WDPS Assignment 1 Report

Group 27

- Simei Li, 2738043
- Yiran Li, 2730767
- Kairui Wang, 2731737
- Summer Xia, 2703936

Introduction

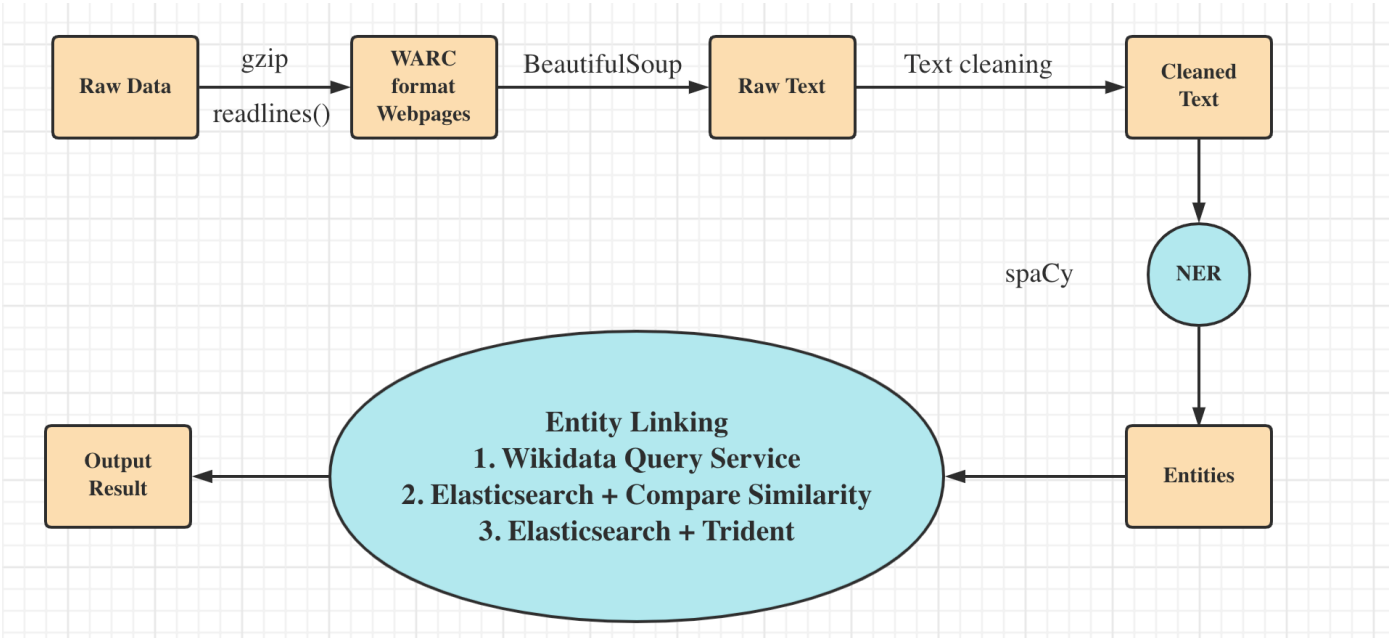
This porject is to implement a program that can recognize entities mentioned in web pages and link them to the entities in Wikidata.

Run Code Instructions

- Run `run_example.sh`, it concludes the evironment package installation
- We have three optional `.py` files, it can be changed in the `run_example.sh` file
  - code_query : use method 4.1 (default)
  - code_es : use method 4.2
  - code_es_2 : use method 4.3
  - code_trident_es_search : use method 4.4
- Git: https://github.com/SummerXIATIAN/wdps_asg1_group27.git

Pipeline

0. Overview and Description



1. Read WARC File

- Use `gzip` to decompress the raw `.gz` file and read the content line by line.
- Set `KEYNAME="WARC-TREC-ID"` in order to seperate the whole file into several parts that each part contains one web page record in *warc* format .

2. HTML to Text

- For each part, remove the *warc* annotation texts and get the remaining information which is in HTML format.
- Use `BeautifulSoup` to read the html data, get rid of the HTML tags and retrieve the text.
- Text cleaning
    - Use regular expression to remove the redundant blank, uncommon punctuations, and Non-English characters.
    - Remove stopwords (not a default operation)

3. Named Entity Recognition

In this NLP part, we tried traditional statisitcal models as well as pre-trained machine learning models. Finally we use the `spaCy` , an open-source library, with its pre-trained model `en_core_web_md`.

Our NER choice

- Load spacy English model `en_core_web_md`
- Retrieve the entities with labels by the built-in function in spaCy : `doc=nlp(text)` , `doc.ents`

Tried but not used:

- basic HMM model (without stopwords)
- Bert (keeping the stopwords gives a better accuracy)
- NLTK package (`pos_tag` , `ne_chunk` in NLTK)

4. Candidate Generation and Entity Linking

We have dicussed and figured out several different methods to get this result, where an entity is linked to its web page in Wikidata. There are differences in processing time and accuracy(recall) between these methods. To use which method may depend on the situation we actually faced. But in the report we will show all the result of each method on the sample test set.

```
# For example
Entity = "Twitter"
Link = "<http://www.wikidata.org/entity/Q918>"
```

4.1 Wikidata Query Service

Wikidata provides a common api called "Wikidata Query Service" that allows people to use sparql to search data out of Wikidata. The figure below shows the query and its result for finding the specific web page of one entity. Therefore, we can use it to find the web page which is linked with our entities.

Recall in the trident, we have the same library of Wikidata so that we can also use the query and it should return the same result. However, in fact, it cannot. Finally, we have to use the online query service to get the answer, resulting in a quite long restful http request process.

4.2 Elasticsearch + Compare Words Similarity

We can query elasticsearch and retrieve many relevant pages (70 pages). Next, the question goes into "how to find the entity that **IS** the one we really want". Here we compare the similarity between the page labels and our entity, and choose the most similar one as our linking result, by using `difflib` library.

4.3 Elasticsearch + Consider word meaning in sentence

When get candidates from elasticsearch, extract the “description” from the response which describes the meaning of the candidate. Next, get the sentence that includes the entity in the clean text.

Then we build the BOW model of these two sentence, and calculate the Cosine similarity of these two sentences, the similarity describes how similar the two sentences are.

Finally, for each entity, we compare all the candidates's description with the sentence include the entity. We choose the candidate that has the highest similarity as our result for that entity.

4.4 Elasticsearch + Trident

In addition to the above, our group also used other methods which involves using Trident.

Firstly, it was known that with the higher entity popularity, the entity is more likely to be the expected result. To find the more popular candidate, we use SPARQL to find all possible properties and items when the entity id comes from Elasticsearch is input as subjects. The more results there represent the more links and citations it has with other entities, which reveals its popularity. The highest linked entity ID would be output.

One way is to prepare a bunch of entities and their corresponding wiki data entity id in advance. Find the entities' label through NER. While due to NER may not be accurate enough, so some examples were added manually. Query the instance of these entities through trident and save the entity ID which has a higher number of occurrences in the testing. When processing with the WARC file, the entity would be checked if it is an instance of the types from the model.txt. This method can filter the candidates with the expected type.

Some Tricks

1. Rule method
    - We have a "sample-label-cheat.txt" file, which lists some entities and their linking pages. To achieve higher score in the real case, we can add this file as rule to our existed method. Combining models and rules is popular in industrial pipelines, we also can do it. (Not used for testing result.)
2. Multiprocess
    - Since the program does a "linearly" job. We use `multiprocessing` library to create multiple processes to do the job, and finally collect their output into one file. In this way, the processing time is reduced exponentially.

Result

Method	4.1 Wikidata Query Service	4.2 Elasticsearch + Compare Similarity	4.3 Elasticsearch + sentence meaning	4.4 Elasticsearch + Trident
Gold	500	500	500	500
Predicted	4752	13666	17364	-
Correct	134	63	79	-
Precision	0.0282	0.0046	0.00454	-
Recall	0.268	0.126	0.158	-
F1	0.05102	0.00889	0.00883	-
Time	3h 3 min	2h 47min (request public ES)	-	-

Improvement

Execution Time

The biggest problem in our discovery is dealing with the local elasticsearch and trident service. During the query process, we often encounter connection interruptions. In order to get an available result, we laterly use the public server for elasticsearch and trident service. It works well but it runs more slower since the http request is involved.

Accuracy

A good program for this problem should provide both a good precision and recall, which means it not only gives the right linking between entity and Wikidata url, but also performs well in named entity recognization. In our work, bert with google pre-trained model gives the best NER result but it is too "heavy" (and slow) for this project. Finally, we use spacy and its general model. Since we will never know what kind of web pages are we going to deal with, the NER model performance can be improved if some train data set is given.

I/O

Some of us use `fo.readlines()` to deal with the text, which may require large memory. If the target file size is very large, the program may not work.