DSCI 558: Building Knowledge Graphs

Homework 3: Entity Resolution, Blocking & Knowledge Representation

Released: Sep 12th, 2020

Due: Sep 23rd, 2020 @ 23:59

Ground Rules

This homework must be done individually. You can ask others for help with the tools, however, the submitted homework has to be your own work.

Summary

In this homework, you will link movies from the Internet Movie Database (IMDb) to the American Film Institute (AFI), implement and test two blocking techniques, and then represent some of your data using RDF. The entity resolution and blocking tasks will be done using The Record Linkage ToolKit (RLTK), an open-source record linkage platform. You will use RDFLib, a Python library for working with RDF, for the task of knowledge representation. We provide a python notebook (`ER_KR.ipynb`), which contains instructions, code and descriptions on how to use the tools we mentioned.

Task 1: ER (4 points)

In this task, you are given a dataset of movies from IMDb (`imdb.jl`) and a dataset of movies from AFI (`afi.jl`). Your goal is to match records from these 2 datasets using record linkage methods. This means you need to figure out which pairs of movies in the two datasets are referring to the same movie.

IMDb and AFI datasets contain several attributes, some are unique for each dataset, some are present in both. For the task of linking, we are interested in the 3 fields that are present in both datasets (**movie title/name**, **release date/year** and **genre**).

We provide the template file `hw03_tasks_1_2.py` which includes some of the code you see in the given notebook, you may use the notebook to develop and test your code, but the final code for tasks 1.1 and 1.2 (and also task 2) should be implemented in this file.

Task 1.1 (2 points)

For **each attribute** (**total of 3**): (1) Analyze the given data and choose string similarities that you think are appropriate. (2) Explain your choices in the report. (3) Implement a method that computes the field similarity of this field between the records for the 2 datasets (look for `` `# ** STUDENT CODE. Task 1.1` `` in the submission file)

Notes:

- You may customize string similarity methods or change/clean attribute values if necessary. For example, you can choose the Levenshtein similarity method for the movie names, which you derive from the original attribute values.

Task 1.2 (2 points)

Design a scoring function to combine your field similarities. Explain your choices of weights in the scoring function in the report. Implement a method that predicts the corresponding AFI movies for the IMDb movies (in `imdb.jl`) using your scoring function (complete the code in `main()`). Set the value to `null` if there is no corresponding entry in the AFI dataset. Export your prediction to an output file (`Firstname_Lastname_hw03_imdb_afi_e1.json`) with the format:

```
[ { "imdb_movie": <imdb_url>, "afi_movie": <afi_url>}, ... ]
```

For example:

```
[
  {
    "imdb_movie": "https://www.imdb.com/title/tt0033467/",
    "afi_movie":  "https://catalog.afi.com/#dc440a1a7fa4a6bd30f183eded493ef2"
  },
  {
    "imdb_movie": "https://www.imdb.com/title/tt0108052/",
    "afi_movie":  "https://catalog.afi.com/#642a1d0b14872b56d8fde9228170da6f"
  }
]
```

Notes:

- Look for `# ** STUDENT CODE. Task 1.2` in the submission file
- The attached notebook offers an RLTK-built-in method for running evaluation. You can use the provided code to test your performance over the provided development set (`imdb_afi_el.dev.json`) before finalizing your implementation in the file.

Task 2: Blocking (3 points)

In this task, you will use RLTK to implement two blocking techniques and evaluate their effectiveness. Your code for this task will be implemented in the same file from the previous task (`hw03_tasks_1_2.py`).

Task 2.1 (1 pts)

Complete the missing code for this task (Look for `# ** STUDENT CODE. Task 2.1` in the submission file).

Task 2.2 (1 pt)

Implement hash-based and token-based blocking (Look for `# ** STUDENT CODE. Task 2.2` in the submission file).

Task 2.3 (1 pts)

To evaluate the performance of the blocking techniques you implemented we calculate the reduction ratio and pairs completeness.

2.3.1 Run the script `hw03_eval_blocking.py` with a '`hash`' argument (i.e., `python hw03_eval_blocking.py hash`).
Include the numbers you get in your final report.

2.3.2 Run the script with a '`token`' argument (i.e., `python hw03_eval_blocking.py token`).
Include the numbers you get in your final report.

2.3.3 Explain the difference in the results you get between the two blocking techniques (max 2 sentences, in your final report)

Task 3: KR (3 points)

In this task, you will represent the movie data (after linking) using RDF. The ontology (vocabulary/schema) you will use is **schema.org**. As this ontology may not include all necessary classes and properties to model your data, you will need to extend the ontology with classes that you define on your own.

Task 3.1 (1 point)

Describe the model (in the report) you will use to generate the RDF data to describe the merged movie entry with all of the available attributes from the two sources you have matched.
There's a total of 13 attributes: title, release-date, certificate, runtime, genre, imdb-rating, imdb-metascore, imdb-votes, gross-income, producer, writer, cinematographer and production-company.
Use the appropriate classes and properties from **schema.org**. Define your own if you could not locate a suitable one in **schema.org**. Finalize the file describing your model (`model.ttl`) with the missing attributes and rename it to `Firstname_Lastname_hw03_model.ttl`.

Notes:

- As a starting point, you may want to use the class `` `https://schema.org/Movie` `` to represent a movie and the property `` `https://schema.org/datePublished` `` to represent a predicate that describes that movie's release time (as seen in `model.ttl`).
- The attribute 'production company' should not be referred to as a plain literal from the movie entry. Instead, create a local URI for each production company (as depicted in `model.ttl`, the movie's production company is an instance of a class, not a literal).

Task 3.2 (1 point)

Implement a python program that uses the data from the 2 datasets (from task 1) and your results file (`Firstname_Lastname_hw03_imdb_afi_el.json`). The program should convert the combined movie data to RDF triples (in turtle format, `ttl`) using the model you defined in task 3.1, the generated file should be named `Firstname_Lastname_hw03_movie_triples.ttl`.

Notes:

- Use the IMDb URI as the identifier of the node (subject). You can discard the AFI URI
- See the attached notebook for an example of how to create and generate RDF graph (triples) in `ttl` format (syntax).

Task 3.3 (1 point)

Choose two movie instances and a single production company instance (locate them in your `ttl` file, the two movies should refer to the same production company) and visualize the triple data in your report. Use the online tool at http://www.ldf.fi/service/rdf-grapher to visualize the triples in a graph. The result should look like what is shown in Figure 1 (the figure shows partial data, yours should be complete).

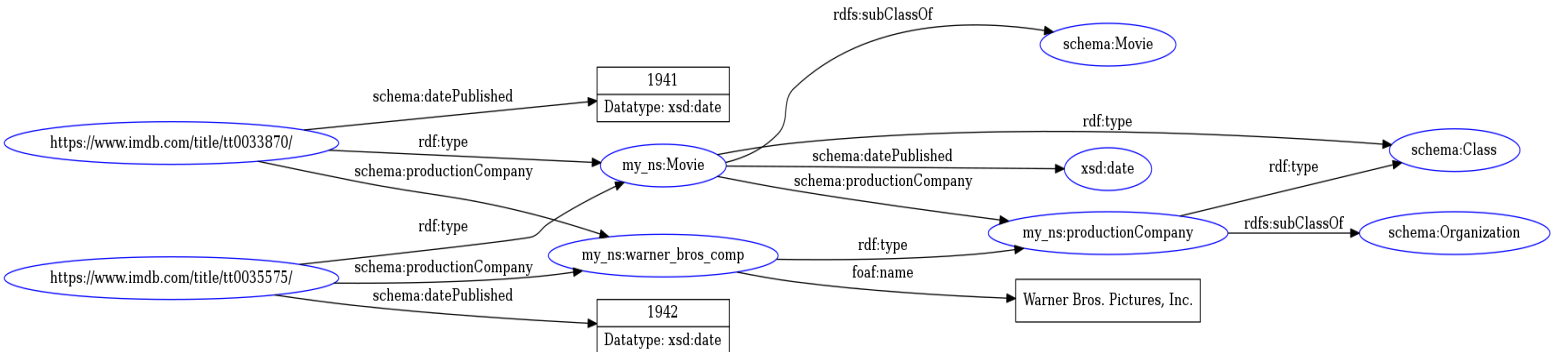

Figure 1: An example of a graph visualization

Submission Instructions

You must submit (via Blackboard) the following files/folders in a single `.zip` archive named `Firstname_Lastname_hw03.zip`:

- `Firstname_Lastname_hw03_report.pdf`: pdf file with your answers to Tasks 1, 2 & 3
- `hw03_tasks_1_2.py`: as described in Tasks 1+2
- `Firstname_Lastname_hw03_imdb_afi_el.json`: as described in Task 1.2
- `Firstname_Lastname_hw03_model.ttl`: as described in Task 3.1
- `Firstname_Lastname_hw03_movie_triples.ttl`: as described in Task 3.2
- `source`: This folder includes all the additional code you wrote to accomplish the tasks