Thursday, December 3, 2020

Course Project: Final Report

W. Daniel Child

Notes on Setup

In addition to the code provided under the project itself, you wish to run this code on your own computer you will need to install Stanza, chromedriver, search engine parser, and BeautifulSoup.

Installing chromedriver. You must also have the correct version of chromedriver according to your OS. I can only speak for my own system, but if using this code on a Mac with Catalina (MacOS 10.15.x), then it is critical that you have the appropriate version of chromedriver installed in the folder where the Python code is running. Also, because Catalina has strict developer recognition requirements, you may have to give the driver permission to run. Under System Preferences / Security, you must explicitly blessing chromedriver. For details https://stackoverflow.com/questions/60362018/macos-catalinav-10-15-3-error-chromedriver-cannot-be-opened-because-the-de for details. The version of chromedriver I am uploading to github may not be the one you need for your own computer.

Installing Stanza. Theoretically, installation is simple enough using pip. Simply employ

```
>>> pip install stanza
```

Although Stanza can be installed using Anaconda for Python 3.7 or earlier, conda installs will not work if using Python 3.8. (I inadvertently tried running stanza from Python 3.7 within conda after a pip install and promptly received missing resource errors. So if you use Anaconda regularly and want to use Stanza, you should install it as follows:

```
>>> conda install -c stanfordnlp stanza
```

Taking this step promptly resolves error messages involving missing resources.

Search Engine Parser. It appears that Google has discontinued *free* options for using their search API. Another option has been developed for Python, however, and it seems to work perfectly well. So I am using that as my starting point. So I installed the search engine parser as follows:

```
>>> pip install search-engine-parser
```

It turns out that this search-engine-parser supports different search engines, including Google, Yahoo, and (allegedly) Bing. However, during tests, Bing failed miserably, flagging searches as possibly illegal, and so I didn't use it. I did compare Yahoo and Google for the query '黒文字' (literally "black characters" but also the name of a flower, in Japanese) and found that they had very different results.

Project Objective

As delineated in the project proposal, my goal has been to develop a suite of small application that make it easier to search for rare Japanese patent terms and to show those terms in context. With rare terms like this, search engines often return irrelevant pages, and one has to hunt for the immediate context in question. From a translator's perspective, it would be extremely helpful to have concise reports on term contexts to understand how the terms are used, with small text extracts showing exactly where the expression is used.

Why Is Finding Terms a Problem

In Japanese patents, one frequently encounters obscure terms that are not readily found in dictionaries. In addition, because Japanese does not partition words by spaces, parsers routinely make mistakes in determining word boundaries. In the case of rare terms, the tendency is to break a longer term into something shorter. As a result, search engines like Google frequently return text retrieval results that do not in fact contain the full term in question. For example, if a term had characters ABCDE, and if BC can be treated as a word, and DE can be treated as a word, results may be based on matches for BC, DE, or even BCDE, but not the full expression ABCDE. Depending on the actual term in question this may or may not be helpful.

Because so many pages can be unhelpful or off topic, the translator may have to browse a large number of irrelevant pages before finding one that contains the term of interest. The purpose of this project is to make it easy to locate pages that actually contain the full term in question (ABCDE) and to then use Stanza's NLP parser to analyze sentences that actually contain the data.

Software Structure

There are three main pieces of code:

```
analyzer.py
scraper.py
search.py
```

Now, search.py is the main one used to run the software, whereas scraper.py and analyzer.py implement classes that respectively perform the scraping and analysis methods.

Because the analyzer and scraper codes employ an Analyzer and Scraper class, and many of the interior functions are encapsulated within the class, running the code is extremely simple.

Within the search, you need to simply choose three things:
(1) the expression to search (e.g. 黒文字)
(2) whether you want to use Google or Yahoo engines for the search
(3) the maximum depth of the search

The expression here is should be Japanese. I have provided a number of expressions for testing purposes directly into the code. To run one program, you simply need to copy the correct variable names into the class constructor:

```
scraper = Scraper(target = <target variable name here>,
                  max_depth = <set depth value here>,
                  engine = “google”, “yahoo”, or “both” (default)>)
```

Search engine parser returns a page of results that one would see on a normal browser. Because I am using a headless browser, the equivalent is returned as “one page” level of depth, which will contain 8 - 10 urls.

Operation

Once these settings have been made, the line

```
        scraper.run_search()
```

will run code to search each page for the target expression. There is a lot that goes on behind the scenes. The biggest issue was cleaning up the "soup" produced by BeautifulSoup. Most sources recommend "decompose" but it turned out to not be that helpful. A more productive approach was to regular expressions to find if the expression was found, and to indicate which element contained the target expression, and then return that element's text.

The results will indicate whether a given url has matches, as follows:

```
Url 0 has matches for 出来形
a	出来形

Url 1 has no matches for 出来形
Url 2 has matches for 出来形
h3	2014.12.02
出来形と出来高の違い

p	土木・建築用語に、「出来形」と「出来高」がありますが、この2つはとてもよく似た言葉ですが、意味は異なります。
```

Syntactic Analysis

After the max depth has been reached and all urls have been explored, then operations are conducted by the analyzer, whose main function is to utilize Stanza functions. The analyzer obtains the data that was stored by the scraper under its "results" instance variable and applies Stanza parsing and analysis to that data.

This analysis shows that the word in question is similar to another but has a different meaning.

```
2014.12.02	CD	NUM
出来	VV	VERB
形	NN	NOUN
と	PS	ADP
出来高	NN	NOUN
の	PN	ADP
違い	NN	NOUN
土木	NN	NOUN
```

```
・	SYM	SYM
建築	NN	NOUN
用語	NN	NOUN
に	PS	ADP
、	SYM	PUNCT
「	SYM	PUNCT
出来形		NN	NOUN
」	SYM	PUNCT
と	PQ	ADP
「	SYM	PUNCT
出来高		NN	NOUN
」	SYM	PUNCT
が	PS	ADP
あり	VV	VERB
ます	AV	AUX
が	PC	SCONJ
、	SYM	PUNCT
この	JR	DET
2	CD	NUM
つ	XSC	NOUN
は	PK	ADP
とても	RB	ADV
よく	RB	ADV
似	VV	VERB
た	AV	AUX
言葉	NN	NOUN
です	AV	AUX
が	PC	SCONJ
、	SYM	PUNCT
意味	NN	NOUN
は	PK	ADP
異なり	VV	VERB
ます	AV	AUX
。	SYM	PUNCT
```

Parts of speech are indicated in the analysis. Because the default printout is rather long, I have used a more tabular approach.

At the end, the extracted text and Stanza analysis are written to file, and if all goes well, the program terminates.

Performance Analysis

Being able to get clean text containing the target expression is a huge win. The analysis from Stanza (shown above on this page) is helpful as well.

Under the hood, the basic activities undertaken by the scraper instance performs the following:

- for each url in the page
- chromedriver gets the page
- BeautifulSOup creates a soup object
- the soup object is cleaned

Occasional Pitfalls

Twenty hours is not enough time to develop a truly robust parsing function. If pages turn out to no longer exist or if there is an anti-scraping function embedded in the page, then I have tried to catch the error within the Scraper code. Unfortunately, the error seems to sometimes emanate from within chromedriver itself, and theres not much I can do about that.

```
Traceback (most recent call last):
 File "search.py", line 29, in <module>
  scraper.run_search()
 File "/Users/danielchild/Desktop/TIS/PROJECT/CourseProject/ProjectCode/scraper.py", line
106, in run_search
  self.process_page_set(self.current_page_set)
 File "/Users/danielchild/Desktop/TIS/PROJECT/CourseProject/ProjectCode/scraper.py", line
99, in process_page_set
  self.get_soup(retrieved_urls[url_num])
 File "/Users/danielchild/Desktop/TIS/PROJECT/CourseProject/ProjectCode/scraper.py", line
64, in get_soup
  self.driver.get(url)
(ENTERING GOOGLE CODE BELOW)
 File "/Users/danielchild/opt/anaconda3/lib/python3.7/site-packages/selenium/webdriver/
remote/webdriver.py", line 333, in get
  self.execute(Command.GET, {'url': url})
 File "/Users/danielchild/opt/anaconda3/lib/python3.7/site-packages/selenium/webdriver/
remote/webdriver.py", line 321, in execute
  self.error_handler.check_response(response)
 File "/Users/danielchild/opt/anaconda3/lib/python3.7/site-packages/selenium/webdriver/
remote/errorhandler.py", line 242, in check_response
  raise exception_class(message, screen, stacktrace)
```

```
selenium.common.exceptions.WebDriverException: Message: unknown error:
net::ERR_CONNECTION_CLOSED
 (Session info: headless chrome=87.0.4280.67)
```

To avoid such errors, I wrapped the get(url) function in a try-except block, and this usually solves the problem.

It turns out that Stanford's Stanza wrapper for CoreNLP is also buggy. Consider this error:

```
Traceback (most recent call last):
 File "search.py", line 44, in <module>
  analyzer.analyze_data()
 File "/Users/danielchild/Desktop/TIS/PROJECT/CourseProject/ProjectCode/analyzer.py", line
24, in analyze_data
  doc = self.nlp(d)
(ENTERING STANZA CODE BELOW)
 File "/Users/danielchild/opt/anaconda3/lib/python3.7/site-packages/stanza/pipeline/core.py",
line 166, in __call__
  doc = self.process(doc)
 File "/Users/danielchild/opt/anaconda3/lib/python3.7/site-packages/stanza/pipeline/core.py",
line 160, in process
  doc = self.processors[processor_name].process(doc)
 File "/Users/danielchild/opt/anaconda3/lib/python3.7/site-packages/stanza/pipeline/
pos_processor.py", line 30, in process
  sort_during_eval=True)
 File "/Users/danielchild/opt/anaconda3/lib/python3.7/site-packages/stanza/models/pos/
data.py", line 48, in __init__
  self.data = self.chunk_batches(data)
 File "/Users/danielchild/opt/anaconda3/lib/python3.7/site-packages/stanza/models/pos/
data.py", line 150, in chunk_batches
  (data, ), self.data_orig_idx = sort_all([data], [len(x[0]) for x in data])
 File "/Users/danielchild/opt/anaconda3/lib/python3.7/site-packages/stanza/models/
common/data.py", line 39, in sort_all
  return sorted_all[2:], sorted_all[1]
IndexError: list index out of range
```

To avoid such errors, I similarly employed a try-except block. Similar errors can appear when writing to file.

```
*** WRITING DATA TO FILE:  出来高 analysis.txt  ***
```

```
Traceback (most recent call last):
  File "search.py", line 45, in <module>
    analyzer.write_analysis()
  File "/Users/danielchild/Desktop/TIS/PROJECT/CourseProject/ProjectCode/analyzer.py", line
41, in write_analysis
    doc = self.nlp(d)
(ENTERING STANZA CODE BELOW)
  File "/Users/danielchild/opt/anaconda3/lib/python3.7/site-packages/stanza/pipeline/core.py",
line 166, in __call__
    doc = self.process(doc)
  File "/Users/danielchild/opt/anaconda3/lib/python3.7/site-packages/stanza/pipeline/core.py",
line 160, in process
    doc = self.processors[processor_name].process(doc)
  File "/Users/danielchild/opt/anaconda3/lib/python3.7/site-packages/stanza/pipeline/
pos_processor.py", line 30, in process
    sort_during_eval=True)
  File "/Users/danielchild/opt/anaconda3/lib/python3.7/site-packages/stanza/models/pos/
data.py", line 48, in __init__
    self.data = self.chunk_batches(data)
  File "/Users/danielchild/opt/anaconda3/lib/python3.7/site-packages/stanza/models/pos/
data.py", line 150, in chunk_batches
    (data, ), self.data_orig_idx = sort_all([data], [len(x[0]) for x in data])
  File "/Users/danielchild/opt/anaconda3/lib/python3.7/site-packages/stanza/models/
common/data.py", line 39, in sort_all
    return sorted_all[2:], sorted_all[1]
IndexError: list index out of range
```

I avoid such issues using the same approach.

Assessment of the Analysis

Being able to quickly isolate instances where a particular expression is used, while bypassing web pages that contain portions of the expression but not the entire expression, is highly useful. The Stanza analysis is somewhat haphazard, as sometimes a portion of the expression is interpreted as a verb, and sometimes as a noun, even though the usage is exactly the same.

Future Functions

I have already gone well over 30 hours on this project, but if I had more time I would add functions to check for definitions specifically, as well for English translations of the terms.