Building a Knowledge Extraction Engine

Azim Afroozeh, Jinfu Chen, Wesley Shann, You Hu
{a.afroozeh, j14.chen, w.genizshann, adolphus.hu}@student.vu.nl

***Abstract*—The Internet is a valuable corpus with a large amount of knowledge behind it. Compared to create a hand-made corpus, extracting knowledge from the web is a more efficient and inexpensive way. Entity linking is an essential part of knowledge extraction. Here we build an engine to extract entities from web pages. By testing the engine in sample dataset (93 pages), it achieves performance with an F1 score of 0.13.**

I. Introduction

THE web contains a huge amount of information available online. Some of this information is structured in easily accessing formats, such as databases tables, CSV tables and JSON objects. However, a significative amount of this information is stored without any structured format, this accounts for posts, blogs, books, web pages, etc. In that sense, knowledge extraction techniques and tools are used to transform this raw text in useful information.

The first step in a knowledge extraction process involves executing Natural Language Processing (NLP) tasks in the unstructured text. NLP is composed of a set of smaller tasks, such as tokenization and lemmatization, that are combined together to achieve the detection boundaries of sentences present in a document. There are several open-source available tools to execute NLP tasks, such as Stanford NLP[1], Apache OpenNLP[2], Beautiful Soap[3] and LingPipe[4].

Next, a search for named entities in the text is done. A named entity is anything that can be named, such as people, countries and organizations. After they are found, they are classified. This process is called Named Entity Recognition (NER) and is sometimes categorized as an NLP task.

After detecting entity mentions in the document, they should be linked to their mapped candidate entities in a Knowledge Base. This process consist of three main operations:

- **Candidate Entity Generation**: Link any entity mention to a set of candidates entities in the knowledge base.
- **Candidate Entity Ranking**: Ranking the most likely match between the entity mention and candidates entities.
- **Unlinkable Mention Prediction**: Detect and handle all cases in which there is no suitable candidate entity to a particular entity mention.

In each stage, there is a problem to be addressed. In the NLP phase, for instance, due to the liberty in which web pages, posts and blogs can be written, it is normal to not have a perfectly written text. There are also numerous words that are so frequently that are considered noise, such as "of", "the" and "a". These words can, however, be part of composed words or modify the sentence meaning, for example, "King of Hearts" and "Bank of America".

The previous example illustrates a challenge for the named entity recognition task, as there as named entities that are defined by 2 or more words. Besides the two previous examples, another example would be identifying that "United States of America" is the name of a country and not individual words.

Regarding entity linking, there is the scenario that each entity mention which needed to be linked in the unstructured text does not have the exact attribute values and the size of candidate entity for each mention is often larger than one. This makes the task of ranking the set of candidate entities and selecting the best entity candidate for each entity mention quite difficult. Another problem is to handle the unlinkable situation. It is possible that one entity mention does not have its corresponding entity in the Knowledge Base, this kind of mention would be marked as "NIL" which means "unlinkable".

In this project we design and implement a knowledge extraction engine, adhering the tasks previously described. The remaining of this report is structured as follow: In Section II, is given an overview of the used tools. In Section III, a detailed description of the proposed engine is defined, explaining the executed tasks. The experiments and results from executing the engine are described in Section IV. A discussion of the overall solution and results is given in Section V.

II. Technologies and Libraries

A. DAS

The Distributed ASCI Supercomputer (DAS) 4[5] is a six-cluster wide-area distributed system. It is accessed by multiple institutions, including the VU Amsterdam, for research and classes. Therefore, the sample and testing

This report was submitted for the lab assignment of the 2018-2019 edition of the Vrije Universiteit Amsterdam, Computer Science Master Degree, Web Data Processing System course, under supervision of prof. dr. Jacopo Urbani (jacopo@cs.vu.nl) and Benno Kruit (bennokr@gmail.com)

[1]https://nlp.stanford.edu/software/CRF-NER.shtml
[2]https://opennlp.apache.org/
[3]https://www.crummy.com/software/BeautifulSoup/bs4/doc/
[4]http://alias-i.com/
[5]https://www.cs.vu.nl/das4/

data used for this project, as well the testing and deploying environment are located in the DAS 4 cluster. For this project, we can use 8 head nodes and multiple work nodes.

B. FreeBase

Provided by Google, FreeBase[6] is a knowledge base with 1.9 billion of registered tuples. The base contains 22GB of compressed data, totalizing 250GB when uncompressed. The data is stored in the N-Triples Resource Description Framework (RDF) format.

Although the Freebase has been discontinued, it was used in this project due to three reasons. First, it is available in the DAS-4 cluster. And this project have two imposed constraints: The engine must run on the DAS-4 cluster and must not access external data. Therefore, any knowledge base not available in the cluster can't be used. Nevertheless, by using FreeBase, the Candidate Entity Generation task can be skipped

C. Beautiful Soap

An HTML/XML parser for Python, Beautiful Soup (BS) is used to convert complex HTML documents into a complex parse tree structure. It is powerful enough to be able to parse even invalid markup. Each node of the tree is a python object, and all objects can be classified into four types:

- **Tag**: Is the XML/HTML tag present in the original document.
- **NavigableString**: Is the text within the tag.
- **BeautifulSoup**: Represent the document as a whole.
- **Comment**: Is the HTML comments present in the original document. Everything not classified in the previous object types, is classified as a comment.

We decided to use version 4 of BS over version 3 due to the fact that it has more features, is faster and works with third-party parsers, such as lxml[7] and html5lib[8].

D. Spacy

A Python and Cython open-source library for NLP tasks, Spacy provides a concise and complete API to access its methods and properties, such as tokenization, part-of-speech (POS) tagging, entity detection, and dependency parsing. Therefore, we decide to use it for the extraction of entity mentions.

E. ElasticSearch

A highly scalable open-source full-text search and analytics engine, ElasticSearch allows the use of multiple parameters to query any kind of document and returns JSON data that is easy to be handled afterwards. It is convenient to leverage it to query FreeBase and rank entities with their scores.

[6]https://developers.google.com/freebase/
[7]https://lxml.de/
[8]https://pypi.org/project/html5lib/

III. System Design

The designed application has three major phases:

- **Entity Extraction**: Will load the data and execute NLP and NER tasks.
- **Entity Linking**: Will query the FreeBase to link the extracted entities to their matches in the knowledge base.
- **Relation Extraction**: Will identify relations between the identified entities.

Next, we will describe each of the mentioned phases with more detail and explain the implementations decision taken.

A. Entity Extraction

Processed entirely in Spark, Entity Extraction is composed of two tasks: NLP preprocessing and information extraction. This phase receives the sample data as input in the format Web ARChive (WARC) format, which is a method for combining multiple digital resources into an aggregate archive file together with related information, as defined by the ISO 28500:2017[9]. As a result, a set of extracted entity mentions containing their NER tag in the format *(WARC-File-ID, Label , NER Tag)* are returned. In Figure 1, an example of a few extracted entities is presented.

```
('clueweb12-0000tw-00-00007', '21,977', 'CARDINAL')
('clueweb12-0000tw-00-00007', '21,458', 'CARDINAL')
('clueweb12-0000tw-00-00007', '511', 'CARDINAL')
('clueweb12-0000tw-00-00007', 'Avg', 'PERSON')
('clueweb12-0000tw-00-00007', '      ', 'GPE')
('clueweb12-0000tw-00-00007', ' WordPress', 'ORG')
('clueweb12-0000tw-00-00007', ' Elegant Themes       ', 'ORG')
('clueweb12-0000tw-00-00008', ' Home', 'ORG')
('clueweb12-0000tw-00-00008', 'Feb 9, 2012   ', 'DATE')
('clueweb12-0000tw-00-00008', 'Christ', 'ORG')
('clueweb12-0000tw-00-00008', 'NKJV', 'ORG')
('clueweb12-0000tw-00-00008', 'First', 'ORDINAL')
('clueweb12-0000tw-00-00008', 'Years later', 'DATE')
('clueweb12-0000tw-00-00008', 'all day', 'DATE')
('clueweb12-0000tw-00-00008', 'the Holy Spirit', 'FAC')
('clueweb12-0000tw-00-00008', 'millions of gallons', 'QUANTITY')
('clueweb12-0000tw-00-00008', 'Those millions of gallons', 'QUANTITY')
('clueweb12-0000tw-00-00008', 'Millions of gallons', 'QUANTITY')
('clueweb12-0000tw-00-00008', 'Contextual Related', 'ORG')
('clueweb12-0000tw-00-00008', 'Name * Email', 'ORG')
('clueweb12-0000tw-00-00008', 'href=', 'GPE')
('clueweb12-0000tw-00-00008', 'every day', 'DATE')
('clueweb12-0000tw-00-00008', '7', 'CARDINAL')
('clueweb12-0000tw-00-00008', ' ', 'NORP')
('clueweb12-0000tw-00-00008', 'Popular', 'NORP')
('clueweb12-0000tw-00-00008', '33)Humble', 'CARDINAL')
('clueweb12-0000tw-00-00008', '31)The', 'PERSON')
('clueweb12-0000tw-00-00008', '20)Why', 'CARDINAL')
('clueweb12-0000tw-00-00008', '19)Starting', 'DATE')
('clueweb12-0000tw-00-00008', '17', 'CARDINAL')
('clueweb12-0000tw-00-00008', 'Categories Apologetics', 'ORG')
('clueweb12-0000tw-00-00008', '6', 'CARDINAL')
('clueweb12-0000tw-00-00008', 'Ministry (1) Christ', 'ORG')
('clueweb12-0000tw-00-00008', '18', 'CARDINAL')
```

Fig. 1: Entity Extraction phase result.

In summary, the following tasks are executed to complete this phase:

- **Loading data**: Utilizing Spark, the sample data available in the DAS-4 Hadoop Distributed File System (HDFS) is read using the method *newAPIHadoopFile*, which load the data as a Resilient Distributed Datasets (RDD), a Spark data structure. The loaded data is split into multiple pages.

[9]https://www.iso.org/standard/68004.html

- **Extracting Text**: First, a filter is applied to remove unnecessary HTML tags, such as scripts and styles tags. After that, the text is split into multiple sentences. Both tasks are achieved by using the Beautiful Soap library.
- **Entity generation**: For this task, Spacy offers four trained models in the English language. As described in their website, 3 models are "English multi-task convolutional neural network trained on OntoNotes[10], with GloVe[11] vectors trained on Common Crawl[12]", varying in size; while the fourth is a "300-dimensional word vector trained on Common Crawl with GloVe". As all four have similar properties and accuracies, we opted to select the smaller model, the en_core_web_sm[13]. With this model, an entity generation process is applied to each extracted sentence in the previous step. The result of this operation are entities pairs labelled with their respective NER tags.

B. Entity Linking

Taking the extracted entities from the previous phase and creating a link between these entities and a knowledge graph can be divided into the following steps:

- **Querying the knowledge base**: Querying the FreeBase knowledge base by sending requests to ElasticSearch service with surface form of entities.
- **Ranking the candidate**: The results returned from ElasticSearch were sorted by the score given from the query engine.
- **Entity linkage**: Using the sorted results, the following logic is applied:
  1) For each mention, search the candidate list in the score descending order.
  2) If a candidate and the mention have a lexical similarity of 1 (exactly match), this particular candidate is selected. Lexical similarity is a measure to verify how similar two words are. The highest the measure, the more similar the words are.
  3) If no candidate in the list meets the condition (2), the candidate with the highest score is selected.
- **Filter the unlinkable entities**: At the time transforming the results of candidate ranking, we have set up the following rules to filter unlinkable entities:
  - Remove words containing punctuation.
  - Remove words that start with space or number.
  - Remove non English words.
- **Evaluate the result**: The final result is analysed by calculation the F1 score, which provides a quick overview of the implementation accuracy.

[10]https://catalog.ldc.upenn.edu/LDC2013T19
[11]https://nlp.stanford.edu/projects/glove/
[12]http://commoncrawl.org/
[13]https://spacy.io/models/en#en_core_web_sm

C. Relation Extraction

This phase is an extension of the implemented entity linking task. The relations will be extracted according to two handwritten patterns. Both patterns are used for extracting relation: the entity A is a specific of entity B. For instance, in the sentence "They can combine tax breaks in ways that companies like Netflix and Adobe cannot", we extract the tuple (companies, Netflix) according to the dependency between these two entities. In this example, the entity Netflix is a specific case of the entity companies. The relation extraction can be achieved by executing the following tasks:

- **Sentences generation**: Extract text from sample data, and split text into separate sentences.
- **Dependency parsing**: Parse the sentences to get a (basic) dependency between words.
- **Applying patterns**: Generate the relations with the hand made rules:
  - Pattern 1: nsubj [*] (attr|dobj): "Current version is 32.0209 and fixes a diagnostic error preventing some customers "⇒ (Current version, 32.0209)
  - Pattern 2: [*] perp pobj: "was more deserving of a nomination than 95% of the field." ⇒ (nomination, 95%)

IV. Experimental Results

A. Experimental setup

The experimental environment is the DAS 4. We use python as our programming language, and bash scripts to execute the algorithm.

For phase one, we have used anaconda[14] for the packaging and virtual environment management. The configuration of Spark is set with 8 executors and 2GB of memory per execution with cluster mode. The Spacy model and required libraries are packaged in a virtual environment.

For phase two, the ElasticSearch service will be running in work nodes by using Prun[15] command, a synchronous user interface provided in DAS-4, the service can be visited by sending requests.

The relation extraction experiment was executed on the member local machines, with the same python environment and libraries configured in the cluster. The experiments couldn't be executed in the cluster to the fact that during the development and experiments of the relation extraction phase, the DAS was unstable and in some periods, the connection to the cluster was refused. This could be due to an overload of jobs being executed in the cluster by students of multiple courses.

B. Experiments

Considering there are only 93 web pages in the annotation sample file, the testing input was limited to this amount. The result performance and accuracy will be automatically given at the end of the knowledge extraction

[14]https://www.anaconda.com/download/
[15]https://www.cs.vu.nl/das4/jobs.shtml

processing with the calculation of the F1 score. We used the F1 score because it considers both precision and recal, thus offering a broader view of the engine performance.

The phase one execution is about 10 minutes, however, this time depends on the status of the cluster. If the cluster is overloaded with jobs, the performance will be affected and the execution runtime will be longer. The *venv* file upload will usually take 1 minute. And the second phase can be completed within 5 minutes.

After executing the engine, the F1 score was calculated using the formula in Figure 2, resulting in a score of 0.13 for the implemented engine, with a precision of 0.09 and a recall of 0.23.

$$F_1 = \left(\frac{\mathrm{recall}^{-1} + \mathrm{precision}^{-1}}{2}\right)^{-1} = 2 \cdot \frac{\mathrm{precision} \cdot \mathrm{recall}}{\mathrm{precision} + \mathrm{recall}}.$$

Fig. 2: F1 Score formula.

For relation extraction, we extracted 57000 lines of relations. The whole process can be finished within 10 minutes without parallelization. The relation extraction performance and accuracy was not measured, however, by checking a random sample of the extracted relations, it could be verified that although multiple correct relations are extracted, the accuracy of the extraction is low.

V. Discussion

According to the result, we can observe that the precision score of 0.13, between 0 and 1, is very low. In phase one, we generate multiple entity mentions and filter the irrelevant mentions in the second phase. By applying filter rules, we achieve 10 times improvement in precision if compared to the implementation without such rules. In the output, we verified that there are many names of person or diseases that are not considered as named entities according in the annotation file. However, after checking their FreeBase ID, the result shows that the linking have been succesfully completed.

For the candidate ranking, no complex rules or algorithm was implemented, due to the scores provided by the ElasticSearch were sufficient as well as the project constraints of not accessing any external searches. Nevertheless, the accuracy could be improved by implement features such as bag-of-words or skip gram. Overall, the presented solution is a simple solution to achieve the minimum requirement of accuracy score equals higher than 0.1, but the project still has space to be improved.

VI. Conclusion

In this report, we propose an approach to knowledge extraction based implemented in the Python programming language, combining several techniques and technologies. The designed model is a simple unsupervised model inspired by the principle of existent information retrieval techniques. Despite the achieved F1 score being higher than 0.1, there are still opportunities for further enhancements, such as detecting entity mentions composed of multiple words and increasing the ranking accuracy.

Appendix A
Project

A. Source Code

The source code for this project is publicly available in a Github repository[16] for anyone interested in the project implementation.

B. Team Workload

The workload to complete this project was split by the members between researching, designing, development, experiments and writing this report.

[16] https://github.com/azimafroozeh/WDPS