Towards solving Part II of the Coursework

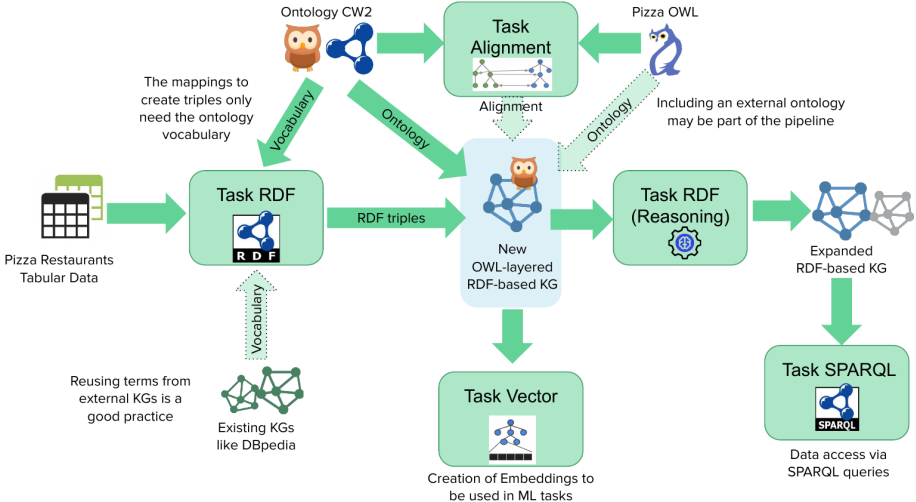
Ernesto Jiménez-Ruiz
Lecturer in Artificial Intelligence

CW Part 2

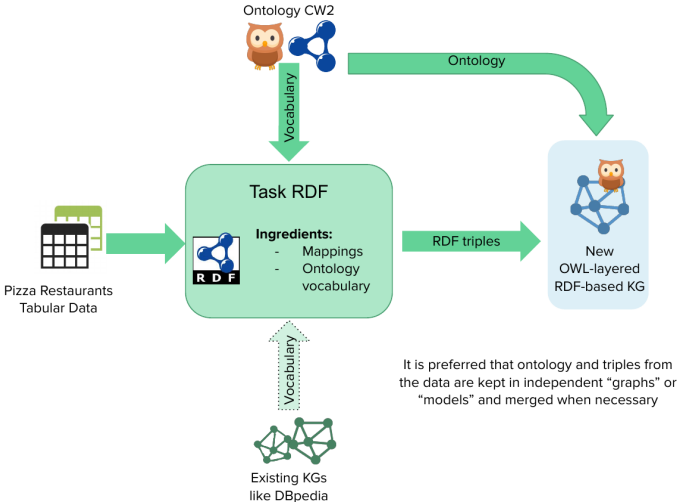
- Sunday, 12 May 2024, 5:00 PM
- Team registration March 17 (see form in moodle).

The global picture

The global picture



Global picture: Task RDF



Tips and tricks

Datasets and ontology: Part 2

– We are using the same dataset as Part 1.

✗ Do not use the ontology you created in Part 1

– The target ontology to be used is the one I have created as model solution. Statistics:

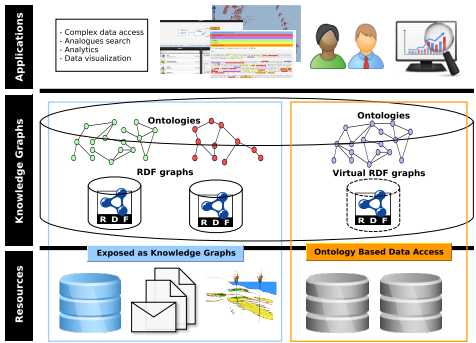
# Classes	151
# Object properties	18
# Data properties	6
# Logical axioms	255
# Annotation assertions	160

Exposing data as RDF: Ingredients

– **Ontology vocabulary**. Custom and/or given by a public KG.

– **Mappings**. Define a transformation from the tabular data to RDF data.

– **Ontology Axioms** (optional) - ♠



♠ Ernesto Jimenez-Ruiz and others. **BootOX: Practical Mapping of RDBs to OWL 2**. ISWC 2015

Common mistakes

✗ Yo DO NOT need to recreate the ontology axioms in python/java.

– *e.g.*, `cw:Restaurant rdfs:SubClassOf cw:Location` is already in the given ontology.

✗ No need to extend the ontology (its conceptual definitions).

Common mistakes

- ✗ Yo DO NOT need to recreate the ontology axioms in python/java.
  - *e.g.*, `cw:Restaurant rdfs:SubClassOf cw:Location` is already in the given ontology.
- ✗ No need to extend the ontology (its conceptual definitions).
- ✓ Part 2 is about creating RDF data (triples) from the CSV file, *e.g.*,:
  - ✓ `ejr:ernesto_pizzeria rdf:type cw:Restaurant .`
  - ✓ `ejr:ernesto_pizzeria cw:locatedInCity ejr:oxford .`

Common mistakes

✗ Yo DO NOT need to recreate the ontology axioms in python/java.

– *e.g.*, `cw:Restaurant rdfs:SubClassOf cw:Location` is already in the given ontology.

✗ No need to extend the ontology (its conceptual definitions).

✓ Part 2 is about creating RDF data (triples) from the CSV file, *e.g.*,:

✓ `ejr:ernesto_pizzeria rdf:type cw:Restaurant .`

✓ `ejr:ernesto_pizzeria cw:locatedInCity ejr:oxford .`

✓ Note that you have to reference the ontology vocabulary, that is, (re)use same URIs or compact URIs (case sensitive). *e.g.*, `cw:Restaurant`

`http://www.semanticweb.org/city/in3067-inm713/2024/restaurants#Restaurant`

Common mistakes (ii)

✗ Not using the same vocabulary will lead to the failure of answering the blind SPARQL queries during the marking process.

Example of blind query:

```
PREFIX rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX cw:  <http://www.semanticweb.org/city/in3067-inm713/2024/restaurants#>
SELECT DISTINCT ?rest WHERE {
    ?rest rdf:type cw:Restaurant .
}
```

Creating RDF data

– You can add the new RDF data to an empty `graph` (python) or `Model` (java).

Creating RDF data

- You can add the new RDF data to an empty `graph` (python) or `Model` (java).
- You can load the ontology into a `graph` (python) or `Model` (java) and add the new RDF data there.

Creating RDF data

- You can add the new RDF data to an empty `graph` (python) or `Model` (java).
- You can load the ontology into a `graph` (python) or `Model` (java) and add the new RDF data there.
- I prefer to keep graphs independently, *e.g.*, one for the ontology and another for the generated RDF data.

Creating RDF data

- You can add the new RDF data to an empty `graph` (python) or `Model` (java).
- You can load the ontology into a `graph` (python) or `Model` (java) and add the new RDF data there.
- I prefer to keep graphs independently, *e.g.*, one for the ontology and another for the generated RDF data.
- If necessary they can be merged later using python or java:
  - To load the ontology + RDF data in `Protégé` (otherwise `Protégé` gets confused).
  - To give OWL2Vec* in Task Vector a single file.

Example data in Ontology and tips

I have created a few triples as example together with the model ontology.

- **Ingredients:** for simplicity, the "same" ingredient in different restaurants is represented by a generic instance. *e.g.*,
  `cw:mozzarella rdf:type cw:Mozzarella .`
  (in the model ontology `cw:isIngredientof` is not functional.)

(*) Tip: use **unique** URIs for the data.

Example data in Ontology and tips

I have created a few triples as example together with the model ontology.

- **Ingredients:** for simplicity, the "same" ingredient in different restaurants is represented by a generic instance. *e.g.*,
  `cw:mozzarella rdf:type cw:Mozzarella .`
  (in the model ontology `cw:isIngredientof` is not functional.)

- **Pizzas**: Better to differentiate *same* named pizzas served in different restaurants. A margherita pizza served in restaurant A is a different from a margherita pizza in restaurant B.

(*) Tip: use **unique** URIs for the data.

Example data in Ontology and tips

I have created a few triples as example together with the model ontology.

- **Ingredients:** for simplicity, the "same" ingredient in different restaurants is represented by a generic instance. *e.g.*,
  `cw:mozzarella rdf:type cw:Mozzarella .`
  (in the model ontology `cw:isIngredientof` is not functional.)
- **Pizzas**: Better to differentiate *same* named pizzas served in different restaurants. A margherita pizza served in restaurant A is a different from a margherita pizza in restaurant B.
- **Cities**: same city name may appear in different states or countries.

(*) Tip: use **unique** URIs for the data.

Quality of the generated RDF data

- Main objective is to enhance data access via queries.
- Run simple queries over the RDF data (with the ontology and enabling reasoning).
- **Marking**: I will run a number of (simple) queries.

```
PREFIX rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX cw:  <http://www.semanticweb.org/city/in3067-inm713/2024/restaurants#>
SELECT DISTINCT ?rest WHERE {
    ?rest rdf:type cw:Restaurant .
}
```

How to create RDF data?

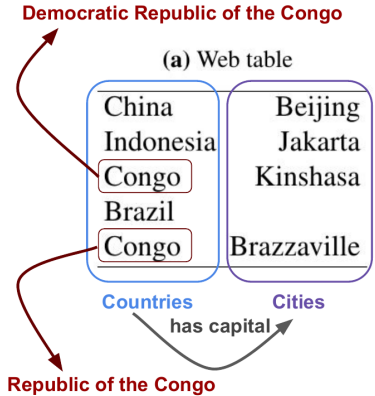
How to create RDF data? Check Lab 5 solution

– Column 1 is composed by Countries.

```
for value in data_frame[col1]:
    subject = ‘‘ex:’’ + value   #e.g., ex:China
    create_triple(subject rdf:type ex:Country)
```

– Column 2 entities are the capitals of column 1 entities.

```
for row in data_frame:
    subj = ‘‘ex:’’ + row[col1]   #e.g., ex:China
    obj = ‘‘ex:’’ + row[col2]   #e.g., ex:Beijing
    create_triple(subj ex:hasCapital obj)
```



Transformation to RDF: Creating mappings

- Modularized into small components or **mappings**.
- A mapping *maps* data to ontology elements.
- Mappings require as input: *(i)* one or more columns, and *(ii)* one or more ontology components (*e.g.*, concept or property).
- Mappings define a **template** to generate triples from the given input.

Transformation to RDF: mappings in lab 5 solution (i)

- **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*.

Transformation to RDF: mappings in lab 5 solution (ii)

- **mappingToCreateTypeTriple**: *e.g.*,
  `lab5:ahmadabad rdf:type lab5:City`
- **mappingToCreateLiteralTriple**: *e.g.*,
  `lab5:ahmadabad lab5:latitude "23.03"^^xsd:float`
- **mappingToCreateObjectTriple**: *e.g.*,
  `lab5:baoding lab5:cityIsLocatedIn lab5:china`
- **mappingToCreateCapitalTriple**: *e.g.*,
  `lab5:bangkok lab5:isCapitalOf lab5:thailand`

Creating new (template) URIs

- Create new URIs for the data (*e.g.*, restaurants, pizzas)
- The URI template creation may involve input from one or several cells
- *e.g.*, `ejr:ernesto_pizzeria_in_oxford`
- Alternatively, instead of creating fresh URIs (*e.g.*, `lab5:ahmadabad`) we can reuse them from DBpedia (*e.g.*, `dbr:Ahmadabad`).

Retrieving/reusing ontology URIs (i)

- This is part of the creation of the mappings and may involve some **manual injection of our table understanding** (*e.g.*, column 1 is of type `lab5:City`).
- Since lab 5 ontology is very small, the solution encodes its vocabulary.

Retrieving/reusing ontology URIs (i)

- This is part of the creation of the mappings and may involve some **manual injection of our table understanding** (*e.g.*, column 1 is of type `lab5:City`).
- Since lab 5 ontology is very small, the solution encodes its vocabulary.
- Larger (possibly dynamic) ontologies may require a different approach. Recall the *Semtab* challenge.

Retrieving/reusing ontology URIs (ii)

- Fuzzy matching, as the look-up service we use for DBpedia, may be a good compromise.
- For the coursework solution, one could automatically load and process the ontology to create an index of vocabulary (*e.g.*, a dictionary in python).
- For example, *e.g.*, “mushroom” will imply that there is an ingredient or topping of type `cw:Mushroom`.

Lab 5 solution

Task

– Understand the solution.

– Following exactly the same approach will lead to a pass mark in coursework part 2.