Web Science: Assignment #3

Alexander Nwala

Mohd. Nauman Siddique

Thursday, February 28, 2019

Contents

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. from the command line:

```
% curl http://www.cnn.com/ > www.cnn.com
% wget -O www.cnn.com http://www.cnn.com/
% lynx -source http://www.cnn.com/ > www.cnn.com
```

www.cnn.com is just an example output file name, keep in mind that the shell will not like some of the characters that can occur in URIs (e.g., "?", "&"). You might want to hash the URIs to associate them with their respective filename, like:

```
% echo -n "http://www.cs.odu.edu/show_features.shtml?72" | md5 41
    d5f125d13b4bb554e6e31b6b591eeb
```

("md5sum" on some machines; note the "-n" in echo – this removes the trailing newline.)
Now use a tool to remove (most) of the HTML markup for all 1000 HTML documents. "python-boilerpipe" will do a fair job see http://ws-dl.blogspot.com/2017/03/2017-03-20-survey-of-5-boilerplate.html:

```
    from boilerpipe.extract import Extractor
    extractor = Extractor(extractor='ArticleExtractor', html=html)
    extractor.getText()
```

Keep both files for each URI (i.e., raw HTML and processed). Upload both sets of files to your github account.

SOLUTION

The solution to the problem is as below:

1. **Download Raw HTML**: The urls are read from a file name *Urls_Uniq_Expanded.txt* and fed to the *requests.get* method to fetch the response. The response is then saved into a file which has a name of the *sha26* of the url.

```
'''
Function to fetch URLs
'''


def get_raw_html():
    file_urls = open("Urls_Uniq_Expanded.txt", "r")
    file_urls_downloaded = open("Urls_downloaded.txt", "w")
    count = 0
    for line in file_urls:
        url = line.rstrip().split("|||")[0]
        url_digest = hashlib.sha3_256(url.encode()).hexdigest()
        try:
            response = requests.get(url)
            if response.status_code == 200 and count < 1000:
                count += 1
                file_urls_downloaded.write(url + ": " + url_digest + "\n")
                file_raw_html = open("rawHTML/" + url_digest, "w")
```

Problem 1 continued on next page...

```
                file_raw_html.write(response.text)
                file_raw_html.close()
            elif count == 1000:
                break
        except Exception as e:
            print(e)
    file_urls_downloaded.close()
    file_urls.close()
```

2. **Boilerpipe**: I have used the links from the urls to extract the text using *python-boilerpipe* library. I tried extracting the text from raw html but the documentation for boilerpipe library did not provide me with enough insights onto how to use the different arguments.

```
'''
Function boilerpipe code
'''


def boilerpipe_code():
    files_urls = open("Urls_Uniq_Expanded.txt", "r")
    for line in files_urls:
        url = line.rstrip().split("|||")[0]
        try:
            extractor = Extractor(extractor='ArticleExtractor', url= url)
            url_digest = hashlib.sha256(url.encode()).hexdigest()
            file_open = open("boilerHTML/" + url_digest, "w")
            response = extractor.getText()
            file_open.write(response)
            file_open.close()
        except Exception as e:
            print(e)
```

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.

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. Don't forget the log base 2 for IDF, and mind your significant digits!
https://en.wikipedia.org/wiki/Significant_figures#Rounding_and_decimal_places

SOLUTION

```
'''
Find term frequency and inverse term frequency from Corpus
'''


def find_tf_idf():

    words_list = ["Twitter", "Facebook", "Sports", "America", "Trump", "President", "
        Friends", "Tweet", "Post", "Senate"]
    list_outputs = []
    for i in range(0, len(words_list)):
        list_outputs.append([])
    for word in words_list:
        for files in os.listdir("boilerHTML"):
            file_open = open("boilerHTML/" + files, "r")
            count = 0
            count_total = 0
            for line in file_open:
                words_in_line = line.split(" ")
                for word_in_line in words_in_line:
                    count_total += 1
                    if word.lower() == word_in_line.lower():
                        count += 1
            if count > 0:
                list_temp = []
                list_temp.append(files)
                list_temp.append(str(count / count_total))
                list_outputs[words_list.index(word)].append(list_temp)
            file_open.close()
    print(list_outputs)

    total_corpus = 0
    for files in os.listdir("boilerHTML"):
        total_corpus += 1

    for i in range(0, len(list_outputs)):
        for j in range(0, len(list_outputs[i])):
            idf = math.log((total_corpus/ len(list_outputs[i])), 2)
            list_outputs[i][j].append(str(idf))
            list_outputs[i][j].append(str(idf * float(list_outputs[i][j][1])))
```

```
    file_hashes = open("Urls_hash.txt", "r")
    list_hash = []
    list_url = []
    for line in file_hashes:
        line_split = line.split("|||")
        list_hash.append(line_split[0])
        list_url.append(line_split[1])
    file_hashes.close()

    for i in range(0, len(list_outputs)):
        file_words = open("wordFrequency/" + words_list[i] + ".txt", "w")
        for j in range(0, len(list_outputs[i])):
            for k in range(0, len(list_outputs[i][j])):
                if k == 0:
                    file_words.write(list_url[list_hash.index(list_outputs[i][j][k])].
                        rstrip() + "|||")
                else:
                    file_words.write(list_outputs[i][j][k] + "|||")
            file_words.write("\n")
        file_words.close()
```

The solution for this problem is outlined by the following steps:

Choosing Query Terms: I chose 10 query terms for this question but I will be using only 1 of them in my results. All the 10 query terms have more than 10 hits in my corpus of raw htmls.
Compute Term Frequency: The Term Frequency(TF) has been calculated by the formula mentioned in week 5 lecture.
$TF = \frac{Numberofoccurrencesfortheterminthedocument}{Totalnumberoftermsinthedocument}$
Compute Inverse Document Frequency: The Inverse Document Frequency(IDF) has been calculated using the formula from wek 5 lecture.
$IDF = \log_2 \frac{Numberofdocumentsinthecorpus}{Numberofdocumentswithoccurencesfortheterm}$
I solved the problem in two ways. In my first approach. I used the number of documents in the corpus to be my own list of documents while, in the second approach I just queried the term on google and used the number of results from it as my corpus size.
Compute TFIDF: TFIDF is the product of TF and IDF.

Using independece model we can calculate the size of collection.
$N = \frac{f_a * f_b}{f_a b}$
Query for dog on Google has 4,510,000,000 entries.
Query for cat on Google has 4,580,000,000 entries.
Query for dog cat on Google has 2,520,000,000 entries.

$N = 81,967,460,300,000,000,000$

Query term america on Google has 6,310,000,000 entries.

$IDF = 33.59669224192899$

TFIDF	TF	IDF	
0.005715823657356178	0.0008880994671403197	6.436017438183057	https://edition.cnn.com/2019/02/13/af
0.005194525777387455	0.0008071025020177562	6.436017438183057	
0.009722080722330901	0.0015105740181268882	6.436017438183057	https://bigleaguepolitics.com/ex
0.013325087863733038	0.002070393374741201	6.436017438183057	ht
0.003967951564847754	0.0006165228113440197	6.436017438183057	https://www.dcclot
0.003475171402906618	0.0005395568034557236	6.436017438183057	
0.02383710162290021	0.003703703703703704	6.436017438183057	
0.004962233953880537	0.0007710100231303007	6.436017438183057	
0.011039481026042979	0.0017152658662092624	6.436017438183057	
0.006026233556351177	0.0009363295880149813	6.436017438183057	
0.009220655355563118	0.0014326647564469914	6.436017438183057	
0.018441310711126237	0.0028653295128939827	6.436017438183057	

Table 1: IDF calculated from the document corpus dowloaded from the 1000 URLs on the query term: America

TFIDF	TF	IDF	
0.02983720448	0.0008880994671403197	33.59669224192899	https://edition.cnn.com/2019/02/13/africa/k
0.02711597437	0.0008071025020177562	33.59669224192899	
0.0507502904	0.0015105740181268882	33.59669224192899	https://bigleaguepolitics.com/exclusi
0.06955836903	0.002070393374741201	33.59669224192899	https://
0.02071312715	0.0006165228113440197	33.59669224192899	https://www.dcclothesli
0.01814076255	0.0005395568034557236	33.59669224192899	
0.1244321935	0.003703703703703704	33.59669224192899	ht
0.02590338646	0.0007710100231303007	33.59669224192899	
0.05762725942	0.0017152658662092624	33.59669224192899	https
0.03145757701	0.0009363295880149813	33.59669224192899	
0.04813279691	0.0014326647564469914	33.59669224192899	
0.09626559382	0.0028653295128939827	33.59669224192899	

Table 2: IDF calculated from the document corpus from google which has 6,280,000,000 records for query term: America

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.

Problem 3 continued on next page...

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

I used https://smallseotools.com/google-pagerank-checker/ link to find pagerank. All the page rank scores have been normalized by 10, as the website reports it in the sclae of 10.

Pagerank	
0.9	https://edition.cnn.com/2019/02/13/africa/kenya-rare-black-leopard-black-panther/
0.2	https://theintercept.com/2019/
0	https://bigleaguepolitics.com/exclusive-taqiyya-ilhan-omar-panders-to-transc
0	https://uslibertywire.com/fox-news-finds-skelet
0	https://www.dcclothesline.com/2019/02/14/mexican-scientists-fi
0	http://downwithtyranny.blogspot.com/20
0	https://qoinbook.com/news/hsbc-exec-says-
0.6	https://www.out.com/news-opinion/2019
0.9	https://www.youtube.com/watch?v=vMm5HfxNXY4&i
0	http://trenchtrenchtrench.com/feature
0.8	https://www.el
0	http://scrafinance.com/wall-stree

Table 3: Page rank for all the urls for query term America

The biggest highlight between the approach of TFIDF and pagerank is that TFIDF always provides us with a number even the the document might be very rare but in case of page rank it is calculated from domain name which creates a bias in the page rank and also provides a number 0 for very less popular pages. The results for all the URLs in TFIDF are very similar while calculating the pagerank ranks the pages with popular domain on the top. TFIDF is content based approach while page rank is popularity based approach for a URL. The need of the user determines which metric is better for them.

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.
See: http://stackoverflow.com/questions/2557863/measures-of-association-in-r-kendalls-tau-b-and-tau-c
http://en.wikipedia.org/wiki/Kendall$_t$au$_r$ank$_c$orrelation$_c$oefficient#Tau-b
http://en.wikipedia.org/wiki/Correlation$_a$nd$_d$ependence

SOLUTION

I added all the values manually calculated from previous problem to my function nd calulated Kendel Tau value.

tau = - 0.074
p_value = 0.757

```
'''
Calculate Kendel Tau
'''


def kendel_tau():
    # TF-IDF Values
    a = [0.03, 0.027, 0.051, 0.07, 0.021, 0.018, 0.124, 0.026, 0.058, 0.031, 0.048,
        0.096]
    # Page Rank
    b = [0.9, 0.2, 0, 0, 0, 0, 0,0.6, 0.9, 0, 0.8, 0]
    tau, p_value = stats.kendalltau(a, b)
    print(tau)
    print(p_value)
```