ForMe: SDM Project

Yi Wu
Facultad de Informática de Barcelona
Universitat Politècnica De Catalunya
yi.wu@estudiantat.upc.edu

Davide Rendina
Facultad de Informática de Barcelona
Universitat Politècnica De Catalunya
davide.rendina@estudiantat.upc.edu

Contents

1 M1 Project Description

ForMe is an establishments recommender system based on user's preference. In our app, the user is able to add their preferences in the form of tags which indicate factors such as food or type of cuisine. The recommendations are then made based on this tags, giving a personalised experience and allowing users to find establishment that match their preferences. As such, it is crucial for our purpose to exploit he underlying data to make tailored recommendations to the users. In this context, a graph database would allow to define relationships between data and quickly query those relationships.
Therefore, it would highly benefit to use the data obtained from the user's reviews in GoogleMaps and Yelp used in the BDM project, in the form of keywords, to build a graph on which to perform analytical operations.
The aim is to build a recommendation system similar to the one built for publications in the first lab of this course.

2 M2 Graph Family Justification

We decided to build a Labelled Property Graph (LPG) which would be used to perfom graph analytics, such as finding restaurants that a certain user's friends like and thus recommending them to the the user. In addition, LPG would allow us to uniquely identify instances of relationships of the same type (e.g. a user visited a certain establishment twice or a restaurant offers more than one cuisine) and qualify instances of relationships (allowing attributes in relationships, which is the case of the relationship 'prefers' in our case), things that using RDF would not be possible as noted by Barrasa.

3 M3 Graph Design

A visual representation of the graph is presented in the Figure below:

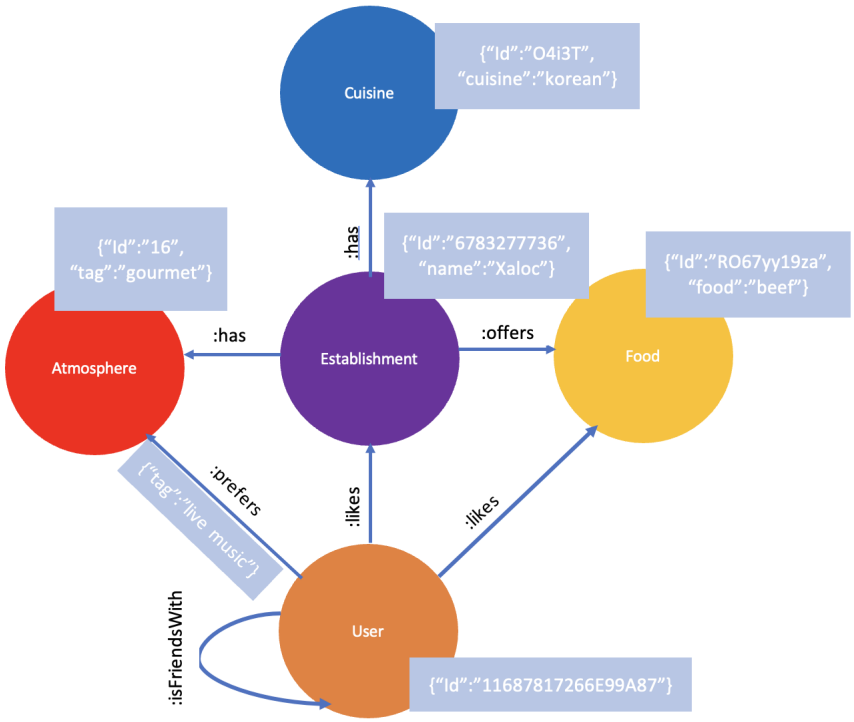

Figure 1: ForMe LPG

Metadata is represented by colored circles while data is presented by grey boxes.

4 M4 Data Flow

An important aspect of this project is how the data flows and is integrated into the graph. The data used to populate the graph has been extracted following the Machine Learning tasks performed in the BDM part for the project. In particular, from text analysis and classification we extracted tags from words used by users in their reviews of establishment, which would define the type of food, the atmosphere and the cuisine.

The data was first loaded using a Python driver for Neo4j and following the instructions provided by Neo4j [c]. Single property indexes were created on nodes to speed up the search for nodes when loading the edges, as suggested in Neo4j [b].
Full script to load the data into Neo4j can be found in the attached SDM_ProjectP1.ipynb file. This is adapted from the script used in lab 1 (ref *Lab1_HernandezRendina*). Schema of the data loaded on Neo4j displayed using the APOC plugin can be found in the Figure below.

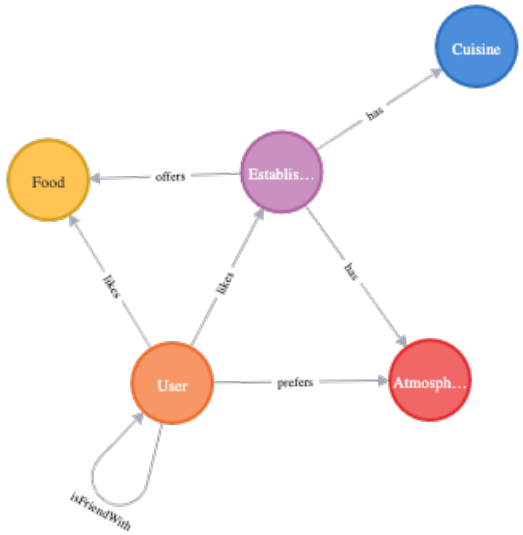

Figure 2: Graph schema

Once ForMe has been deployed, users will able to insert tags related to the represented categories (i.e. Atmosphere, Food and Cuisine) which will be integrated into the existing graph.

5 M5 Graph Exploitation

To Exploit the Graph we propose two recommendations algorithms, which are widely used in the context of graph databases as shown by Marrancone. These algorithms are:

- **Content-based Filtering**: which considers similarities between products.
- **Collaborative Filtering**: which considers users behaviours toward products (like, purchase etc), with the assumption that similar users might behave similarly.

5.1 Data

Based on the data loaded into our graph database, as illustrated in the Data Flow section, we will focus on the following relationships:

In the first case we have node User(orange) which likes a node Establishment(pink) which has a node Cuisine(blue). This is expressed in an instance below.

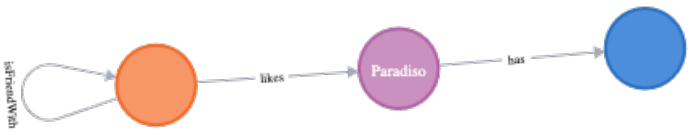

Figure 3: User-likes->establishment-has->Cuisine

In the second case we have a node User(orange) which like a node Establishment(pink) which is liked by another node User(orange). An instance is illustrated below.

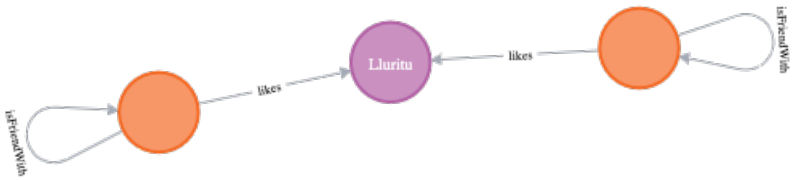

Figure 4: User-likes->establishment<-likes-user

5.2 Content-based Filtering

As already mentioned, this algorithm aims to suggest products that are similar to other products a certain user liked.
In this case, this would be able to exploit the graph using the relationship expressed in Figure 3 in which an user likes an establishment which has one or more types of cuisines. The aim is to suggest to the user establishments that have similar cuisines to the establishments they already liked.
A visual representation of the algorithm is shown in the Figure below.

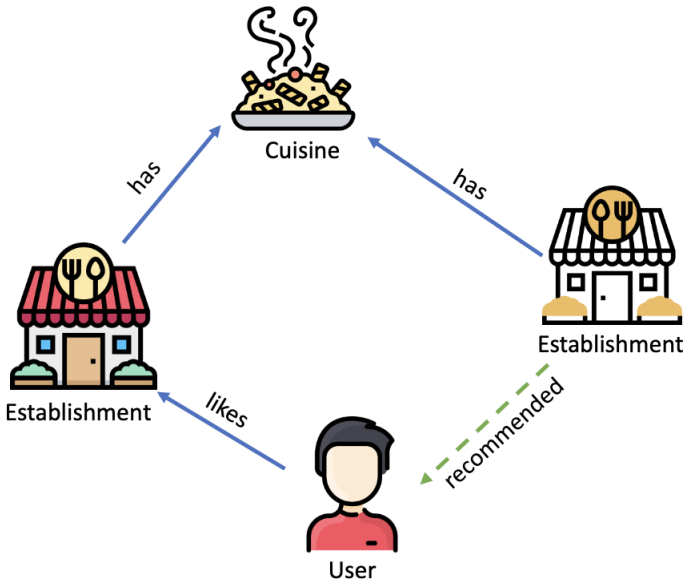

Figure 5: Content-based filtering ForMe

To define the similarity score of our recommendations we used the Jaccard index which as explained by Marrancone "is a number between 0 and 1 that indicates how similar two sets are. The Jaccard index of two identical sets is 1. If two sets do not have a common element, the Jaccard index is 0. The Jaccard is calculated by dividing the size of the intersection of two sets by the union of the two sets". In our case, each establishment (i.e. bar, restaurant) offers one or more type of cuisines which are marked by some keywords (e.g. "fusion"). First we get the cuisine of the establishments that the user liked. Following, we find other establishments that also offer those type of cuisine. The higher the number of matching cuisines between two establishments, the higher the Jaccard similarity.
Finally we are able to recommend to the user other establishment looking at calculated the similarity score.

5.3 Collaborative Filtering

Collaborative Filtering Recommendation could be either based on similar users or similar establishments. Here we implement the former. Where the edge 'likes' has no weights.

```
(:User)-[:likes]->(:Establishment)
```

A visual representation of the algorithm is shown in the Figure below.

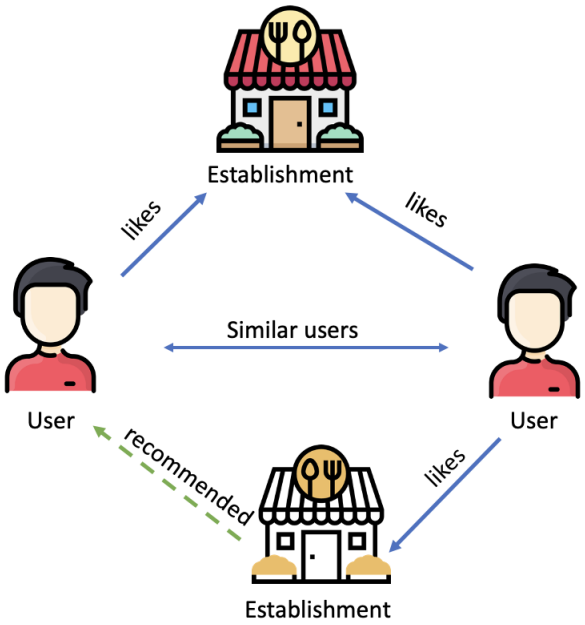

Figure 6: Collaborative filtering ForMe

6 M7 Proof of Concept

For deploying a Proof of Concept of the proposed LPG model we used Neo4j, this for the following reasons:

- Native graph storage and processing, therefore optimized for graph operations.
- Cypher query language, which is intuitive and we were already familiar as it was used to implement the first Lab of this course.
- Staggering loading speed of huge data sets, which will be our case in the future as the app expands its user base.
- Graph Data Science Library, which allows implementation of algorithms and Machine Learning tasks.

6.1 GitHub repository

All scripts and data used for this project can be found in our public GitHub repository available at $\text{SDM}_{project}$

6.2 Content-based Filtering

As mentioned already in the previous section, Jaccard similarity was used to asses how similar two establishments are based on Cuisine. These establishments will be then recommended to the user.
The Cypher query for this recommendation is displayed below:

```
MATCH(u:User{id:
    "116878172175502198628"})-[:likes]->(s:Establishment)-[:has]->(c:Cuisine)<-[:has]
-(z:Establishment)
WHERE NOT EXISTS ((u)-[:likes]->(z))
WITH s, z
MATCH (s)-[:has]->(sc:Cuisine)
WITH s, z, COLLECT(DISTINCT id(sc)) AS s1, COLLECT(DISTINCT sc.cuisine) AS c1
MATCH (z)-[:has]->(zc:Cuisine)
WITH s, z, s1, c1, COLLECT(DISTINCT id(zc)) AS s2, COLLECT(DISTINCT zc.cuisine) AS
    c2
WITH s, z, s1, s2, c1, c2
RETURN s.name as UserPreference, z.name as Recommended, c1 as
    UserPreferenceCuisine, c2 as RecommendedCuisine,
    gds.alpha.similarity.jaccard(s1,s2) AS similarity ORDER BY similarity DESC
```

First, we query establishment that a certain user has seen, extracting the Cuisine. This cuisine is then used to find Establishments that the user has never seen. Then, we create two sets based on the cuisines of the collected establishments. Finally, we compare the two sets using the Jaccard similarity index available through the GDS library of Neo4j. The result, ordered by the Jaccard attribute, is displayed below:

UserPreference	Recommended	UserPreferenceCuisine	RecommendedCuisine	similarity
"Cecconi S"	"Gocce Di Latte"	["italian"]	["italian"]	1.0
"Cecconi S"	"Italian District"	["italian"]	["italian"]	1.0
"Dos Palillos"	"Brugarol"	["japanese", "fusion"]	["fusion", "creative", "japanese"]	0.6666666666666666
"Dos Palillos"	"Kame House Kitchen BCN"	["japanese", "fusion"]	["fusion", "japanese", "thai"]	0.6666666666666666
"Tickets"	"Rasoterra"	["creative"]	["vegetarian", "creative"]	0.5
"Tickets"	"Somorrostro"	["creative"]	["creative", "seafood"]	0.5

Figure 7: Sample content-based recommendations

6.3 Collaborative Filtering

To represent the nodes similarity among users, a FastRP embedding and KNN method is implemented as illustrated by Neo4j [a].

First, create a gds graph and implement the fastRP embedding. There are 3 embedding methods offered by neo4j, fastRP, node2vec and graphSage. Although the last two are based on neural network, graphSage is slow and node2vec doesn't support node property embedding, thus we chose fastRP.
Among the modes of stat, stream, mutate and write, we chose the mode mutate, since it has

the effect of updating the named graph with a new node property containing the embedding for that node, which means that the embedding vector can be written into our original graph as a property named 'embedding'.
When setting up the parameters of fastRP, according to our node size, which is about 10 to the power of 5, so embeddingDimension is set at 256. And iterationWeights is set as [0.8, 1, 1, 1], the latter iteration has more weights.

```
//Create gds graph user-likes->establishment
CALL gds.graph.create(
  'userLikesEstablishment',
  ['User','Establishment'],
  {
    likes: {
    }
  }
  )
//Transform node to vector and write in property
CALL gds.fastRP.mutate('userLikesEstablishment',
  {
    embeddingDimension: 256,
    mutateProperty: 'embedding',
    iterationWeights: [0.8, 1, 1, 1]
  }
)
YIELD nodePropertiesWritten

CALL gds.graph.writeNodeProperties('userLikesEstablishment', ["embedding"],
    ["User"])
```

Second, compute the node similarity via gds KNN method. The aim is to generate the top 5 most similar neighbors nodes for each user. The result is written back to the orginal graph through creating an edge between one user and its 5 most similar user neighbors, with the edge label as 'SIMILAR' and the edge property 'similarityScore'.

```
CALL gds.beta.knn.write('userLikesEstablishment', {
    topK: 5,
    nodeWeightProperty: 'embedding',
    randomSeed: 42,
    concurrency: 1,
    sampleRate: 1.0,
    deltaThreshold: 0.0,
    writeRelationshipType: "SIMILAR",
    writeProperty: "similarityScore",
    maxIterations:100
})
YIELD nodesCompared, relationshipsWritten, similarityDistribution
RETURN nodesCompared, relationshipsWritten, similarityDistribution.mean as
    meanSimilarity
```

Now the graph is updated as follows, with a new relationship 'SIMILAR' among User nodes(the yellow one). We did embedding for Establishment as well, that's why it shows 'SIMILAR' too. But in this report, we focus on user-based recommendation.

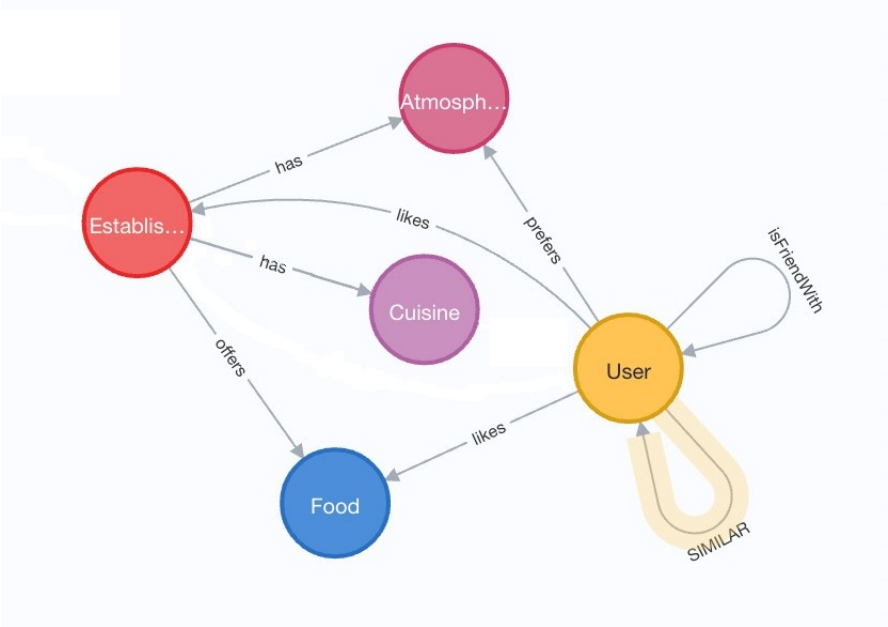

Figure 8: Added edges named 'SIMILAR' among users

Finally, output the recommendation results. Since for each target User, we got 5 similar users nodes already, the establishments will be recommended if they fall into what neighbors like, but have not been liked by the target User.

```
MATCH (m:User)-[:likes]->(e1:Establishment)
WITH m,COLLECT(e1) AS historyLikes
MATCH (m)-[r:SIMILAR]->(n:User)-[:likes]->(e2:Establishment)
WHERE NOT e2 IN historyLikes
RETURN ID(m),COLLECT(DISTINCT e2.name)
```

The result {Node ID, Recommended Establishment} shows as follows.

89548	["9 Nine","Can Xurrades","Raffaelli Ristorante Italiano","Cecconi S","BON","Vic Braseria"]
203166	["Ekaterina","Nuestra Brasa Garibaldi","Thai Zaap","Cecconi S","La Piazzetta GranVia","EdeNova","Baritimo"]
48485	["Arume","Elsa Y Fred"]
24439	["Cecconi S","Lasarte"]
16257	["Gourmet Sensi","Gallo Nero","Oria By Martin Berasategui","Mamajin"]

Figure 9: Sample collaborative recommendations

References

J. Barrasa. Rdf triple stores vs. labeled property graphs: What's the difference? URL https://neo4j.com/blog/rdf-triple-store-vs-labeled-property-graph-difference/. 2

L. Marrancone. How to create recommendation engine in neo4j. 5, 5.2

Neo4j. fastrp-knn-example, a. 6.3

Neo4j. Indexes for search performance, b. 4

Neo4j. Get started, c. URL https://neo4j.com/blog/rdf-triple-store-vs-labeled-property-graph-difference/. 4