Constructing an RDF Knowledge Graph Knowledge Representation & Reasoning-Mod. 2 Artificial Intelligence 2022-2023 Final Project

Alessandro Ghiotto

1 Datasets and Task

The dataset consisted of tabular data, two CSV files, the first is "albumlist.csv" with dimension (500, 6), and the second is "rym_top_5000_all_time.csv" of dimension (5000, 9).
The main task is to create a RDF Knowledge Graph that follows the next points:

1. create an RDFLib Graph from data contained in CSV files,
2. integrate the data with DBpedia's,
3. gathering information from unstructured data,
4. include a small ontology and a genres taxonomy,
5. materialize inferences,
6. query the Graph.

2 Workflow

Load the CSV files in two pandas DataFrames and join them

After some analysis on the data contained in the CSV files, I know that in the second dataframe I have some row that have the same pair ['Album', 'Artist Name'], and that the value in the Album column is the same as the value in the Artist column, so I have assumed that the rows of this kind have the name of the artist inserted in both columns.
I have renamed this albums, setted the couple ['Album', 'Artist'] as index (since after our changes each of this couples is unique) and merged on the index, that is faster then merging on simple columns. The result is the outer join of the two DataFrames.

Create an RDFLib Graph and add triples from the data of the dataframes

I started with creating an RDFLib Graph and the namespace "df" that I will use to add the elements of our merged dataframe to the graph. Then I have created some ad-hoc function that will transform the data in the merged dataframe to useful URIs or Literals. For example for the Artist_Name I will replace the spaces with underscores, the date will be transformed in the format YYYY-MM-DD, and so on...
Finally I have iterated over the tuples in the merged dataframe and added all my informations with the just created functions and the RDFLib method add().

Link the URIs of my current graph with DBpedia

Link the URIs of my current graph (which use the "df" prefix) with DBpedia. I tried 3 ways to match the entities that I have stored in my graph with the corresponding DBpedia resources (prefix "dbr"), this ways are the following ones:

- replace "example.org/DataFrame/" with "dbpedia.org/resource/", that means searching for entities of the kind:
  "dbr:The_Dark_Side_Of_The_Moon",
- replace "example.org/DataFrame/" with "dbpedia.org/resource/"
  and adding "_(Album)", that means searching for entities of the kind:
  "dbr:The_Dark_Side_Of_The_Moon_(Album)"
- searching every entity that has a "rdfs:label" that is equal to the corresponding literal of the name of the album
  ?album_uri rdfs:label "The Dark Side Of The Moon".

I have used all three methods for searching the corresponding album and only the first and the third for searching the corresponding artist. The first two methods employ an ASK query and the third a SELECT query (to the DBpedia endpoint).

Enrich my information with DBpedia

Enrich my information by querying DBpedia with some CONSTRUCT query, that creates a graph that I save in my memory and that I will merge with my current graph. Mainly I added information regarding the members of the bands, the hometown of this members and the bands.

Gathering Information from Wikipedia

Gathering information from unstructured data, like extracting the role in a band of a member, from the first 200 words of the Wikipedia summary of the article that concern the member that I'm analyzing. First I built a set with an exhaustive number of roles, and then checked for some simple pattern matching between the words in the summary and the roles in my roles_set. The summary of the article from Wikipedia is extracted using Wikipedia-api.

Domain and range restictions

Adding the schema with some domain and range restrictions for my properties, that will be useful for materializing inferences when I will apply the RDF_OWLRL reasoner.

Genres taxonomy

Adding a genre taxonomy and the rule that implies the following entailment pattern: if I have ("an_album" df:genre "a_genre"), then I add ("a_album" rdf:type "a_genre"). For building the genre taxonomy I have proceeded in this way:

- do an iteration over the Albums (subjects of triples of the kind (album, df.genre, None)),
- store each genre and subgenre (linked to the current album) in two lists,
- add the triple (a_genre, RDFS.subClassOf, a_subgenre) whenever the string "a_genre" is in the string "a_subgenre".

Materialize inferences

For materializing inferences I have added some further schema rules, that make use of the OWL and the RDFS vocabulary (for example (dbp:hometown rdfs:subPropertyOf mi:foundingCity)), so that I can easily add the inferences by simply running the RDFS_OWLRL reasoner. While for other entailment pattern that are slightly more complex I have applied UPDATE query on my graph, for example the one that infer the genre of the band from the albums:
INSERT { ?artist mi:genre ?genre } WHERE { ?album df:genre ?genre; df:artist ?artist }

Query the Graph

I run some SPARQL query so that I can see the usefulness of having added information from DBpedia and the result of the reasoning on my data. I can query directly using the predicates that I have added previously, instead of simulating an entailment pattern with my SPARQL query.

3 RDF Knowledge Graph

Starting KG containing only the data from the two CSV files

Classes: df.Album, df.Artist, df.Genre.
Predicates: df.artist (domain album, range artist), df.releaseDate, df.year, df.genre, df.subgenre, df.descriptors, df.rsRanking, df.rymRanking, df.averageRating, df.numberOfRatings, df.numberOfReviews.
We will have something of this kind:

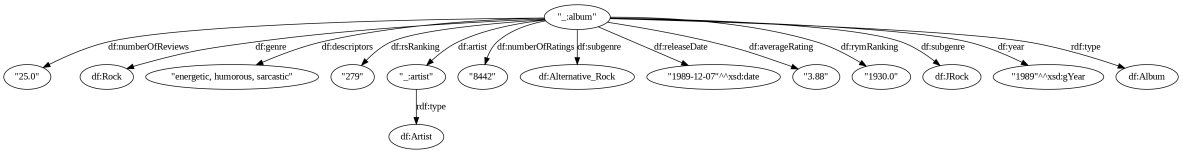

Figure 1: Starting RDF KG

KG linked with the DBpedia's entities

Here I have linked with the predicate owl:sameAs my URIs with the corresponding dbr version.

```
df:an_album owl:sameAs dbr:corresponding_db_IRI.
df:an_artist owl:sameAs dbr:corresponding_db_IRI.
```

Figure 2: owl:sameAs and dbr entities

KG enriched with new DBpedia's information

Here I have added information regarding band members, former band members, birthplace of the members and founding place for the bands.

```
dbr:a_band a dbo:Band;
           dbo:bandMember ?bandMember;
           dbo:formerBandMember ?formerBandMember.
dbr:a_band dbp:hometown ?foundingCity.
?foundingCity dbo:country ?foundingCountry.
dbr:a_member dbo:birthPlace ?cityOfBirth.
?cityOfBirth dbo:country ?countryOfBirth.
```

Figure 3: new DBpedia's information

Entailment schemas

```
# ------------------------------------------------ #
|                  SCHEMA  1                     |
# ------------------------------------------------ #
df:artist rdfs:domain df:Album ; rdfs:range df:Artist .
df:genre rdfs:domain df:Album ; rdfs:range df:Genre .
df:subgenre rdfs:domain df:Album ; rdfs:range df:Subgenre .
dbo:artist rdfs:domain dbo:Album ; rdfs:range dbo:Artist .
dbo:bandMember rdfs:domain dbo:Band ; rdfs:range dbo:BandMember .
dbp:hometown rdfs:domain dbo:Band ; rdfs:range dbo:City .
dbo:birthPlace rdfs:domain dbo:Person ; rdfs:range dbo:City .
dbo:country rdfs:domain dbo:City  ; rdfs:range dbo:Country .
wiki:role rdfs:domain dbo:BandMember ; rdfs:range wiki:MusicRole.
```

Figure 4: Domain and range restrictions

```
df:a_genre rdfs:subClassOf df:a_corresponding_subgenre.
df:an_album rdf:type df:a_corresponding_genre
```

Figure 5: Genres taxonomy

```
# ------------------------------------------------ #
|                    SCHEMA 2                      |
# ------------------------------------------------ #
dbp:hometown rdfs:subPropertyOf mi:foundingCity.
dbo:birthPlace rdfs:subPropertyOf mi:cityOfBirth.
dbp:hometown rdfs:subPropertyOf mi:bornIn.
dbo:birthPlace rdfs:subPropertyOf mi:bornIn.
dbo:country rdfs:subPropertyOf mi:bornIn.
mi:bornIn rdf:type owl:TransitiveProperty.
mi:subgenre rdfs:subPropertyOf mi:genre.
```

Figure 6: Other entailment rules

4 Conclusion

I tried to create a KG that resembled a real one, for example the DBpedia KG, so instead of analyzing the data before adding them, I wanted to incorporate them with the predicate owl:sameAs. For example If I had searched for the DBpedia's corresponding URIs and added only them, instead of adding at the beginning all my data with the "df" prefix I could have obtained a much more light graph. But instead doing in this way (create my own prefix and then incorporate with DBpedia's information), I can maintain the original shape and my original data from the two CSV files, that correspond to the base of my KG, and then adding new information that are like globally recognised (since are DBpedia's URIs) that makes the graph not only mine, but like it is globally connected for every user.
I think that the main limitations of my program is the lack of efficiency in the way in which it retrieves the information from DBpedia, and the lack of quality that would be required for a well matched set of entities.
Thanks for the attention.
Alessandro Ghiotto