IN3067/INM713 Semantic Web Technologies and Knowledge Graphs

Laboratory 5: Exposing Tabular Data as an RDF-based Knowledge Graph

Ernesto Jiménez-Ruiz

Academic course: 2023-2024
Updated: March 5, 2024

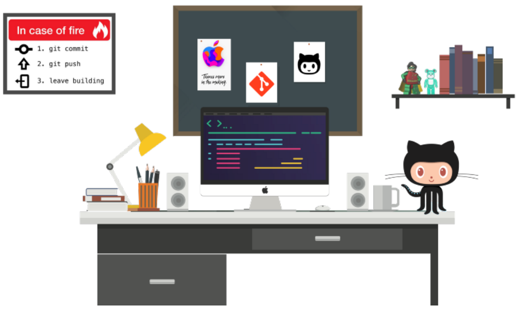
Contents

1 Git Repositories

Support **codes** and **datasets** 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`

For Java developers we use maven to deal with dependencies (see pom file). For Python developers there is always a `requirements.txt` within each lab folder. It is recommended to use environments, but it is not strictly necessary.



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 and moodle:

- Full dataset: `worldcities-free.csv`
- 100 rows only: `worldcities-free-100.csv`

3 CSV to Knowledge Graph

As we saw in the lecture notes, the direct CSV-to-RDF transformation does not properly captures the semantics of the data. The world cities dataset is simple but one could already think about a smart transformation to indicate that the elements in the first column are cities and that cities are located in a country.

I have created a simple ontology capturing the domain of the world cities dataset: `ontology_lab5.owl` and `ontology_lab5.ttl`. Note that the ontology is relatively simple and it does only contain a few instances as a proof of concept.

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

`lab5:london`	`rdf:type`	`dbo:City.`
`lab5:london`	`lab5:latitude`	`"51.5072"^^xsd:float.`
`lab5:london`	`dbo:populationTotal`	`"10979000"^^xsd:long.`

Table 1: Example triples.

3.1 Towards 4 $\star$ data

One of the steps to obtain 5-star and FAIR data, as we have seen in the lecture, is to:

- provide unique identifiers to the elements of the data;
- transform the data into RDF triples; and
- describe the data using an ontology.

Task 1: Convert (using Python or Java) the World Cities CSV file intro triples using the vocabulary of the provided ontology (*i.e.*, `ontology_lab5`) as in Table 1. Create fresh entity URIs for the cities and countries (*e.g.*, `lab5:london`, being `lab5:` the prefix defined for your lab 5 namespace).

Support codes: Check **Task 2.4** model solution in lab session 2. This solution follows a semi-automatic approach, that is, some "manual" assumptions have been made about column types and relationships among columns (*i.e.*, mapping/link to the ontology vocabulary), then the transformation to RDF triples has been made automatic.

3.2 Towards 5 $\star$ data

Linking your data to state-of-the-art KGs will increase its FAIRness as it will be more interoperable. These links are also a pre-requisite of 5 $\star$ data.

Task 2: Same as Task 1, but reusing entity URIs from DBpedia or Wikidata for cities and countries (*e.g.*, `dbr:London` instead of `lab5:london`). Use DBpedia's or Wikidata's KG look-up services to extract KG entity URIs. A look-up service will receive a string as input and retrieve a set of candidate KG entities. For example, the string "United Kingdom" will retrieve, among others, the entity `dbr:United_Kingdom` from DBpedia and the entity `wikidata:Q145` from Wikidata.
Tip: *If there are more than one candidate entity, you could either (1) get the top-1 candidate according to the look-up, or (2) apply additional techniques (e.g., lexical similarity, contextual information) as we saw in the lecture notes to get the best candidate.*

Support codes. I have added to the GiHub repositories the following support codes (including a Python notebook invoking the methods below):

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`.

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 (see `requirements.txt`) and the ISub[2] method in `isub.py`. In Java (see `LexicalSimilarity.java`) we use the Apache Commons Lexical Similarity library[3] and the ISub class (`I_Sub.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. For example, to query for the semantic types of `dbr:United_Kingdom` (*i.e.*, `dbo:Country`). The Endpoints will specially be useful if there is the need to apply disambiguation techniques to select the right look-up entity or to, for example, infer the semantic type of a column (*e.g.*, most of the cells are of type `dbo:City`).

3.3 Query your 5 $\star$ data

We have now transformed the World Cities dataset into RDF triples and we can perform SPARQL queries over the generated KG.

Task 3: Load the generated RDF graph in Task 2, 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.

4 Solutions

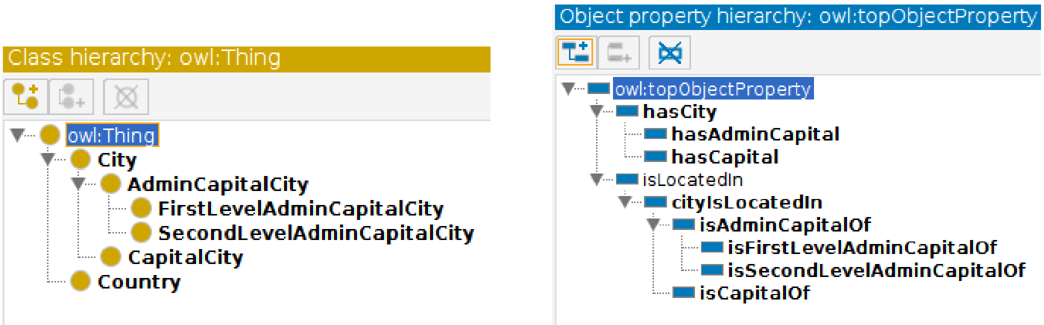
The idea is to make a transformation as complete as it is reasonably possible. A perfect transformation, however, is outside of the scope of this module. For this lab and coursework, I am more interested in smart solutions and implementations than covering all possible cases in all rows. Furthermore, 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.

About the ontology. 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 (more to come in Week 7). The ontology also contains some example instances to test the inferences from Protégé (see Figure 2).

Task 1. `lab5_solution.py` (and `lab5_solution_notebook.ipynb`) and `Lab5_Solution.java` in the respective GitHub repositories, contain a model solution for the transformation. The RDF graph with the transformed data is available in: `worldcities-free-100-task1.ttl`. 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/`



(a) Class hierarchy.

(b) Object property hierarchy.

Figure 1: Ontology for the world cities dataset.

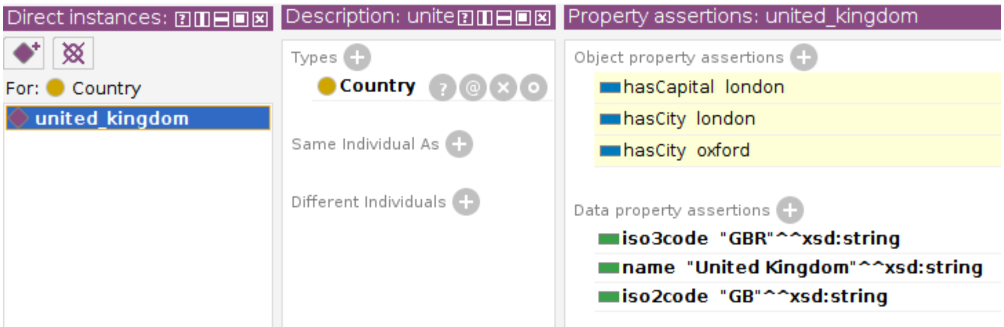


Figure 2: Example data and entailments.

- When we create triples we just refer to the ontology vocabulary via its URI (loading the ontology is not strictly necessary to create the triples). 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 a very manageable ontology in this lab, we can integrate their vocabulary within the code. For the coursework, it may be useful to create an **index** (*e.g.*, a **dictionary** in Python) that links occurrences in the csv file to URIs. For example, *e.g.*, "Mushroom" will imply that there is an ingredient or topping of type `ejr:Mushroom`.

- The transformation to RDF has been modularized into small components or **mappings**. 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). A mapping define 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 creates different triples according to the value of the column *capital*.

- In addition, the python model solution also provides a transformation with a unique mapping/function (see method `SimpleUniqueMapping` in `lab5_solution.py`), which is less modular. (Python only)

Task 2. This solution builds on top of the implementation for Task 1; 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-task2.ttl` contains the RDF graph with the transformed data.

Task 3. The solution to this task depends on the defined ontology vocabulary.

SPARQL query:

```
SELECT DISTINCT ?country ?city ?pop WHERE {
    ?city rdf:type lab5:City ;
          lab5:isCapitalOf ?country ;
          lab5: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 1 and Task 2 are `worldcities-free-100-task1-query-results.csv` and `worldcities-free-100-task2-query-results.csv`, respectively.