COMP 6751

Natural Language Processing: Project 1

running text preprocessing pipeline in NLTK in Linux and proofreading results

Submitted To: Dr. Sabine

Submission Date: September 22 2023

Iymen Abdella

Student No. 40218280

INTRODUCTION AND BACKGROUND

This report is an attempt to capture the complexities and challenges involved with creating, testing, and running a text processing pipeline using classical language processing techniques. The pipeline is designed for the analysis and subsequent extraction of relevant entities within large bodies of text.

While natural language entity extraction is a deeply researched and robust field, our project specifically leverages the suite of available processing tools from the Natural Language Toolkit library (NLTK) [1]. Praised for its comprehensive offerings such as Parts-of-Speech tagging and plentiful tokenization options, NLTK has emerged as an industry benchmark particularly for classical open-source development as evidenced by its more than 2.7 million weekly downloads [2]. Necessary, but not sufficient for our purposes, we import the library as part of our python environment along with the popular Pandas library [3] for data manipulation.

GAZETTEERS

Furthermore, we derive a sufficiently exhaustive list of measurements for our unit gazetteer by leveraging the Pint library [4] for unit representation. While its primary use case is the complex computation of unit measurements in the realm of scientific and engineering applications, we make use of its extensive list of stored default units. Secondly, our currency and country gazetteers are acquired from the GeoNames [5] library of "geographical database cover[ing] all countries and … available for download free of charge." GeoNames was mainly chosen for its extreme level of detail, and open accessibility via a freely available API for file downloads.

The content of our gazetteers was entirely pulled from external sources, which introduces concessions. The maintenance and responsibility of support is delegated to an external source beyond our control. Additionally, whenever an external source is introduced, we risk future compatibility issues if modifications to the accessibility end up being inconsistent with our implementation. However, the benefits far outweigh the risks and downsides especially in our case where the sources are reputable, and we can trust their ability to effectively maintain their data. It saves considerable time during implementation because we are not required to research, enumerate, and validate the gazetteer data. Overall, relying on a reliable external source is far more time efficient than developing a domain specific gazetteer from the ground up.

IMPLEMENTATION DETAILS

We dedicate this section to a discussion involving the implementation details of the *preprocess.py* script created as a pipeline for natural language interpretation. Of course, the code base contains in-line comments for further elaboration on the specifics of the code.

TOKENIZATION AND SENTENCE SPLITTING

Tokenization and sentence splitting remains unchanged from the default implementation from the NLTK library. We employed the sentence tokenizer on the raw text data before individually tokenizing the sentences to get a list of words.

POS TAGGING

To accomplish Parts of Speech (POS) tagging we used the POS tagger found within the NLTK library without any modifications. The performance of the POS tagger was sufficient, but inconsistent. The tagging also performed more favorably on contemporary publications during testing compared to pre-contemporary publications where some words were abbreviated or perturbed.

This behavior could possibly be attributed to the NLTK developers preferring a modern interpretation of words rather than how they may have been understood in the past. If so, then it stands to reason that POS taggers should be adapted to be time sensitive for the content they are intended to tag. However, the documentation of the POS taggers [6] indicates a keen awareness of this inherent limitation and offer a variety of alternative taggers with room for expansion upon them:

1. **Default** tagger: labeling every word as the same POS.
2. **Regular Expression** Tagger: Sequential matched regular expressions.
3. **Lookup Tagger:** create a lookup table for frequent words.
4. **N-Gram Tagging:** statistical approach of assigning the likeliest POS for a word.
5. **Combining Tagger:** a combination of taggers working in conjunction.

GAZETTEER ANNOTATION

As alluded to earlier, the gazetteer was extracted from external sources and processed using the pandas library to form three distinct lists:

- **Gaz_country:** gazetteer extracted from GeoNames (countryInfo.txt).
- **Gaz_currency:** gazetteer extracted from GeoNames (countryInfo.txt).
- **Gaz_units**: gazetteer extracted from Pint default units.

Before looking up the words in the gazetteer we eliminate an edge case when the word has an unacceptable POS tag and account for the case when a unit is pluralized. Then we simply look up the word in our gazetteer and tag them with the appropriate list if found. In general, the performance of the gazetteer was adequate, but could be further improved by adding more elements to the list such as:

1. Alternative spelling or aliases of countries: ‘U.S.’ for the United States of America.
2. Pluralized units with uncommon spelling: pluralized for of the unit ‘ton’ is ‘tons’, but ‘tonnes’ must also be accounted for.

NAMED ENTITY (NE) RECOGNITION

We employed the named entity recognition module provided by NLTK: **ne_chunk** but with *binary=True*. Performance was sufficient for our analysis, but as cautioned by the documentation [7] there were indeed edge cases unaccounted for with the NLTK implementation particularly with classification of named entities when the *binary* parameter is omitted. One vexing issue during testing was the extreme case sensitivity the detector has, which can impact words in all caps as a modern convention for heightened emphasis.

Finally, to capture the output of the processing steps we combine the POS, word tokenization, named entity recognition and gazetteer annotation into one dataframe which gets saved as output for further analysis in a locally available .csv file.

MEASURED ENTITY DETECTION

There was no measured entity detection implementation available within the NLTK library, so we designed our own based off the concepts of the regular expression tagger. In particular, our implementation seeks to capture all the numbers within the sentence using a list of regular expressions. Then those matches are combined with the next word in the sentence, but only if they are tagged as a noun or ‘cardinal numbers’.

TESTING STRATEGY AND SAMPLE OUTPUT

Below we present a varied sample of outputs used for testing, each with their associated csv file for POS tagging, word tokenizing, named entity recognition and gazetteer annotation. The measured entity detection is given as a command line output. Despite the few test cases there are signs of sufficient performance, but ample improvement opportunities.

File ID	Measured Entity Detection	Relative Path to C
test/14826	['10 billion', '15.6 billion', '95 pct', '53 billion', '7.1 billion', '4.9 billion', '30 pct']	output\test\1482
test/14828	['19 provinces', '12 pct', '1.575 mln', '25 pct', '2.1 mln', '30 pct']	output/test/1482
test/14832	['4.5 billion', '2.1 billion', '65.1 billion', '58.7 billion', '23 pct', '60.6 billion', '56.6 billion', '18 pct', '66 pct', '45 pct', '26 pct', '64 pct', '57 pct', '35 pct']	output/test/1483
test/14829	['550 mln', '600 mln', '27 pct', '23 pct', '21 pct']	output/test/1482
test/15618	['320,000 tonnes', '225,000 tonnes', '25,000 tonnes', '20,000 tonnes', '139 Ecus', '145 Ecus', '142.45 Ecus', '138.50 Ecus', '40,000 tonnes', '139 Ecus', '85,000 tonnes', '130 Ecus', '55,000 tonnes', '131 Ecus', '105,000 tonnes']	output/test/1561
training/9865	['675,500 tonnes', '245,000 tonnes', '22,000 tonnes', '20,000 tonnes', '141.00 Ecus', '141.81 Ecus', '137.65 Ecus']	output/training/9
training/9880	['25 MLN', '25 mln', '266 mln', '350 mln']	output/training/9
training/9920	['7,600 customers']	output/training/9

LIMITATIONS AND SUGGESTED IMPROVEMENTS

TOKENIZATION AND SENTENCE SPLITTING

I believe the NLTK based tokenizer is sufficient, but incomplete on its own because it fails to account for certain edge cases. Such cases include:

1. **A large number** separated by a space is interpreted as two separate numbers. Ex: 30 000 is tokenized as '30', '000' instead of '30 000'.
2. **Contractions** are interpreted as two separate words separated by an apostrophe, rather than a complete word. Even for words that employ a possessive 's' Ex: Japan's is tokenized as 'Japan', ''s' instead of 'Japan's'.
3. **Locations with spaces:** some locations are interpreted as two words. Ex: 'Hong Kong' is tokenized as 'Hong', 'Kong', instead of 'Hong Kong'. This issue is particularly grating because it interferes with gazetteer annotation.

However, these are minor edge cases and can be easily accounted for with some supplemental regex detection, or contextual analysis with POS tagging. For example, (1) can be solved by supplementing the tokenizer with similar regex as found in the measured entity annotation implementation.

NAMED ENTITY (NE) RECOGNITION

One vexing issue during testing was the extreme case sensitivity the detector has which can impact words in all caps as a modern convention for heightened emphasis.

The performance of the gazetteer lends inspiration to a layered detection of named entities where the gazetteer works in conjunction with the named entity detector.

The NLTK supplied: ne_chunk is presented in a tree structure, which can be cumbersome to traverse to extract relevant information. There should be an option to present the data in alternative data structures, like a nested list, or a dataframe for easier traversal and implementation. To circumvent this behavior, we implemented a helper method explicitly for navigating the tree structure.

MEASURED ENTITY DETECTION

Due to the strict requirements, the measured entity detector performed better than the named entity detector but depended heavily on the POS tagger. The nature of the comparison prevented the measured entities from being integrated into the csv output like other processing steps.

REFERENCES

1. *View nltk on Snyk Open Source Advisor*. (n.d.). Snyk Advisor. https://snyk.io/advisor/python/nltk
2. *NLTK :: Natural Language Toolkit*. (n.d.). https://www.nltk.org/
3. *Getting Started — pint 0.22 documentation*. (n.d.). https://pint.readthedocs.io/en/stable/getting/index.html
4. *Getting started — pandas 2.1.1 documentation*. (n.d.). https://pandas.pydata.org/docs/getting_started/index.html
5. *GeoNames*. (n.d.). https://www.geonames.org/
6. *7. Extracting Information from Text*. (n.d.). https://www.nltk.org/book/ch07.html
7. *5. Categorizing and tagging words*. (n.d.). https://www.nltk.org/book/ch05.html