
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 3, 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.

**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.

[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/`