CS 432: Homework Number Three

Due on Febuary 14, 2018 at 4:20 PM

Alexander Nwala

Tim Bruce

Problem 1

Download the 1000 URIs from assignment #2. "curl", "wget", or "lynx" are all good candidate programs to use. We want just the raw HTML, not the images, stylesheets, etc.
Now use a tool to remove (most) of the HTML markup for all 1000 HTML documents.

Solution
This part was fairly easy to do by reusing older code. First, the links were pulled line by line. Then for each link, the html was pulled using the exact same function that was used in the first homework.

Listing 1: Code to pull get html code from links.

```
# Gets the html code for the webpage. If the link is not claimed by any server or
#   lacks a top level domain, it will raise an error.
def get_html(url):
    try:
        temp_html_reader = ul.urlopen(url)                 # Opens the page.
    except ule.URLError:
        return None

    return temp_html_reader.read()                         # Returns the html code.
```

Next, the raw html code is turned in to (hopefully) mostly plain text. At time of writing this is done using beatuifulsoup. Beautifulsoup is used because of difficulties with boilerpipe. Finally, the link, raw html, and the parsed text is put into a dictionary entry together in the manner shown below, and then everything is put into a JSON file.

Listing 2: Code to put the html code into a dictionary.

```
for key in html_dump.keys():                            # For everything in html dump
        raw_html = html_dump[key]                          # The raw html.
        soup = BeautifulSoup(raw_html, 'html.parser')      # Create html parser.
        text = soup.get_text()                             # The parsed html.

        linkdict = dict()                                  # This is where the clean
    formatted data goes.
        linkdict['raw_html'] = str(raw_html)               # Put in the raw html.
        linkdict['clean_html'] = str(text)                 # Put in the clean text.
        linkdict['link'] = str(key)                        # Put in the link for output.

        to_save_link_text[hash(key)] = linkdict
```

Problem 2

Choose a query term (e.g., "shadow") that is not a stop word (see week 5 slides) and not HTML markup from step 1 (e.g., "http") that matches at least 10 documents (hint: use "grep" on the processed files). If the term is present in more than 10 documents, choose any 10 from your list. (If you do not end up with a list of 10 URIs, you've done something wrong).
As per the example in the week 5 slides, compute TFIDF values for the term in each of the 10 documents and create a table with the TF, IDF, and TFIDF values, as well as the corresponding URIs. The URIs will be ranked in decreasing order by TFIDF values. For example:
Table 1. 10 Hits for the term "shadow", ranked by TFIDF.

Problem 2 continued on next page. . .

```
TFIDF TF IDF URI
-------------
0.150 0.014 10.680 http://foo.com/
0.044 0.008 10.680 http://bar.com/
```

You can use Google or Bing for the DF estimation. To count the number of words in the processed document (i.e., the deonminator for TF), you can use "wc":

```
% wc -w www.cnn.com.processed
2370 www.cnn.com.processed
```

It won't be completely accurate, but it will be probably be consistently inaccurate across all files. You can use more accurate methods if you'd like, just explain how you did it.

Solution

I was able to search the document for a word and get a search term count using the string.split method in Python. This returns the indices of instances of a matching string. Using this I was able to calculate the TF of a term in the following way.

Listing 3: Code to calculate TF and TF-IDF.

```
instances = clean_html.count(searchTerm)        # Get count of of instances of the search term in the text.
words = clean_html.split(' ')
word_count = len(words)                         # Get word count of clean text.
tf = instances/word_count                       # Calculate tf value.
tfidf = (instances/word_count) * IDF            # Calculate tf-idf value.
```

The IDF value is a little bit more controversial in my mind. The assignment states that we can use the Google search results to calculate the IDF value, but this number is useless without the number of total pages that Google can link to. The "how Google search works" page claims that Google has over 130 trillion possible results, so that is the number I went with. It would be easy to just use the IDF from the sample data, and I'm not sure why the assignment does not just do that.

From this point it was fairly simple to write a function that picked ten of them at random that matched the search result and to write out the data about them. As an aside, I also made a sorter to get the top 12 values. I picked the number 12 because one of them was a repeat with a source tag and because one of them was following a redirect because the page had become a 404 since I first retrieved it. The results shown are from this sorting.

Listing 4: Output from the program. Links are long, sorry about the text wrapping.

```
TFIDF    TF          IDF          URI
-----    --          ---          ---
7.66    0.5         15.3        http://www.topnewstrends.tk/2019/02/nasa-calls-time-on-silent-opportunity.html
0.356    0.0232       15.3        https://abcnews.go.com/US/nasa-ends-mission-mars-rover-opportunity-15-years/story?id=61046744
0.356    0.0232       15.3        https://abcnews.go.com/US/nasa-ends-mission-mars-rover-opportunity-15-years/story?id=61046744&cid=social_twitter_abcn
0.288    0.0188       15.3        https://www.express.co.uk/news/science/1087755/Nasa-opportunity-mars-rover-last-message-opportunity-last-words
0.214    0.014       15.3        https://www.cbc.ca/news/technology/mars-opportunity-rover-dead-1.5018038
0.214    0.0139       15.3        https://www.nationalgeographic.com/science/2019/02/nasa-mars-rover-opportunity-dead-what-it-gave-humankind/
```

Problem 2 continued on next page. . .

```
0.158    0.0103         15.3       https://www.express.co.uk/news/science/1087904/nasa-
    opportunity-dead-last-words-mars-rover-dies-twitter
0.151    0.00984         15.3       https://motherboard.vice.com/en_us/article/pank98/nasas-
    mars-opportunity-rover-is-likely-dead
0.15    0.00981         15.3       https://www.express.co.uk/news/science/1088018/nasa-
    opportunity-rover-elon-musk-spacex-rescue-mars-rover
0.124    0.00812         15.3       https://www.space.com/mars-rover-opportunity-declared-
    dead.html
0.123    0.00805         15.3       https://www.space.com/mars-rovers-opportunity-spirit-
    change-exploration.html?
```

Problem 3

Now rank the same 10 URIs from question 2, but this time by their PageRank. Use any of the free PR estimaters on the web, such as:

`http://pr.eyedomain.com/`
`http://www.prchecker.info/check_page_rank.php`
`http://www.seocentro.com/tools/search-engines/pagerank.html`
`http://www.checkpagerank.net/`

If you use these tools, you'll have to do so by hand (they have anti-bot captchas), but there are only 10 to do. Normalize the values they give you to be from 0 to 1.0. Use the same tool on all 10 (again, consistency is more important than accuracy). Also note that these tools typically report on the domain rather than the page, so it's not entirely accurate.
Create a table similar to Table 1:
Table 2. 10 hits for the term "shadow", ranked by PageRank.

```
PageRank    URI
--------    ---
0.9         http://bar.com/
0.5         http://foo.com/
```

Briefly compare and contrast the rankings produced in questions 2 and 3.

Solution
The annoying thing about this problem is that these PageRank websites only really work for the homepage, so I really feel that their input is completely worthless, because most of my links are articles and my keyword is kinda specific.
Once I got the PageRank from http://pr.eyedomain.com/ because it gives more significant figures, I got the following output:

Listing 5: PageRank of top 10 TF-IDF values in order of TF-IDF rank.

```
PageRank    URI
--------    ---
9.2         https://abcnews.go.com/
8.5         https://express.co.uk/
8.7         https://www.cbc.ca/
8.9         https://www.nationalgeographic.com/
8.5         https://www.express.co.uk/
8.5         https://motherboard.vice.com/
```

Problem 3 continued on next page. . .

```
8.5         https://www.express.co.uk/
8.5         https://www.space.com/
8.5         https://www.space.com/
8.6         https://www.dictionary.com/
```

While these do trend downwards, the information gained here is anecdotal at best due to our low sample size. abcnews.go.com is on top of both rankings. This may have to do with ABC really understanding how the internet ranking system works down to a fundamental level in their article writing. Because they are targeting the search term ”Opportunity” and say it a lot, it appears at the top of the TF-IDF list. This would mean its fundamental understanding of this has made it a highly linked-to page. The same can be said down the board. The repeat top-level domains are all clustered together in TF-IDF due to how they write articles, and it may show how well they do this in their page rankings.

Problem 4

Compute the Kendall Tau_b score for both lists (use ”b” because there will likely be tie values in the rankings). Report both the Tau value and the ”p” value.

Solution

For this question, I was able to use the SciPy library, which has a Kendall Tau_b function in it, that made this really easy. I hand inputted the values in arrays like so:

Listing 6: Kendall Tau Calculator for TF-IDF and PageRank.

```
tfidf_values = [0.356, 0.288, 0.214, 0.214, 0.158, 0.151, 0.15, 0.124, 0.123, 0.114]
page_rank = [9.2, 8.5, 8.7, 8.9, 8.5, 8.5, 8.5, 8.5, 8.5, 8.6]

print("FOR TFIDF AND PAGERANK")
print(kendallTauCalculation(tfidf_values, page_rank))
```

This method produces the following output:

Listing 7: Kendall Tau_b for the TF-IDF and PageRank scores of ten web pages.

```
FOR TFIDF AND PAGERANK
KendalltauResult(correlation=0.3578132236660672, pvalue=0.18457255283988294)
```

This low correlation is probably once again due to the lack of relatedness between the two topics. One of the numbers is an iffy aggregate for the top-level domain, and the other is a search ranking for a specific term using a low sized and biased sample set of data. The correlation is probably due to a relationship between the page ranking and how these websites present their data.

Problem 5

Compute a ranking for the 10 URIs from Q2 using Alexa information (see week 4 slides). Compute the correlation (as per Q4) for all pairs of combinations for TFIDF, PR, and Alexa.

Solution

I was able to retrieve Alexa scores from January 31st, 2019 from https://www.rank2traffic.com, which made it a lot easier than from the official source. Doing this, I was able to reuse some of the code from the last part, you will note the similarities:

Listing 8: Kendall Tau Calculator for TF-IDF, PageRank, and Alexa.

Problem 5 continued on next page. . .

```
tfidf_values = [0.356, 0.288, 0.214, 0.214, 0.158, 0.151, 0.15, 0.124, 0.123, 0.114]
page_rank = [9.2, 8.5, 8.7, 8.9, 8.5, 8.5, 8.5, 8.5, 8.5, 8.6]
alexa = [1721, 814, 934, 1462, 814, 162, 814, 4914, 4914, 597]  # https://www.
    rank2traffic.com/
print("FOR TFIDF AND PAGERANK")
print(kendallTauCalculation(tfidf_values, page_rank))

print("FOR TFIDF AND alexa")
print(kendallTauCalculation(tfidf_values, alexa))

print("FOR alexa AND PAGERANK")
print(kendallTauCalculation(alexa, page_rank))
```

The output from this code was:

Listing 9: Kendall Tau_b for the TF-IDF and PageRank scores of ten web pages.

```
FOR TFIDF AND PAGERANK
KendalltauResult(correlation=0.3578132236660672, pvalue=0.18457255283988294)
FOR TFIDF AND alexa
KendalltauResult(correlation=0.09417632186960223, pvalue=0.714342227424899)
FOR alexa AND PAGERANK
KendalltauResult(correlation=0.22810637940488043, pvalue=0.40790488230961974)
```

This is actually quite interesting! I used the top-level domains for the Alexa score, and the PageRank-tf-idf correlation still had more similarity than the PageRank-Alexa. This indicates to me that the PageRank and Alexa scores are fundamentally different, and TF-IDF is closer to PageRank... or our sample size is too low for useful analysis.