Assignment 4: Knowledge Graph Quality

Task 1 (50%)

We want to assess the understandability of the DBPedia schema and one factor that plays a (negative) role is ambiguity. For that, you are asked to develop a term ambiguity detector and use it to identify ambiguous class and property names in the DBPedia schema. You can use an LLM for that purpose but, along with that, also use a dictionary-based approach (e.g. use Wordnet or Wiktionary). Comment on the effectiveness and efficiency of the dictionary-based approach vs the LLM approach.

Solution

With the following SPARQL we are able to retrieve all different entities, based on which afterwards, calculate the ambiguity scores.

```
SELECT DISTINCT ?entity_uri ?label ?entity_type ?abstract

  WHERE {

     {

        ?entity_uri rdf:type owl:Class ;

                    rdfs:label ?label .

        OPTIONAL { ?entity_uri dbo:abstract ?abstract }

        FILTER(LANG(?label) = "en")

        FILTER(LANG(?abstract) = "en" || !BOUND(?abstract))

        BIND("Class" AS ?entity_type)

     }

     UNION

     {

        ?entity_uri rdf:type rdf:Property ;

                    rdfs:label ?label .

        OPTIONAL { ?entity_uri dbo:abstract ?abstract }

        FILTER(LANG(?label) = "en")
        FILTER(LANG(?abstract) = "en" || !BOUND(?abstract))
        BIND("Property" AS ?entity_type)
    }
    UNION
    {
        ?entity_uri rdf:type ?someType ;
                    rdfs:label ?label .
        OPTIONAL { ?entity_uri dbo:abstract ?abstract }
        FILTER(LANG(?label) = "en")
        FILTER(LANG(?abstract) = "en" || !BOUND(?abstract))
        FILTER(?someType != owl:Class && ?someType != rdf:Property)
        BIND("Entity" AS ?entity_type)
    }
}
ORDER BY ?entity_type ?label
LIMIT 10000
"""
```

Evaluation of the generated queries

Ambiguity Analysis Summary:

Maximum number of synsets found: 9

Number of terms analyzed: 3

Sample of normalized scores:

Word: label

Normalized ambiguity score: 1.0

Raw synset count: 9

Word: type

Normalized ambiguity score: 0.889

Raw synset count: 8

Task 2 (50%)

As we discussed in class, abstract concepts are generally more difficult to accurately model than concrete ones. To see how well DBPedia does this kind of modeling, perform the following experiment:

- Consider the ESCO skill entities available in esco_skills_en.csv and find their equivalent entities (if any) in the DBPedia graph, as well as these entities' classes
- Use an LLM to judge if the entity-class pairs are accurate and describe your findings,

including which DBPedia classes tend to have the most mistakes.

Failed experiment

We first try to download all distinct entities but the query does not return the results we want. We finally are able to via the following code, which we run in batches to improve the overall speed of execution. One thing to note is that we limit down to 100 results per entity.

```
import pandas as pd
from SPARQLWrapper import SPARQLWrapper, JSON
from tqdm import tqdm  # Useful for displaying a progress bar

# DBpedia SPARQL endpoint
sparql = SPARQLWrapper("http://dbpedia.org/sparql")
sparql.setReturnFormat(JSON)  # Set return format once

# Optimized function to perform a SPARQL query for a list of skill
labels
def query_dbpedia(skills):
    # Prepare SPARQL query with placeholders for skills
    skill_filters = " || ".join(
        [f'CONTAINS(LCASE(?entityLabel), "{skill.lower()}")' for skill
in skills]
    )

    sparql_query = f"""
    PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
    PREFIX dbo: <http://dbpedia.org/ontology/>
    PREFIX dbp: <http://dbpedia.org/property/>

    SELECT ?entity ?entityLabel ?class ?classLabel
    WHERE {{
      ?entity rdfs:label ?entityLabel .
      ?entity a ?class .

      # Filter for entities matching any of the skills in English
      FILTER (LANG(?entityLabel) = "en" && ({skill_filters}))

      # Optional: Retrieve the class label if available
      OPTIONAL {{
        ?class rdfs:label ?classLabel .
        FILTER (LANG(?classLabel) = "en")
      }}
    }}
    LIMIT 100
    """

    # Execute the query and parse results
    sparql.setQuery(sparql_query)
    try:
        results = sparql.query().convert()
        output = [
            {
                "entity": result.get("entity", {}).get("value", ""),
                "entityLabel": result.get("entityLabel",
{}).get("value", ""),
                "class": result.get("class", {}).get("value", ""),
                "classLabel": result.get("classLabel",
{}).get("value", ""),
            }
            for result in results["results"]["bindings"]
        ]
    except Exception as e:
        print(f"Error querying DBpedia:", e)
        output = []

    return output

# Batch skills into chunks to reduce the number of queries
```

```
batch_size = 50  # Adjust as needed based on performance or endpoint
limits
all_results = []

esco_skills_short = esco_skills_short
for i in tqdm(range(0, len(esco_skills_short["preferredLabel"]),
batch_size)):
   skill_batch = esco_skills_short["preferredLabel"][i : i +
batch_size]
   skill_results = query_dbpedia(skill_batch)
   for result in skill_results:
       result["preferredLabel"] = next(
           (skill for skill in skill_batch if skill.lower() in
result["entityLabel"].lower()), ""
       )
       all_results.append(result)

# Convert results to a DataFrame
results_df = pd.DataFrame(all_results)

# Display the first few rows of the results
print(results_df.head())
```

Using the LLM as Judge

We the following prompt, we attempt to get a classification regarding each entity-class pair.

```
input_prompt = """
You are tasked with evaluating the accuracy of entity-class matches
from DBpedia, derived from Wikipedia and returning them in VALID JSON
strings. For each entity, assess the matched DBpedia classes and
assign a score:

Instructions: For each entity-class pair, assign a score from 1 to 5
based on the match accuracy:

5: Exact match
4: Strong match
3: Moderate match
2: Weak match
```

```
1: Incorrect match
After scoring, calculate the average score for each DBpedia class and
identify "error-prone" classes with an average below 3. Provide a
brief description of misclassification patterns.

Example Input:

Entity: "New York City" -> Matched Classes: City, Place,
AdministrativeRegion, Settlement
Output: {{"New York City": [5, 4, 5, 5] }}

Entity: "Mars Rover" → Classes: Spacecraft, Vehicle, Machine
Output: {{"Mars Rover": [5, 4, 2] }}


Assign a score for the following:
Entity: "{preferred_label}" -> Matched Classes: {matched_labels}


# Rules
- Do not write anything before or after the JSON. Directly output the
JSON object.
- Do not add any explanation or comments.
- Return only valid JSON objects


JSON:{{"""
```

Final results

```
{"criminology": [5]}
{"Haskell": [5]}
{"cardiovascular system": [5]}
{"Erlang": [1, 1]}
{"preventive medicine": [5, 4, 2, 1]}
{"legal studies": [1, 5]}
{"mergers and acquisitions": [1, 1, 1]}
{"environmental policy": [5]}
{"osteology": [5]}
{"literature": [5, 1]}
{"Lisp": [1, 1]}
{"Christianity": [5, 1, 1, 1, 1, 1, 1, 1, 1, 1]}
```

```
{"Sass": [1, 5, 1, 1]}
{"communication": [1, 1, 1, 1, 1, 1, 1]}
{"Spanish": [1, 1, 1, 1, 1]}
{"fine arts": [1, 2, 5, 3]}
{"audiology": [5]}
{"energy": [1, 1, 1, 1, 1, 1]}
{"astrology": [5, 1]}
{"Malay": [1, 1, 1, 1, 1, 1, 1, 1, 5, 1]}
{"Montenegrin": [3, 1, 1]}
{"sexology": [5]}
{"LESS": [5, 1, 1, 1, 1, 2, 3, 1]}
{"logic": [1, 1, 1, 1, 1, 1, 1]}
{"history": [1, 2]}
{"cartography": [5, 1]}
{"radiology": [1, 5, 5, 5]}
{"political science": [1, 5]}
{"journalism": [5, 1, 2, 3, 4, 1, 2, 3, 4]}
{"Arabic": [1, 5, 1, 1]}
{"Ukrainian": [1, 2, 3]}
{"pharmacy law": [5]}
{"business law": [5]}
{"Entity": "Japanese", "Matched Classes": [1, 1, 1]}
{"cameras": [1]}
{"photography": [5]}
{"politics": [1, 2, 3, 4, 5, 5, 5, 5, 5, 5]}
{"yoga": [2, 1, 1, 1]}
{"viticulture": [5, 1, 1]}
{"herpetology": [5]}
{"law enforcement": [5]}
{"Perl": [1, 1, 1]}
{"PHP": [1]}
{"mathematics": [1, 1, 1]}
{"database": [5, 1]}
{"German": [1, 1]}
{"Entity": "football", "Matched Classes": [1, 1, 1]}
{"social sciences": [1, 2, 5]}
{"Entity": "Romanian", "Matched Classes": [1, 2, 3, 4, 5, 1, 1, 1, 1,
1]}
{"comparative literature": [5, 4]}
{"orthodontics": [5]}
{"textile industry": [5, 2, 1]}
{"landscape architecture": [5]}
{"consultation": [5]}
{"R": [1, 1, 1, 1, 1, 1, 1, 1, 1, 1]}
{"public relations": [5]}
{"algebra": [5]}
{"financial management": [5]}
{"genetics": [1, 2, 3, 5]}
{"plastic surgery": [5]}
{"archaeology": [5]}
{"APL": [1, 5, 1, 1, 1, 5, 1, 1, 1]}
{"autism": [1]}
{"aerospace engineering": [5]}
{"Latvian": [5, 1, 1, 1, 1, 1, 1, 1, 1, 1]}
{"pregnancy": [1, 2, 3]}
{"Azerbaijani": [1, 1, 1, 1, 1, 1, 1, 1, 5, 1]}
{"neurology": [5]}
```

```
{"government policy": [1]}
{"mining engineering": [5]}
{"adult education": [5]}
{"pharmacology": [1, 5, 2]}
{"psychopharmacology": [1, 5]}
{"biotechnology": [5, 1, 2, 4, 3, 2]}
{"natural gas": [1, 1, 1, 5, 1, 1, 1, 1, 1]}
{"Telugu": [1, 1, 3, 1, 1, 5, 1, 4, 5, 2]}
{"family therapy": [5]}
{"virology": [2, 1]}
{"Galician": [5, 1, 1]}
{"entrepreneurship": [5]}
{"Russian": [5]}
{"STAF": [1, 1, 1, 1, 1]}
{"English": [1, 5, 1, 4, 2]}
{"Cisco": [1, 1, 1, 1, 1, 1]}
{"wildlife": [5, 4, 3, 2, 1, 1]}
{"oxidation": [1, 2]}
{"computer science": [1, 1, 1]}
{"stem cells": [5]}
{"morality": [1, 1]}
{"consumer protection": [5]}
{"Javanese": [5]}
{"tropical medicine": [5]}
{"cosmetics": [5]}
{"statistics": [1, 1]}
{"Swedish": [1, 1, 1, 1, 1, 1, 1, 5]}
{"graphic design": [1, 2]}
{"linguistics": [5]}
{"civil engineering": [5, 2, 2, 5, 2, 2, 5, 2, 5]}
{"thermodynamics": [5, 2]}
{"pharmacognosy": [5]}
{"Entity": "Norwegian", "Matched Classes": [1, 5, 3, 2, 4, 1, 2, 1,
2, 1]}
{"geophysics": [5]}
{"meteorology": [5]}
{"dies": [1, 1, 1, 1]}
{"economics": [5]}
{"rehabilitation": [5, 4, 1]}
{"Welsh": [1, 1, 1]}
{"Romani": [1, 5]}
{"resuscitation": [5]}
{"Islam": [1, 1, 1, 1, 1, 1, 1, 1, 1, 1]}
{"electronics": [1]}
{"Latin": [1, 5, 1]}
{"public health": [1, 1]}
{"Slovenian": [5, 1, 1, 1, 1, 1]}
{"Gujarati": [1, 1, 1]}
{"mechanical engineering": [1, 2]}
{"id Tech": [5]}
{"tennis": [5, 1, 1, 5, 5]}
{"Basque": [5]}
{"babysitting": [1]}
{"thoracic surgery": [5, 4]}
{"international law": [1, 1, 1]}
{"ski": [1, 5, 1, 1, 1, 3, 1]}
{"petroleum": [5, 1, 2, 5, 1, 4, 5, 1, 3, 5]}
```

```
{"physical medicine": [1, 1]}
{"SPARK": [5, 1, 2, 3]}
{"migration": [1, 2, 5, 3]}
{"transplantation": [1, 5]}
{"snowboard": [1, 1]}
{"property law": [5]}
{"toxicology": [5, 1, 2, 3, 4, 1, 5, 4]}
{"terminology": [5, 2, 3, 1, 4]}
{"stock market": [2, 1]}
{"hepatology": [5]}
{"probability theory": [1]}
{"theology": [5, 1, 1, 1, 1]}
{"spelling": [5]}
{"sporting events": [2, 2, 2, 2, 2, 2, 2, 2, 2, 2]}
{"Urdu": [1, 2]}
{"Vietnamese": [1]}
{"psychology": [1, 5, 5, 5, 5]}
{"anthropology": [1]}
{"Serbian": [5, 1, 1, 1, 1, 1, 1, 1, 3, 2]}
{"psychoacoustics": [5]}
{"defense system": [5]}
{"hair": [1, 1]}
{"electricity": [5]}
{"Dutch": [1, 1]}
{"iOS": [1, 1, 1, 1, 1]}
{"grammar": [1, 1, 1, 1, 1, 1]}
{"medical genetics": [5, 1, 2]}
{"Bulgarian": [1, 1, 1, 1, 1, 1, 5, 1]}
{"cultural history": [2, 3, 1]}
{"sex education": [2, 1]}
{"Yiddish": [1, 1]}
{"beeswax": [5, 1]}
{"Walloon": [1, 5, 1]}
{"Hindi": [1]}
{"pathology": [5, 4, 2, 1, 3, 2]}
{"Czech": [5, 1, 5, 5, 5]}
{"Hungarian": [1, 1, 1, 1, 1, 1, 1, 1, 1, 1]}
{"endocrinology": [5]}
{"Punjabi": [5, 2, 1]}
{"power engineering": [5]}
{"judaism": [1, 1, 1, 1, 1, 1, 1, 1, 1, 5]}
{"sign language": [5, 2, 1, 1]}
{"semiconductors": [5]}
{"human resource management": [5]}
{"respiratory therapy": [5]}
{"social justice": [5]}
{"French": [1, 1, 2, 5]}
{"natural history": [1, 1, 1, 1]}
{"ABAP": [1, 5, 1, 1, 1]}
{"cinematography": [5]}
{"boxing": [1, 1, 1]}
{"Bengali": [1, 2, 5, 3]}
{"protein": [5]}
{"mechanics": [5, 1, 1, 2]}
{"set theory": [5]}
{"alkylation": [5]}
{"Albanian": [5, 1, 1, 1, 1, 1, 1, 1, 1, 1]}
```

```
{"torah": [5]}
{"child protection": [5]}
{"Polish": [1, 1, 1]}
{"osteopathy": [5]}
{"Kazakh": [1]}
{"prayer": [5]}
{"mass spectrometry": [5]}
{"Korean": [1, 1, 1]}
{"psychiatry": [5, 1, 2, 3, 4, 1, 2]}
{"emergency medicine": [5]}
{"Capture One": [5]}
{"cybernetics": [5]}
{"Armenian": [5, 1, 1, 1, 1, 1, 1, 1, 1, 1]}
{"astronomy": [1, 1, 1, 1, 1, 1]}
{"lasers": [5]}
{"Scala": [1, 1, 5, 5]}
```