INFO216

News Graph Aggregator

Candidates: 186, 197, 191

UiB

Word count: 1434

News graph aggregator for catching similar news stories and events

System and process description

For our project we used python for programming rdflib and sparqlwrapper and blazegraph to input test datasets and test queries on the SPARQL-engine. We used GitHub as a tool for version-control and sharing code and ideas among the group members. Our project includes two python-programs (item_aggregator and uuid_generator). The uuid_generator has only one purpose, and that is to generate a hash_value that we use in the other python program to assign IDs to our newshunter events. We also have three different datasets where the difference between them is the size of the datasets, the structure of the data is consistent and same across all sets.

In the item_aggregator program we have three functions which do all the processing and managing of the input data. These are the following:

item_lifter: takes in an item from the graph_constructor function and applies the sparql-query on that item, it then returns the result as JSON and converts the output. This will return the nhterm:sourceDateTime, originalText, sourceIRL, hasContributor and inCollection for a given item.

make_entity_dict*:* Takes in the result from the sparql-query *entity_graph* and processes each item to extract pairs of items, anchors and annotators as well as the entity related to that item. Does some checks on the data structures to determine if they are to be initialised, does some checks to see if the structures are to be added to or if the particular item, anchor or annotator already exist in a structure. Returns three dictionaries containing the entity, item/anchor/annotator pairs.

graph_constructor*:* processes through the values we collect from make_entity_dict and item_lifter to then append it into lists. It also constructs our graph and adds the values from the lists into the graph before returning the graph we constructed.

Purpose of system

We want this to be a tool for journalists and other people working in the news-sphere. This tool makes it possible to for example locate news articles and tweets about a certain topic and make semantically rich knowledge graphs with the information from these sources using the nhterm:item property.

The advantages of using semantics in this context is that news-companies will be able to take greater advantage of the data. Computers can infer context and add additional information based on the existing ontologies, and humans can have a better understanding of what the graph contains.

Now, you can use the graph to get the articles that match on topics. At the same time, we have now made it possible to draw a connection between articles, so that you do not have to manually go into each article and do this process. In other words, the computer significantly simplifies this process and the program can be run on fairly large datasets which makes the task feasible for these types of datasets.

Data sets description and sources

In our project we are using the premade datasets given to us through the task description. There are three datasets in total and the main difference between them is the size of the datasets. Each dataset consists of multiple news graphs/items and is written in turtle format. The format of the datasets are the same across the different sets with each graph having a distinct ID and each graph being defined as a *RDF.type nhterm.Item*.

All of the graphs use the nh vocabulary to define the different entities and attributes and properties. All of the graphs have an entity *nh: hasAnnotation* which consist of a blank node containing properties of the item, like *nhterm:anchorOf*. A graph item can have many annotations or just one. Unlike the rest of the graph, this node also contains *nhterm:hasAnnotator* which points to a tool for annotating, like Dbpedia Spotlight.

The graphs use *nh:nhterm* to define the properties of the item in the graph, all of the graphs have the following properties

nhterm:

- *hasAnnotation*
    - a blank node containing a with label: *RDF.type nhterm:Annotation*.
    - Contains: nhterm.anchorOf, nhterm:hasAnnotator and nhterm:hasEntity
- *anchorOf*
    - some xsd:string containing a keyword related to the news item
- *hasEntity*
    - The dbpedia resource for the news item
- *hasContributor*
    - Points to the URI relating to who contributed the text, could be a news article or a tweet from a twitter user.

- *inCollection*
    - Points to the wikidata resource relating to the media used in contributing the text. (Fo example, twitter)
- *originalText*
    - Some xsd:string containing the text for the news item.
- *sourceDateTime*
    - A xsd:DateTime object including the date and time in a ISO 8601 compliant format, example: YYYY-MM-DDThh:mm:ss.sTZD (eg 2020-09-29T01:04:55+00:00)
- *sourceIRL*
    - The IRI/URI of the source from where the text is lifted, could be a tweet or a news-article. The object is treated as a xsd:anyURI type.

Technologies, tools, standards, vocabularies

We were given three different datasets that consisted of large amounts of news articles. The reason we chose Python is because this is a language we all have experience with, and therefore it becomes natural to choose RDFlib and SparQLWrapper since processing this is supported in Python.

It was stated that we could use Java, but since we have most experience/knowledge with Python, we envisioned that we could get the best possible solution to the problem using Python.

We have chosen to use the vocabulary from the news hunter, which contains nh and nhterm. Nh is specifying the event, while nhterm is more specific details of the event where the different properties are contained. We also use xsd to define types. This means we use LOD for xsd. News Hunter is not by definition a LOD, but we gained access to this as a Linked Data.

Use cases

- Journalists/users find easier which news articles other media write about, which may be relevant for them to cover in their own medium.
- The program can find correlation between articles you wouldn't otherwise think had a relation. For example between articles concerning the same topic, but not necessarily published at the same time.
- The program assumes that you have a dataset that consists of news items in turtle format, so that you get the desired output. This will then be a structured collection of data, where connections are located as an event.
- As a journalist/news agency, it's important to have a good news aggregator, to have the opportunity to present articles early with relevant news for the target group. As we know, this is important because they want to reach out quickly, and to the largest possible audience.

Most complex query

```
def item_lifter(external_item):
    sparql.setQuery("""
    PREFIX nhterm: <https://newshunter.uib.no/term#>
        SELECT ?time ?text ?irl ?contributor ?collection
            WHERE
            {
            ?item a nhterm:Item ;
                nhterm:originalText ?text ;
                nhterm:sourceDateTime ?time ;
                nhterm:sourceIRL ?irl ;
                nhterm:hasContributor ?contributor ;
                nhterm:inCollection ?collection ;

            FILTER(STR(?item) ="%s")
            }
            """ % external_item)
    sparql.setReturnFormat(JSON)

    return_block = sparql.query().convert()
    for result in return_block["results"]["bindings"]:
        return result
```

```
def make_entity_item_dict():
    sparql.setQuery("""
    PREFIX nhterm: <https://newshunter.uib.no/term#>
        SELECT DISTINCT ?item1 ?item2 ?entity ?anchor1 ?anchor2 ?annotator1 ?annotator2 WHERE {

            ?item1 a nhterm:Item ;
            nhterm:hasAnnotation ?superAnnotation1 .
            ?superAnnotation1 nhterm:hasEntity ?entity .
            ?superAnnotation1 nhterm:anchorOf ?anchor1 .
            ?superAnnotation1 nhterm:hasAnnotator ?annotator1 .

            ?item2 a nhterm:Item ;
            nhterm:hasAnnotation ?superAnnotation2 .
            ?superAnnotation2 nhterm:hasEntity ?entity .
            ?superAnnotation2 nhterm:anchorOf ?anchor2 .
            ?superAnnotation2 nhterm:hasAnnotator ?annotator2 .
            FILTER(?item1 != ?item2)
        }
    """)
    sparql.setReturnFormat(JSON)
    entity_graph = sparql.query().convert()
```

Maintainability

For maintainability we would use GitHub to version-control and maintain the code. We would also put the program in a Docker to include all dependencies and other requirements when using the program from a new computer.

Reflections

It has been quite a challenge to complete this project. We started off pretty good and made some decent progress the first weeks with visualizing the problems we had to solve and made some code which worked well to do some basic tasks. However, we quickly hit a brick wall and really struggled with finding ways to program for the queries we were after. We probably spent too much time in the beginning with trying to query for a single property and too little time on the whole task at hand. The start of the semester was deeply impacted by the corona-situation and we found that communicating was quite hard.

We had to deal with not being able to meet each other face-to-face and communicate and also to schedule regular in person meetings. We did have some digital-meetings, but generally we found these to be less productive than in-person. Motivation was tough given the difficulty of learning a bunch of new technologies, terminology, syntax and then having to convert this knowledge into running usable code. Not being able to meet regularly and having to work on difficult tasks alone impacted the result negatively.

The results produced by our program are also not necessarily 100% correct in regards to the entity that is being referenced and the data within that related to that entity, however we found sparse and few deviations from what we think are pretty good results and the return entities seem to match the other values in the graph in most cases.

If we were to do the project again, we would start earlier, make plans for deliverables and deadlines on part of the work, try to make a plan for digital/physical meetings, focus more on the broader task at hand and not individual difficult tasks to get stuck on.