INM713 Semantic Web Technologies and Knowledge Graphs

Laboratory 6: Exposing Tabular Data as an RDF-based Knowledge Graph

Ernesto Jiménez-Ruiz

Academic course: 2020-2021
Updated: March 8, 2021

Contents

1 Git Repositories

Support codes for the laboratory sessions are available in *GitHub*. There are two repositories, one in Python and another in Java:

`https://github.com/city-knowledge-graphs`

2 Dataset

In this lab session we will use a dataset about world cities, more specifically we are using the free subset of the World Cities Database.[1] The dataset is also available in the GitHub repositories.

3 CSV to Knowledge Graph

3.1 Direct Transformation

There are several systems available that can automatically convert CSV files into RDF.

In python, for example:

- RDFLib built-in command `csv2rdf` (code `https://rdflib.readthedocs.io/en/stable/_modules/rdflib/tools/csv2rdf.html`)
- `https://github.com/SemanticComputing/CSV2RDF`

In Java, for example:

- `http://clarkparsia.github.io/csv2rdf/`
- `https://github.com/AtomGraph/CSV2RDF`

As we saw in the lecture notes, the direct transformation does not properly captures the semantics of the data.

Task 1 (Optional): Execute an available CSV to RDF converter over the world cities dataset and analyze the the obtained RDF triples.

3.2 Enhanced Transformation

The world cities dataset is simple but one could already think about an smart transformation to indicate that the elements in the first column are cities and that cities are located in a country.

Task 2: Create in Protégé a very simple ontology capturing the domain of the world cities dataset.

[1] `https://simplemaps.com/data/world-cities`

Task 3: Convert the CSV file intro triples (*i.e.*, into **4 ⋆ data**) using the vocabulary of the defined ontology (*e.g.*, `ex:City`). Create fresh entity URIs for the cities and countries (*e.g.*, `http://example.org/New_York`).

Task 4: Same as Task 3, but reusing entity URIs from DBPedia or Wikidata for cities and countries (*e.g.*, `http://dbpedia.org/resource/New_York_City`). This will get closer to **5 ⋆ data**. Use the respective KG look-up services. ***Tip:*** *If more than one candidate entity, you will need to decide if using the top-1 candidate or applying additional techniques (e.g., lexical similarity, contextual information) as we saw in the lecture notes.*

Task 5: Load the generated RDF graph in Task 4, and design and execute a SPARQL query that returns the countries and capital cities with a population $>$ 5,000,000. Create a CSV file from the results.

Task 6 (Optional): Give a look to the SemTab challenge datasets (`http://www.cs.ox.ac.uk/isg/challenges/sem-tab/`). The ground truths are also available so one can test the implemented solutions. The proceedings contain a link to the papers describing the participating systems.

4 Support Code

In the GitHub repositories there are several scripts to support you in this lab session and also in the coursework.

There are new dependencies. In Java, the `pom` file has been updated accordingly. For Python there are two new non built-in dependencies:

Pandas:

- Documentation: `https://pandas.pydata.org/pandas-docs/stable/`
- Installation: `https://pypi.org/project/pandas/`

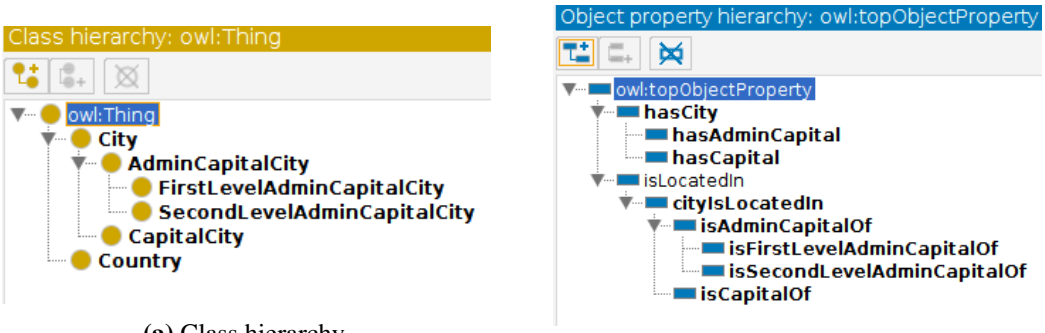
python-Levenshtein:

- Documentation: `https://github.com/ztane/python-Levenshtein/`
- Installation: `https://pypi.org/project/python-Levenshtein/`

4.1 Code Overview

Lookup. There are codes to connect to the look-up services of DBPedia, Wikidata and Google's KG. In python: `lookup.py`, in Java: `DBPediaLookup.java`, `WikidataLookup.java` and `GoogleKGLookup.java`.

Endpoints. DBPedia and Wikidata are open and they offer a SPAQRL Endpoint to access the KG. In python: `endpoints.py`, in Java: `DBPediaEndpoint.java` and `WikidataEndpoint.java`. These codes provide a number of potentially useful SPARQL queries to access relevant KG triples.



(a) Class hierarchy.

(b) Object property hierarchy.

Figure 1: Ontology for the world cities dataset.

String Similarity. String similarity will be useful to compare cell values to the label of KG candidates. In python (see `lexical_similarity.py`) we use the `python-Levenshtein` library and the ISub[2] method in `stringcmp.py`. In Java (see `LexicalSimilarity.java`) we use the Apache Commons Lexical Similarity library[3] and the ISub method (`I_Sub.java`).

CSV Management. There are different ways to open and mange CSV files. In Java I use `opencsv` and in Python I use both `pandas` and the built-in `csv` library.

5 Solutions

The idea is to make a transformation as complete as it is reasonably possible. A perfect transformation, however, is outside of the scope of this module. For this lab and coursework I am more interested in smart solutions and implementations than covering all possible cases in all rows. Furthermore, calling the look-up services may be expensive. If this is a limitation, a solution tested over a reasonable percentage of the original file will be of course accepted.

Task 2. I have created an ontology capturing the domain of the word cities dataset. The ontology `ontology_lab6.ttl` is available in the `lab6` folder in the Python repository or in the `files_lab6` in the Java repository.

The most important modelling choice is the definitions of different types of `City` (see Figure 1a) and the related object properties (see Figure 1b). I have also defined inverses and a hierarchy of object properties to enhance reasoning. The ontology also contains some example instances to test the inferences from Protégé (see Figure 2).

Task 3. `lab6_solution.py` and `Lab6_Solution.java` contain a model solution for the transformation. `worldcities-free-100-task3.ttl` contains the RDF graph with the transformed data. Key aspects:

[2]A String Metric for Ontology Alignment. ISWC 2005: `http://manolito.image.ece.ntua.gr/papers/378.pdf`

[3]`https://commons.apache.org/proper/commons-text/apidocs/org/apache/commons/text/similarity/`

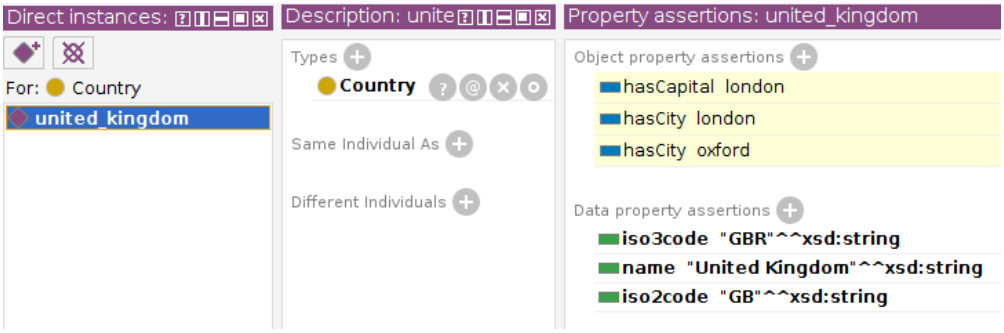


Figure 2: Example data and entailments.

- When we create triples we just refer to the ontology vocabulary via its URI. In a large (or very dynamic) ontology one may need to find a more automatic way to (re)use the ontology vocabulary. For example, via (fuzzy) matching in a similar way to the entity look-up in a large KG like DBPedia or Wikidata. Since we are dealing with very manageable ontologies, we can integrate their vocabulary within the code.

- The transformation to RDF has been modularized into small components. The transformation is tailored to the given data, but the individual mappings are generic and they could be reused for a different dataset. The mappings or transformation functions require as input: *(i)* one or more columns, and *(ii)* one or more ontology components (*e.g.*, concept or property). They provide as output a **template** to generate triples from the given input. The solution contains 4 mappings:

  - **mappingToCreateTypeTriple**: generic mapping to define the `rdf:type` of the elements of a column.
  - **mappingToCreateLiteralTriple**: generic mapping to create triples relating the elements of two input columns via a given data property.
  - **mappingToCreateObjectTriple**: generic mapping to create triples relating the elements of two input columns via a given object property.
  - **mappingToCreateCapitalTriple**: this mappings is more specific as it create different triples according to the value of the column *capital*.

Task 4. This solution builds on top of the implementation for Task 3; but, instead of creating fresh URIs for the countries and cities in the dataset, it tries to connect to the DBPedia look-up service to retrieve a KG entity URI. The proposed solution gets the top-5 entities from the look-up service and keeps the one with the highest lexical score with respect to the original cell value. As we saw in the lecture, this is not a perfect solution as in some cases one may need to use the contexts of the dataset and the KG to identify the right entity for a cell. `worldcities-free-100-task4.ttl` contains the RDF graph with the transformed data.

Task 5. The solution to this task will depend on the defined ontology vocabulary and the choices to transform to RDF. Before executing the SPARQL query one needs to

performed OWL reasoning with the uploaded data and the ontology created in Task 2. We will see more about reasoning during the Lecture 7.

The files `worldcities-free-100-task3-reasoning.ttl` and `worldcities-free-100-task4-reasoning.ttl` contain the extended RDF graphs after reasoning for Tasks 3 nd 4, respectively.

SPARQL queries:

```
SELECT DISTINCT ?country ?city ?pop WHERE {
     ?city rdf:type lab6:City ;
           lab6:isCapitalOf ?country ;
           lab6:population ?pop .
     FILTER(xsd:integer(?pop)>5000000)
}
ORDER BY DESC(?pop)
```

In my case the following alternative query would required to expand the graph via reasoning to return results.

```
SELECT DISTINCT ?country ?city ?pop WHERE {
     ?country rdf:type lab6:Country ;
           lab6:hasCapital ?city ;
     ?city lab6:population ?pop .
     FILTER(xsd:integer(?pop)>5000000)
}
ORDER BY DESC(?pop)
```

The created output CSV files with the results from the above query for Task 3 and Task 4 are `worldcities-free-100-task3-query-results.csv` and `worldcities-free-100-task4-query-results.csv`, respectively.