Candidate 125
Candidate 126

Intelligent Graph Aggregator

Purpose
The news graph aggregator uses a flow of knowledge graphs with semantically lifted data from online news and social media text sources as its input. These item graphs will then be analyzed, and graphs which describe the same or similar event will be aggregated into a larger graph. The purpose of this is, in our eyes, to give a quick overview of the most current events. Running the program outputs aggregated graphs which should all describe different recent news events.

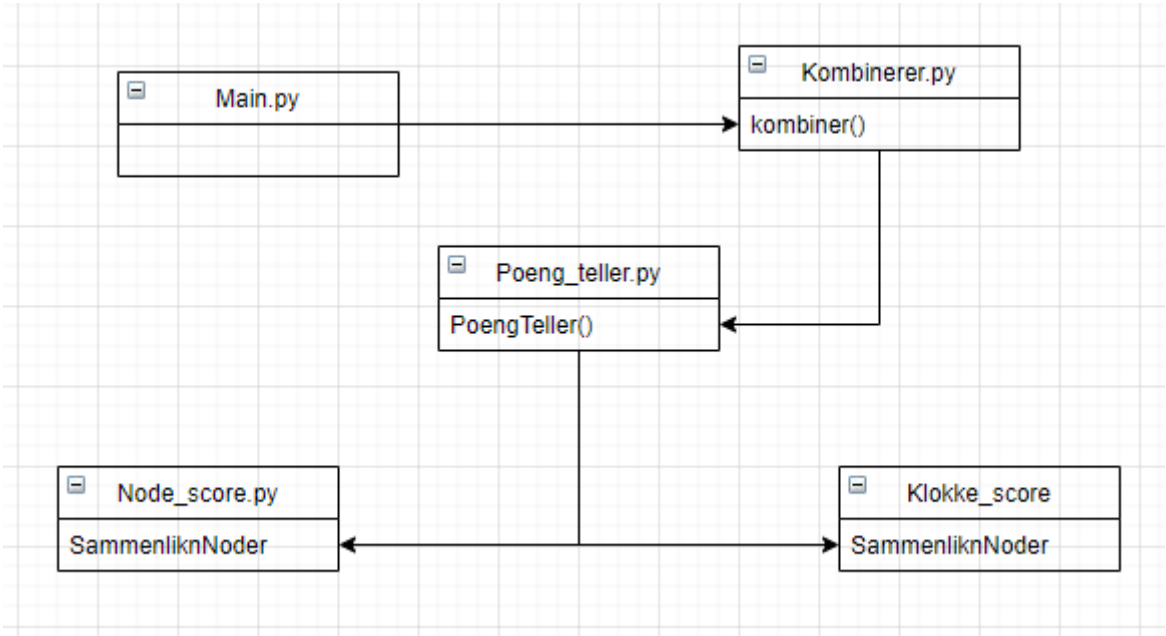
The use of semantic technology allows for a more thorough analysis of the texts, and quicker analysis with similar accuracy as plain natural language processing. We first tried to only use NLP and the date and time of the source, however this was very computationally taxing, and although it found similarities in the texts, it struggled to find and aggregate on real events or similar graphs. Simply comparing the triplets of the graphs, particularly the predicates and the objects, with each other and the text, proved much more rewarding. Comparing nodes is much faster than NLP and the graph groups it produces are more often than not the same event.

The technologies used were:

- rdflib
- sparql
- nltk

This was enough to get an accurate aggregator that was not extremely computationally expensive. We did other methods such as:

Checking for metadata similarities on dbpedia, we could not find a way to compare this information in a way that resulted in higher accuracy however.
Looking at the use of similar vocabularies, the thought was to see if two graphs use for example the ‘bio’ vocabulary, they are probably about the same event if they both happened very recently or close to each other in time.
We hoped that OWL could be useful for transitivity, as if two graphs are about the same event, but use different words, these similarities might not be caught by our algorithm. With transitivity we hoped that these words could work as subclasses of one another. However since this information is not innate in OWL or the graphs, we would have to add it ourselves, and would only apply to those transative features we had recorded. There are NLP solutions for this, however they are too slow and would go against one of the purposes of this program.



A simple class diagram of the program's main function. Here the arrows represent ‘Gets from’.

Hurdles and hindrances.
Learning “fast api” and “docker” took more time than we expected, and even with quite the investment, we could not get docker to work. We wanted to incorporate the logic of OWL or even RDFS, but we could not find a meaningful way to do it. We also wanted to change the prefixes of our end-product files by using the SPARQL CONSTRUCT method, however we could not get it to work properly.

We noticed that the program would generate many files that are very similar, making the folder of aggregated files difficult to navigate. To remedy this we made the fil_navn_generator.py file. It is meant to give the files meaningful names and attempt to delete duplicates. Ideally the program should not generate duplicates,and it should check the files for similarities. It reads the filenames and checks its datetime value and length. We tried doing a thorough check of the files, however this proved to be much slower, so we chose to simply check the file names.

What we wish we had done
We should have planned our project better, as there were many revisions and failed versions.
Because we worked together with most of the project, it was important that we wrote comments explaining each other's code. Looking back we wish we could have been even better at this, as this could have saved us time.
We should have had a better comparison algorithm, we focused a lot of our attention on the efficiency of comparing two files against each other, but the “sorting” algorithm could probably have pruned many files as several very similar event files are produced.
While most of the graphs are well aggregated based on what event they are describing, the ones that are not usually have nothing to do with each other. We should have included a filter for events that are very dissimilar textually.

Contributions

Candidate 126
Code cleanup, coding simple functions.

Candidate 125
researched docker, set up fast api

The rest of the project was made in collaboration, usually working on the same files at the same time in “code with me” sessions.