WEB DATA PROCSSING SYSTEMS

Assignment 1

NOVEMBER 16, 2020

GROUP: 13
Chaoran Li (2688458)
Haochen Wang (2698251)
Mostafa Doroodian (2661234)
Ian Berkhout (2683737)

Introduction

The first assignment is related to the implementation of entity linking. A program is developed that's able to extract entities from a collection of websites and link these entities to the corresponding entity in the knowledge graph Wikidata. This process in executed using three steps: NLP processing, information extraction and disambiguation & linking. What these steps entail is discussed below.

NLP preprocessing

In the first stage of linking entities, data needs to be extract from a HTML file that's located in a WARC file. A WARC file or also known as a Web ARChive is a method that combines digital resources into an aggregate archive file (Wikipedia, 2020). At the beginning, the WARC file is read in binary and analyzed by the ArchiveIterator function from the Warcio library. This library is a fast way to write and read WARC files (ikreymer, 2020). After application of Warcio, the record_id is extracted from the header and used as the key. The record_id is only extracted if the record_type shows the correct response from all records in the WARC file. After which for each id the corresponding HTML file can be obtained, and the content extracted. To obtain the text from the HTML file, the library Beautiful Soup is used to parse the HTML file through use of a LXML parsing analyzer. The raw text is cleaned from any expressions that are certainly not entities (emojis, symbols, etc.), this is to increase efficiency, performance of entity detection, lower used memory and processing time. Therefore, we concentrate more on characters and basic punctuations. After the text is cleaned, everything is assembled with the correct record_id and exported to a JSON file.

Information extraction

From the cleaned text, that was obtained after the NLP preprocessing, the entities need to be extracted. This is done using a technique called Named-Entity Recognition. Named-Entity Recognition (NER) focuses on extracting entities of interest located in text, such as locations, persons, etc. NeuroNer is a NER Program based on a neural network (Dernoncourt, 2019). The

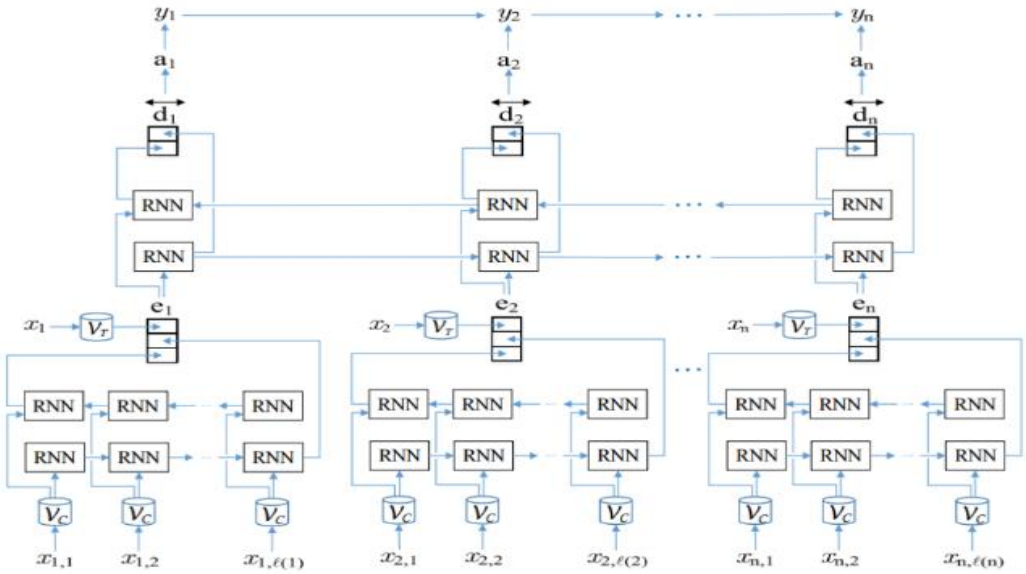

Figure 1: Diagram of the neural network used in the neuroNER (Dernoncourt, 2019)

neural network used is created by Franck Dernoncourt and extracted from a Python library named Neuroner. This neural network is based on a recurrent neural network called long short-term memory and contains three layers: character-enhanced token-embedding, label prediction, label sequence optimization.

NeuroNER is chosen for the entity recognition, in this assignment, so the program can recognize entities based on statistics instead of solely on tricks and prespecified rules. The library developed by Franck Dernoncourt is used, to prevent having to spend a lot of time on developing a neural network. The library also contains multiple pretrained networks, therefore no time needs to be spent training the network. Entities are outputted in a Python dictionary using the following indexation: {'id': String, 'type': String, 'start': Integer, 'end': Integer, 'text': String}. From the generated python dictionary the type and text are saved along with the record_id. After all the entities are extracted the entire set is exported to a JSON file.

Disambiguation and linking

The final step of this program is to find the corresponding Wikidata link for each entity. Using a library called Elasticsearch we obtain a list of possible links. The obtained links are related to the entity in some way, for example the entity could be part of a title of a book or somebodies last name. The correct link needs to be filtered out, mainly based on context of the text it's mentioned in. The technique implemented in the program determines the cosine similarity between two vectors. The idea is that the link with the largest similarity is the correct link. For this technique two articles are selected, the first article is the entity and the second is the description of the link provided by Elasticsearch. The articles are merged into a set and the word frequency of each article is calculated. After which the word frequency vectors are generated, and the cosine similarity is calculated using the following formula:

$$cos(\theta) = \frac{\sum_{i=1}^{n}(x_i \times y_i)}{\sqrt{\sum_{i=1}^{n}(x_i)^2} \times \sqrt{\sum_{i=1}^{n}(y_i)^2}}$$

The link with the highest similarity is selected as the correct link.

Parallelism

The procedure of the program from read a WARC file to linking is like a pipeline for each record of HTML. Considering the scalability, parallelism is applied to speed up the execution. Using the thread pool, the virtual CPU is utilized to execute NER, searching and linking. To fit different machines, it depends on the number of threads of multiprocessing. It will process record by record instead of the whole file, which can reduce the time of execution.

Results

The program was tested using a provided file called sample.warc. In figure 1, the result can be seen.

Considering that the maximum f score is 1, it can be concluded that the program doesn't work as hoped. The result show that while a large number of entities are extracted from the text the linking to the Wikidata links lacks precision. Hence, our implementation of determining cosine similarity maybe is not suited for this program.

The running time for the program is quite large, it was observed that most of the processing time was taken up by the linking. While the program is scalable due to it being able to handle large and small dataset. It's still wise to consider that large dataset will have a significant running time.

There are multiple ideas regarding how we can improve the program and the results. These ideas are discussed in the next section.

Table 1: Results using sample.warc

Gold	**500**
Predicted	4229
Correct	71
Precision	0.017
Recall	0.142
F1	0.03
Running time	~1 hour, 15 min

Recommendations

As is seen in the results the program has some points it needs to improve on. The most obvious is the precision. Precision is increased by picking the right links for the entities and this is done by a linking algorithm. Instead of using the cosine similarity it would be a better option to make use of the type of entity that's predicted by the neuroNER. The neuroNER algorithm that is currently used is able to predict if an entity is for example a person or a location. Using this information, the links that don't match this type can be filtered out. Should more than one link be left over after filtering, it could be interesting to look at if the links are related to other entities in the text. The link that's related to the most entities would then be deemed as correct. This method requires being able to retrieve information from Wikidata using the links provided by Elasticsearch. This can be done by using SparQL queries in combination with certain libraries like Trident. The SparQL queries weren't implemented in this program mainly due to lack of knowledge. We were not able to formulate the proper query, which forced us to use different methods.

The second point that could use improvement is the run time. At this moment the only multi-processing technique that is used is parallelism. Implementation of for example more threads could in theory speed up the algorithm significantly. This was to be implemented after the linking algorithm was completed, but due to low precision the remaining time was spend trying to improve the linking. Therefore, this wasn't not implemented in the current program due to a lack of time. It could also be interesting to see if the use of different information extraction and NLP processing techniques has any significant influence on the running time. Another possibility for the lowering of the running time is removal of duplicate entities, seeing as linking the same entity twice is unnecessary. This will lower the amount of operations to be executed and will hence lower the running time

References

Dernoncourt, F., 2019. *NeuroNER.* [Online]
Available at: http://neuroner.com/
[Accessed 10 November 2020].

Dernoncourt, F., 2019. *NeuroNER.* [Online]
Available at: https://github.com/Franck-Dernoncourt/NeuroNER
[Accessed 11 November 2020].

ikreymer, 2020. *warcio.* [Online]
Available at: https://github.com/webrecorder/warcio
[Accessed 14 November 2020].

Wikipedia, 2020. *Web ARChive.* [Online]
Available at: https://en.wikipedia.org/wiki/Web_ARChive
[Accessed 13 November 2020].