PhD course on Knowledge Graphs in the era of Large Language Models

Creation of a (simple) matching system

Ernesto Jiménez-Ruiz

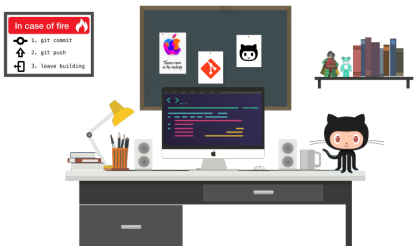
June 2024
Updated: June 4, 2024

Contents

1 Git Repository

Support codes for the laboratory sessions are available in *github*.

`https://github.com/city-knowledge-graphs/phd-course-uji`



2 Option A: A system for Tabular Data to KG Matching

The first option for the lab is the creation of a simple systems that matches tabular data to a knowledge graph. The system could become a participant for the annual SemTab challenge,[1] targeting the classical CEA (Cell-Entity annotation), CTA (Column-Type annotation) and CPA (Columns-Property annotation) tasks.

2.1 Dataset and Support Codes

We will use a dataset about world cities, more specifically we are using the free subset of the World Cities Database.[2] The dataset is also available in the GitHub repositories and moodle:

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

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: `lookup.py`.

String Similarity. String similarity will be useful to compare cell values to the label of KG candidates (see `lexical_similarity.py`). We will use the `python-Levenshtein` library and the ISub[3] method in `isub.py`

[1] `https://www.cs.ox.ac.uk/isg/challenges/sem-tab/`

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

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

Endpoints. DBPedia and Wikidata are open and they offer a SPAQRL Endpoint to access the KG. `endpoints.py`provides 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`).

2.2 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 A.1: Run the support codes to connect to the *lookup* and *SPARQL endpoint* services with different inputs.

Task A.2: Try to match cells in the columns about cities and countries to DBpedia or Wikidata (*e.g.*, matching "London" to `dbr: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.*

Task A.3: Extend your system to be able to automatically predict the type for the columns about cities and countries.

Task A.4: In the GitHub folder `sem-tab-evaluator`, I have uploaded a small portion of the Tough Tables (2T) dataset[4] (used in the SemTab challenge) that targets DBpedia. I have also uploaded the codes to calculate the accuracy in the CEA and CTA tasks (see `CEA_Evaluator.py` and `CTA_Evaluator.py`). Try to use the created system to annotate the provided tables.

Note 1: *You need to create an output similar to the examples given in the folder `system_example`, where each row identifies the source table, row id, column id (only in CEA), and the annotation URI.*

Note 2: *For clarity I have created one ground truth file per table, but in the SemTab challenge there is typically a single ground truth file for CEA and CTA, respectively, as each row already identifies the source table.*

[4]Tough Tables (2T): `https://zenodo.org/records/7419275`

3 Option B: An Ontology Alignment System

The second option for the lab is the creation of a simple ontology alignment system. This system could become a participant in tge annual Ontology Alignment Evaluation Initiative[5] (OAEI).

3.1 Datasets and Support Codes

The Ontology Alignment Evaluation Initiative (OAEI) is an international effort for the systematic evaluation of ontology alignment systems that has been running since 2004. The OAEI provides ontologies and reference alignment to drive a controlled evaluation among ontology alignment systems. In this lab session we are going to use two datasets from the OAEI.

Conference. This dataset aligns 7 ontologies from the same domain (conference organization): Cmt, ConfTool, Edas, Ekaw, Iasted, Sigkdd, Sofsem.

Anatomy. This evaluation dataset consists of finding an alignment between the Adult Mouse Anatomy (mouse) and a part of the NCI Thesaurus describing the human anatomy (human).

The GitHub repositories include support codes (including a jupyter notebook) for this lab session and the ontologies (owl format) and reference alignments (ttl format). The reference alignment files can also serve as an example about how your computed mapping should look like.

String Similarity. String similarity will be useful to compare cell values to the label of KG candidates (see `lexical_similarity.py`). We will use the `python-Levenshtein` library and the ISub[6] method in `isub.py`

Ontology access. We will rely on the Owlready API (`https://owlready2.readthedocs.io/`) to load and process the ontologies. The python script `AccessEntityLabels.py` loads an ontology, iterate over the classes and access the lexical information of the classes (*i.e.*, name/labels available as part of the URI or via the `rdfs:label` annotation). Lexical information is key for an alignment system.

Comparing with the reference alignment. The `CompareWithReference.py` script computes Precision and Recall given as input the system computed mappings and the reference mappings for the corresponding matching task.[7]. Note that not all matching tasks have a reference alignment, specially those coming from real world applications. Reference alignments are useful for evaluation campaigns like the OAEI.

[5] OAEI: `http://oaei.ontologymatching.org/`

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

[7] `https://en.wikipedia.org/wiki/Precision_and_recall`

3.2 Tasks

Task B.1 Basic ontology alignment can be reduced to comparing two list of elements.

1. Try to create a function that given two lists of elements, returns the list of elements in common (*i.e.*, equivalent). For example, given:
   `listA = ["pizza", "tomato sauce", "pepperoni", "restaurant"]`
   and
   `listB = ["pizza", "tomato", "peperone", "restaurant"]`
   return the `listC = ["pizza", "restaurant"]`.

2. Same as above but returning a list of "similar" elements. Recall the lexical similarity tools we used a few weeks back.

On the ontology setting you need to compare list of entities, and for each entity you know their URI and their label(s).

Task B.2 Design and implement a (simple) lexical matcher that loads two ontologies, and finds equivalence correspondences among their entities. Save the alignments as RDF triples in turtle format. For example:

```
cmt:Conference owl:equivalentClass confOf:Conference .
cmt:title owl:equivalentProperty confOf:hasTitle .
```

Tip 1: *align classes against classes and properties against properties. If there are instances, one should also target instances against instances using* `owl:sameAs` *(e.g.,* `city:ernesto owl:sameAs wd:ernesto`*.*
Tip 2: *Use lexical similarity and a threshold to decide which mappings are good enough to be added as output.*

Task B.3 Apply your algorithm over the following pairs of ontologies from the OAEI's conference track:

- cmt.owl - ekaw.owl
- cmt.owl - confOf.owl
- confOf.owl - ekaw.owl

Give suitable names to the mapping files (*e.g.*, `ernesto-cmt-ekaw.ttl`). Use the provided script and reference alignments to compute the precision and recall of the mappings computed by your system.

Task B.4 Apply your algorithm over the OAEI's anatomy track and get the precision and recall of your mappings. Note that, unlike in the conference track, the ontologies in the anatomy track are relatively large and contain $2,744$ classes (mouse.owl) and $3,304$ classes (human.owl). If your algorithm computes a pair-wise comparison, this will imply $> 9,000,000$ lexical comparisons.[8]

[8] LogMap (`https://github.com/ernestojimenezruiz/logmap-matcher`) is a state-of-the-art system that can match the ontologies in the anatomy track in 7s while producing competitive results. Its simple lexical variant LogMapLt takes only 2s. Results: `http://oaei.ontologymatching.org/2021/results/anatomy/`