WEB DATA PROCESSING SYSTEMS

Entity Linking and Relation Extraction

Tianzheng Hu
2760270
Vrije Universiteit Amsterdam
t.hu2@student.vu.nl

Lixiang Zhang
2766774
Vrije Universiteit Amsterdam
l.zhang2@student.vu.nl

Jingye Wang
2702383
Vrije Universiteit Amsterdam
j18.wang@student.vu.nl

Jing Hu
2792797
Vrije Universiteit Amsterdam
j.hu4@student.vu.nl

Abstract

This report presents a computational linguistics approach for the extraction of entities and relations and links them to their corresponding entities within a knowledge base.

First, data preprocessing and cleaning are performed on the given WARC files. After preprocessing, we extract and recognize named entities mentioned in the text and find the candidate Wikipedia links for every entity mention. And then we rank the links to choose the appropriate one. The next step is training a Linear Support Vector Machine (SVM) to gather relations between our entities and link their relation to the Wikidata link using a dictionary.

Keywords: Named Entity Recognition, Entity Linking, Relation Extraction, Relation Linking, Natural language processing

1. INTRODUCTION

In the era of big data, how to extract valuable information from the huge amount of unstructured or semi-structured data has attracted the attention and research of many researchers, and information extraction techniques have emerged. Information extraction mainly includes three sub-tasks: entity extraction, relationship extraction, and event extraction, and relationship extraction is the core task and an important part of information extraction. The main goal of entity-relationship extraction is to identify and determine the specific relationships between entity pairs in natural language text and transform the unstructured information in the text into structured information for storage in the knowledge base. This process provides basic support for intelligent retrieval, semantic analysis, etc., and helps to improve search efficiency and facilitate the automatic construction of knowledge bases.

2. DATA PREPRATION

We open gzipped Web ARChive (WARC) files to get unstructured data. The WARC format specifies a method for combining multiple digital resources into an aggregate archival file and related information[2].

Each file within these files has its unique identifiers.

```
WARC-Type: warcinfo
WARC-Date: 2012-02-10T21:42:47Z
WARC-Filename: 0000tw-00.warc.gz
WARC-Number-of-Documents: 24644
WARC-File-Length: 1045573224
WARC-Data-Type: twitter links
WARC-Record-ID: <urn:uuid:5a67c755-09e8-41f8-b9f9-6e8fcf30f353>
Content-Type: application/warc-fields
Content-Length: 283

software: Heritrix/3.1.1-SNAPSHOT-20120210.102032 http://crawler.archive.org
format: WARC File Format 1.0
conformsTo: http://bibnum.bnf.fr/WARC/WARC_ISO_28500_version1_latestdraft.pdf
isPartOf: ClueWeb12
description:  The Lemur Project's ClueWeb12 dataset (http://lemurproject.org/)
```

Figure 1: Sample WARC Header

We take $< WARC - Record - ID >$ / $< WARC - TREC - ID >$ from this header.

We then extract the HTML content within these files and conduct text pre-processing. There are several built-in functions in python that enables us to do all of this with ease.

3. METHODOLOGY

3.1 NLP and name entity recognition

We use BeautifulSoup to extract text from HTML and use spacy for the NER(Named Entity Recognition). With the help of BeautifulSoup, we verify and remove any (accidentally) leftover tags by creating a blacklist containing the tags' names. For text processing, we clean all non-English words, punctuation marks, and unprintable characters, then tokenize the text. For the convenience of relation extraction, we cut text into sentences. We run the NER provided by spacy to get the text, label, and POS tag of each entity. After that, we mark the first two entities in the original text with $< e1 >$ and $< e2 >$ labels. To perform the task mentioned above, We create an instance of spark and use this context to create an RDD, transform RDD to a spark dataframe and store the result in a parquet file.

3.2 Entity linking

For entity linking, we map each entity to its Wikipedia URL. We build a local elastic search database to improve scalability. For candidate entity generation, we search related entities in the database and get a list of entity candidates. For candidate entity ranking, we calculate the cosine similarity between the original text containing that mention and the description of each entity and select the entity with the lowest distance to the mention.

3.3 Relation Extraction and relation linking

We apply a Linear Support Vector Machine (SVM) to build our model in order to extract relevant features for each of the entities. Our training data is from SemEval2010_task8_all_data[1]. The feature dataset is stored in *features_train.csv* file. Every sentence includes two entities that are labeled by $< e1 >$ and $< e2 >$.

The system as described above has its greatest application in an arrayed <e1>configuration</e1> of antenna <e2>elements</e2>.
The <e1>child</e1> was carefully wrapped and bound into the <e2>cradle</e2> by means of a cord.
The <e1>author</e1> of a keygen uses a <e2>disassembler</e2> to look at the raw assembly code.
A misty <e1>ridge</e1> uprises from the <e2>surge</e2>.
The <e1>student</e1> <e2>association</e2> is the voice of the undergraduate student population of the State University of New York at Buffalo.
This is the sprawling <e1>complex</e1> that is Peru's largest <e2>producer</e2> of silver.

Figure 2: Training sentences

A feature vector extraction method is used to feed our input to our machine learning model in order to train the classifier with a labeled dataset. These labeled features include word tokens, word prefixes, dependency phrasing path, entity pos, and more relevant information that helps the machine better understand the sentence(1).

We filter out the named entities by reading each line of the sentence in the dataset using regular expressions. Linking the entity to other words in the sentence creates a python object to store all the features associated with the entity in this sentence.

After preprocessing the data and extracting all the features. We want to structure the vectors for training. We used the TFIDF Vectorizer, whitespace Tokenizer, and other types of tokenizers to vectorize and transform the feature values. After the feature vector is structured in a manner that the machine learning algorithm can read. We can input the features and train the SVM model we built. We used 75 percent data in the corpus to train the model. After training, we used the left 25 percent data to evaluate the trained classifier model, and it shows 75 percent accuracy. Finally, we used this classifier to predict our entities extracted in name entity recognition. This operation will return the relations between entities.

There are 19 kinds of relations in the dataset(Fig 4). Based on the relations, we built a local dictionary to store the wikidata links of relations. Finally, we query relations in the dictionary of wikidata links for the relations between entities predicted by the model in the previous step.

4. RESULTS

By comparing the performance of several models: SVM, Linear SVM, Random Forest with the feautres we constructed, we found that the Linear SVM performed best for relation extraction with 0.703 f1 scores as shown in Fig 3.

Evaluation

```
print("Accuracy:", accuracy_score(y_test, preds))
print("F1-Score:",f1_score(y_test, preds, average='weighted'))
```

```
Accuracy: 0.7117537313432836
F1-Score: 0.7037227531279986
```

Figure 3: Relation Extraction Model Performance

5. DISCUSSION

Still, we have some limitations for this assignment. For one thing, our initial task is to extract text from HTML content, and due to the complexity of front-end code, this task can be very challenging. Another element is the NER algorithm, which is not that accurate, but there is no way that we are able to double-check its correctness manually.

In further work, the accuracy may be improved by using a better model with more features. Nevertheless, it is also curial to tradeoff the complex of the model.

6. INSTRUCTIONS TO RUN THE CODE

step 1. In order to run the whole pipeline, a computer with python, java, and elasticsearch[1] installed is required.

step 2. In order to install the python dependencies, first run: ./setup.sh

step 3. Then run: python starter_code.py data/warcs/sample.warc.gz

Conclusions

Our knowledge acquisition pipeline architecture mainly consists of two components - an entity detection component and a relation detection component. The input to the knowledge acquisition pipeline is a text string on a web page. We preprocess this text and then perform entity detection to identify all entities in the input text. After the entities are extracted, we perform relation detection. We first extract features from the two named entities and the text content. Then, we invoke the trained relation detection model with the text and labeled entities to identify the relation between the entities.

In order to improve code quality and readability, we added comments for every important function in the code. In order to improve the scalability of our programming, we used elasticsearch as searching datasets. It can be extended as a cluster for more data requirements. We also tried to use pyspark to introduce elastic data processing.

[1]The wikidata elasticsearch indexes are here:

7. APPENDIX

You can find the elastic search indexes that contain wikipedia data through this link

Description	Feature
Entity	e1, e2, e1_e2, e1_e2_bigram
Head	head_e1, head_e2, head_e1_e2, head_e1_e2_bigram
NER	e1_ner, e2_ner, e1_e2_ner
POS Tag	e1_postag, e2_postag
Context	before_e1, after_e2, between_e1_e2
Bigram	before_e1_bigram, after_e2_bigram, between_e1_e2_bigram
Phase	shortest_path, sentence_string

Table 1: Features for model training - Relation Extraction

Cause-Effect e1,e2	has cause	https://www.wikidata.org/wiki/Property:P828
Cause-Effect e2,e1	has effect	https://www.wikidata.org/wiki/Property:P1542
Component-Whole e1,e2	part of	https://www.wikidata.org/wiki/Property:P361
Component-Whole e2,e1	consist of	https://www.wikidata.org/wiki/Q55692548
Content-Container e1,e2	content	https://www.wikidata.org/wiki/Q1260632
Content-Container e2,e1	container	https://www.wikidata.org/wiki/Q987767
Entity-Destination e1,e2	entity	https://www.wikidata.org/wiki/Q35120
Entity-Destination e2,e1	destination point	https://www.wikidata.org/wiki/Property:P1444
Entity-Origin e1,e2	entity	https://www.wikidata.org/wiki/Q35120
Entity-Origin e2,e1	origin	https://www.wikidata.org/wiki/Q3885844
Instrument-Agency e1,e2	instrument	https://www.wikidata.org/wiki/Property:P1303
Instrument-Agency e2,e1	agency	https://www.wikidata.org/wiki/Q352450
Member-Collection e1,e2	member	https://www.wikidata.org/wiki/Q9200127
Member-Collection e2,e1	collection	https://www.wikidata.org/wiki/Q2668072
Message-Topic e1,e2	message	https://www.wikidata.org/wiki/Q628523
Message-Topic e2,e1	topic	https://www.wikidata.org/wiki/Q200801
Product-Producer e1,e2	product	https://www.wikidata.org/wiki/Q2424752
Product-Producer e2,e1	producer	https://www.wikidata.org/wiki/Property:P162
Other nan		

Figure 4: Realtion dictionary

References

[1] Iris Hendrickx et al. "Semeval-2010 task 8: Multi-way classification of semantic relations between pairs of nominals". In: *arXiv preprint arXiv:1911.10422* (2019).

[2] loc.gov. *Sustainability of Digital Formats,Planning for Library of Congress Collections*. URL: https://www.loc.gov/preservation/digital/formats/fdd/fdd000236.shtml.