Report

Group11: Wenchen Lai, Shuqi Yan, Jiamian Liu, Xiaoyu Yang

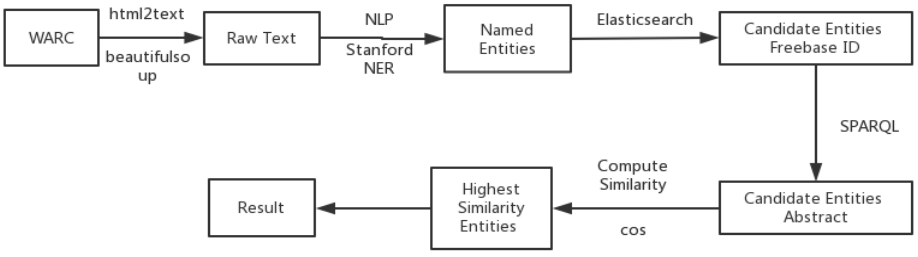
This project is to perform Entity Linking on a collection of web pages.

1. Most important choices:

The most important choice is to choose context-dependent features to rank candidate entities. Elasticsearch can return 10 candidate entities with highest popularity scores. At first, we rank these 10 candidate entities by computing cosine similarity of candidate entity's abstract and the text where the entity mention retrieved from. However, it seems not enough. We found that in the abstract, the first noun phrase (before the first verb) is always highly relevant to the entity mention (sometimes the same). Then, we also consider this cosine similarity, but it is really hard to find a good weight to decide which similarity is more important. Though 1:2 seems better on sample, we still use 1:1. The process helps us perform the pipeline of Entity Linking more clearly.

2. Technologies and motivation:

We perform Entity Linking mainly in 4 steps: extracting text from HTML pages, NLP preprocessing, candidate entities generation and candidate entities ranking. The following picture presents the pipeline of our method. We also perform our method with Spark on cluster mode. Section 4 shows the performance of our method.



The first step of our method is to extract text from HTML pages. The input data is a gzipped WARC file. It contains document IDs and HTML pages. In order to pull the text out of HTML pages, we use "BeautifulSoup4" package to remove all the HTML tags and useless text, because "BeautifulSoup" is easy to use with its clear tutorial and has a good quality of extraction.

The second step is NLP preprocessing. NLTK is a famous python package to do natural language processing. NLTK also provides a wrapper to Stanford NER tagger and Stanford NER can enhance the accuracy of the named entity recognition. We use NLTK to tokenize the text and remove stop words. Then, the named entities can be recognized by Stanford NER. However, Stanford NER takes about 20 minutes to recognize named entities on sample data, while NLTK NER Chunker just costs 5 minutes. Stanford NER seems too slow, so now we choose NLTK NER Chunker to perform NER.

Then, the next step is candidate entities generation. Elasticsearch is an open-source, RESTful, distributed search engine and it has been set up on the DAS-4 cluster. We use the Elasticsearch to link each entity mention to a set of candidate entities in Freebase. In this step, we will get 5 candidate Freebase entities IDs with highest popularity for each entity mention. Although sometimes Elasticsearch met errors due to conflicts between instances, Benno helped us to fix the bugs and gave every team to their own copy of Elasticsearch.

The final step is candidate entities ranking. This task consists of two subtasks: 1) query the candidate entities' abstract from DBpedia using Sparql; 2) ranking all possible entities based on how likely they are link targets. Sparql is a query language and a protocol for accessing RDF and has been implemented in Trident REST service on the DAS-4 cluster. We can use Sparql to query candidate entities' abstracts from DBpedia with their Freebase IDs. For the second subtask, we consider two cosine similarities. One is candidate entities' abstract and the text where the entity mention retrieved from. The other is the entity mention and candidate entities' abstract object which is the noun phrase before the first verb in abstract. Scikit-learn provides simple and efficient API to convert a text to a matrix of TF-IDF features, which helps us to compute cosine similarity more easily. We use Scikit-learn to compute two similarities and sum of them is candidate entity's similarity score. Then, we can link the entity mention to the candidate entity with highest similarity score and print the result.

We also use Spark to perform distributed computing to allow our approach run on large-scale dataset. First, Virtualenv is used to package up a virtual environment with necessary python package (nltk, beautifulsoup4, etc). Then, we use the RDD operations to compute the task.

3. Work Division

html2text, NLP: Xiaoyu Yang, Wenchen Lai
Elasticsearch, spark, Virtualenv, scalability: Jiamian Liu, Wenchen Lai
Sparql, Starter-code: Wenchen Lai, Shuqi Yan
run.sh: Shuqi Yan
Readme: Wenchen Lai, Shuqi Yan, Xiaoyu Yang, Jiamian Liu
Report: Xiaoyu Yang, Jiamian Liu, Wenchen Lai, Shuqi Yan
Extension: Jiamian Liu, Xiaoyu Yang, Wenchen Lai, Shuqi Yan

4. Performance

The performance of our method is measured by precision, recall, F1 score and runtime. We run our code on sample dataset. The best F1 score shown in the following table is 0.0537. The maximum reservation time of nodes on DAS-4 cluster is 15 minutes, which means that after 15 minutes Elasticsearch and Sparql services are stopped. We create two temporary dictionaries to save what have been search by Elasticsearch or query by Sparql in cache. This method improves our efficiency, but not obvious. It the reservation time is 15 minutes, the best F1 score of our program is 0.0278. If the reservation time is 90 minutes (the first 100 pages can finish), the best F1 score is 0.0537, as shown in the following table. We calculate the average time of processing first 10 pages when Elasticsearch and Sparql are still working and the runtime is about 25 seconds per page.

gold	560	predicted	2008	correct	69
precision	0.0344	recall	0.123	F1	0.0537

5. Extension

5.1 Motivation

Detecting salient entities is as important as salient object detection in image recognition. We could not only know what are popular in corpus but also learn the meaningful relevant parts of these “popular” entities. For example, we got several entities from one text includes Trump, building, EU, flower, etc. Only after we detect the salient entities such as Trump, EU, we could know what is mainly stated in this text. This kind of processed knowledge is what we need and could be used in big data analysis in the field of policy analysis, commercial activities, etc.

5.2 Idea and Novel

We assume that the most salient entities are relevant to the title of HTML pages and compute Jaccard similarity of entity’s abstract and title of the page. It is a simple way to detect salient entities than using machine learning, such as a neural network.

5.3 Implementation

Based on the entity linking, we extra the title of each HTML. Then, we compare the title and abstract of all entities in the text by Jaccard index. The Jaccard index is used on Bag-of-Words model of a sentence. The Jaccard index is defined as follows on two sets A and B :

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

After that, the highest score entity is the most salient one.

5.4 Limitation

Sometimes our implementation has a low recall because the title and salient entities are not relevant. We could introduce the Relevant Sentence Detection (RSD) model to compute the relevance score of each sentence for each website joined with the abstraction which we plan to construct. This kind of model involves many parameters involved in the similarity and importance of sentences. After introducing the RSD model, one possible improvement is that we could train the result of the model via a multi-layer BP neural network with the help of Tensorflow. Thus, we could know which are the right salient entities of the text and if we have lost any salient entities. The accuracy of this task will undoubtedly be increased.