dataset. However, for the “TA restaurant curated” which is the dataset that contains restaurant information about 31 European cities, since it came from another source (kaggle and not from Barcelona Open Portal) the names weren't exactly the same, and we needed to solve this differences in order to link with the actual sources in the graph that mean the same but had different names, therefore, after loading the datasets, we performed the entity resolution process, for example, in one case we have as name of the restaurant "Tapes tapas" and on the other dataset the name is "Restaurant Tapes tapas", so the names didn't match at first. In order to solve this differences, we used an algorithm to compare the strings based on the algorithm called Levenshtein Algorithm, that works as follow: We remove the secondary words from the dataset. With secondary words I mean we removed the "Restaurant" from the name, then we create the code in such way that it would compare 3 first and 3 last characters/letters of every word. This method seemed to give a pleasant result, with and accuracy of more than 95 per cent. The algorithm performed to solve the entity resolution it's attached in the following figure.

Important note: There are some small differences between the schema and the code. Here is the equivalency hasPunctuation = hasRate; hasEatIn = hasEatenIn; Punctuation = Rate.

```
CREATE INDEX ON :Rate(name,rate);

USING PERIODIC COMMIT 500
LOAD CSV WITH HEADERS FROM "file:///TA_restaurants_curated.csv" AS row FIELDTER
MINATOR ','
WITH row
WHERE row.City = "Barcelona"
WITH row
WHERE NOT row.Name IS  null
WITH row
WHERE NOT row.Rating IS null
MERGE (a:Rate {name:row.Name, rate:row.Rating})
WITH a
MATCH (r:Restaurant)
WITH toUpper(r.name) AS RESTAURANT, toUpper(a.name) AS PUNCTUATION , a ,r
WITH replace(RESTAURANT, "RESTAURANT", "") AS parsedrestaurant, replace(PUNCTUATION,
"RESTAURANT", "") AS parsedpunctuation, a, r
WITH replace(parsedrestaurant, "CLUB", "") AS parsedrestaurant, replace(parsed
punctuation, "CLUB", "") AS parsedpunctuation, a, r
WITH replace(parsedrestaurant, "EXPERIENCE", "") AS parsedrestaurant, replace(par
sedpunctuation, "EXPERIENCE", "") AS parsedpunctuation, a, r
WITH replace(parsedrestaurant, "COOKING", "") AS parsedrestaurant, replace(parsed
punctuation, "COOKING", "") AS parsedpunctuation, a, r
WITH replace(parsedrestaurant, "GRACIA", "") AS parsedrestaurant, replace(parsed
punctuation, "GRACIA", "") AS parsedpunctuation, a, r
WITH replace(parsedrestaurant, "BAR", "") AS parsedrestaurant, replace(parsedpunc
tuation, "BAR", "") AS parsedpunctuation, a, r
WITH replace(parsedrestaurant, "LA", "") AS parsedrestaurant, replace(parsedpunct
uation, "LA", "") AS parsedpunctuation, a, r
WITH replace(parsedrestaurant, "EL", "") AS parsedrestaurant, replace(parsedpunct
uation, "EL", "") AS parsedpunctuation, a, r
WITH trim(parsedrestaurant) AS parsedrestaurant, trim(parsedpunctuation) AS parsed
punctuation, a, r
WHERE right(parsedrestaurant,3)=right(parsedpunctuation,3) AND left(parsedrestaurant
,3)=left(parsedpunctuation,3)
MERGE (r)-[:hasPuntuation]->(a);
```

Table 1: Code to solve entity resolution

2.3 Metadata to automate the exploitation process.

In order to give more potential to our graphs queries we wanted to add more metadata, therefore, since we used a dataset that contains rates/reviews for the restaurants, we found it reasonable to auto-generate some users, as the persons who gave those reviews. it was impossible to find a dataset with real information about the users so we decided to create an algorithm and auto-generated them. We called the new entity "User" and we gave to the users some basic attributes such as: name, age and nationality. As it can be appreciated in the file metadata inseriton from the files attached, to fill the users entity with information, we created 3 vectors, with each one of them data of (name, nationality, age) and finally merge them randomly to create the users entities. and finally, the match created was to relate them with the graph was: CREATE (r)-[:hasEatenIn]-(:User name:name, age:age, nationality:nationality); So the person who will check the reviews of a restaurant will also know who and how the restaurant was rated. In future works, we can extend the information with the type of food category, so we can gather information about a user that liked so much a certain kind of food, moreover, to know for instance, which restaurants make similar types of foods in order to make more accurate recomendations and so on. finally, here is the full picture of our integrated schema:

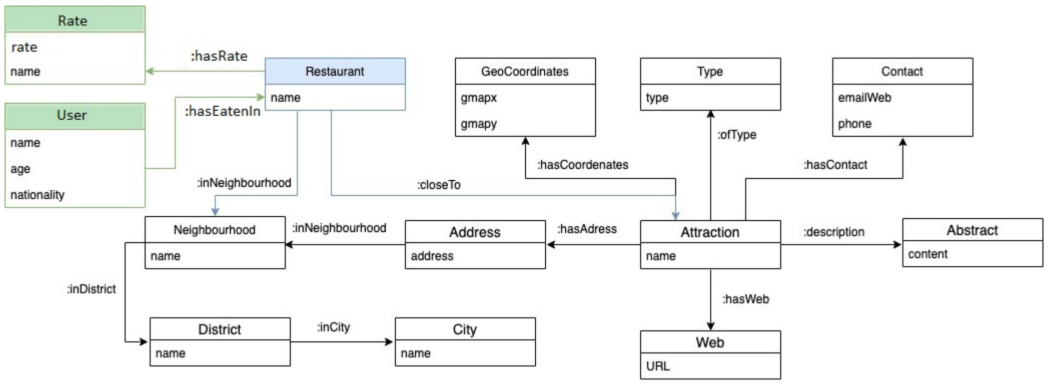

Figure 2: Final integrated schema

3 Exploitation idea.

3.1 Purpose statement refined.

Since in our initial project we have added some metadata such as ratings and users, it's possible to retrieve more powerful data form our graphs, and do recommendations more accurately. The queries we considered more important are the followings:

In the first one we've implemented the Jaccard algorithm and it will be used to recommend attractions in function of the attractions visited by the similarity of its tags. In particular, this similarity algorithm will identify any attraction 'a2' that has the same tags of a given attraction 'a1' that the user has just visited, and this way the system will be able to recommend it to the user. Moreover, with the Jaccard algorithm we also match different restaurants in function of the users that have eaten in. For instance, if almost 50 per cent of clients have eaten in Mc Donalds have also eaten in Burger King, to a user that has eaten in Mc Doanlds, we will recommend the Burger King restaurant.

The second is an ordinary query which generates all the restaurants near to an attraction that the user is visiting and orders them based on the rating they have, so the top rated restaurants are the ones that will be displayed the first.

And the third is Strongly Connected Components Algorithm which will be used to segment the restaurants in different partitions according to their relationships with the attractions and others restaurants. And it's interesting because since the relation between restaurants and attractions have been created just in relationship with the neighborhood they are in, restaurants that are very close to an attraction but that they aren't in the same neighborhood will also be retrieved in the same partitions if they are strongly related with the restaurants or the attractions.

3.2 Data analysis conducted.

Since our model is a recommendation-based system, we thought to analyze our system by making a "restaurant type" prediction by using a collaborative filtering. This means that we will combine the preferences of many users in order to recommend the best choice to one user. Our data analysis will be performed by using the TensorFlow framework. As we have stated before, the user information has been generated by us by using a probabilistic model and we are aware of the facilities of synthetic data (no need for data cleaning). Each person has eaten at least at one of the restaurants(the restaurant where they have eaten was chosen randomly).

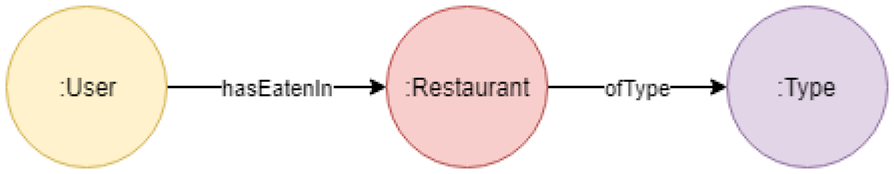

Figure 3: Note: this graph is used for visualization purposes only. In the original dataset the type is not an explicit node, it's a restaurant's attribute.

For training purposes, we will assign an amount of 80% (randomly selected) of the entire data. The remaining (20%) will be used to evaluate the model, testing process. We will use Cypher to retrieve all the users that have eaten in a restaurant. All the obtained rows will be converted in the correct format for TensorFlow (input, output_score) and then we construct the tensor dataset through a high-lever TensorFlow API which allows the framework to transform shuffle and batch our data for training process. When we finish with data distribution, we will create an input function which doesn't require parameters. During the training process, the input function gives to TensorFlow the ability to create a dataset any time that it ends and wants to restart. Now that the input function is ready, we need the model function. The model function is a function coded in python and its used by TensorFlow to allow the framework to instantiate our model when is needed and returns an estimator spectrum (EstimatorSpec). Having prepared the model function and the input function, we can start training our model by using the train_and_evaluate method provided by the Estimator API. Everything is set, so we run the training and evaluation process. The output value indicates the accuracy of the model. If the result is not very satisfiable, we can use random walks to improve the training process. Random walks are a way to represent the graph connectivity. It starts by choosing a random node of the graph and then tries to traverse all the nodes connected to each other until it explores all the connected nodes. The process is repeated many times and each walk gives a fixed length path which represents a sample of the graph.