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, the purpose of this project is to develop an application that can help Japanese translators search for contextual information on rare Japanese patent terms and expressions, and to show those terms and expressions in the context where they are used. When one searches for rare patent terms, search engines routinely return irrelevant pages, and one has to hunt for relevant information out of a large number of retrieved urls. From a translator's perspective, it would be extremely helpful to have concise term reports that contain extracts showing exactly where the expression is used. As a bonus, this project also includes automatically generated syntactic analyses of the extracts that are retrieved.

Why Is Finding Terms Problematic

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, imagine you want to find an expression that contained the characters ABCDE. As it turns out, if BC is considered to be a word, and if DE is considered to be a word, then a typical browser search is liable to return urls that match BC, DE, or even BCDE, but not the full expression ABCDE. Such returned urls are often unhelpful.

Because so many pages end up being off topic, the translator is frequently forced 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 passages that actually contain the data.

Application Structure

There are three main pieces of code:

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

Now, search.py is basically a main function where one can adjust parameters before running a search and conducting an analysis. The other files — namely, scraper.py and analyzer.py, — implement classes that respectively perform the scraping and analysis methods.

Running the Application

The analyzer and scraper files serve to implement Analyzer and Scraper classes that I created. Creating classes in this way made it possible for me to encapsulate nearly all of the functionality that is needed for this project. This makes running the code is extremely simple.

To run the application, you simply need to set three parameters or variables within search.py:

(1) the expression to search (e.g. target = 黒文字)
(2) whether you want to use a Google or Yahoo engine for the search
(3) the maximum depth of the search

The expression here 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)>)
```

The search engine parser returns a page of results that one would see on a normal browser. Because I am using a headless browser, one page of results corresponds to approximately 8 - 10 urls. The max_depth variable indicates how many pages of urls you want to explore (a depth of 5 would correspond to approximately 50 urls). To avoid long runtimes during project exploration, you may want to keep this to 1.

Operation

Once these parameters have been set, 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. Scraper operations can be summarized as follows:

```
until max depth is reached or while no matches are found…
    process a set of pages by
        obtaining the html (“soup”) for each page
        searching for the expression on each page
        collecting text passages where the expression is found
```

The biggest issue here 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 then the default approach used by Stanza

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

Project Evaluation

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. The project has been successful in that it does what it is supposed to do.

Avoiding Occasional Pitfalls

• *Chromedriver Errors*

Twenty hours is not enough time to develop a truly robust parsing function. Things sometimes go wrong when scraping. The page may no longer exist, or it may have an anti-scraping function embedded in it. Such issues seem to throw an error within chromedriver itself. Here is an example of what can happen:

```
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)
```

There is not much I can do about errors internal to chromedriver, so to handle the issue more gracefully, I wrapped the driver's get(url) function in a try-except block. If html retrieval fails, then the program will move on to the next url. This seems to have solved the problem, though many more hours of testing would be needed to make sure every conceivable problem could be anticipated.

- *Stanza Errors*

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
```

This seems to have occurred because Stanza got confused when parsing a very long text passage. To circumvent such errors, I similarly employed a try-except block.

• *File Write Errors*

Because the data that is written to file for future analysis is based on Stanza, analogous 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
```

Once again, to make sure that the program exits gracefully, I used a try-except block.

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 erratic: given two different text passages, a portion of the expression may sometimes be interpreted as a verb, and elsewhere as a noun, even though the usage is exactly the same in both contexts. Clearly more work needs to be done on Stanza's end.

Future Avenues of Development

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. I would also like to explore other Japanese morphological analyzers to see if they perform better. Still, in its current form, this project does what it is supposed to, and should provide a solid foundation for people who want to develop a more robust and function-filled application.