CS-432/532 Introduction to Web Science: Assignment #3

Due on Thursday, February 18, 2016

Dr. Michael L. Nelson

Plinio Vargas
pvargas@cs.odu.edu

Contents

List of Figures

Listings

List of Tables

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, like:

% echo -n “`http://www.cs.odu.edu/show_features.shtml?72`” | md5
41d5f125d13b4bb554e6e31b6591eeb

(“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. “lynx” will do a fair job:

% lynx -dump -force_html www.cnn.com > www.cnn.com.processed

Use another (better) tool if you know of one. Keep both files for each URI (i.e., raw HTML and processed).

1.1 Approach

As suggested in the problem requirement, all URIs were hashed using **MD5** function. A python application < **ProcessRawURI.py** > was developed to solve this problem. The application makes two (2) different iterations:

1. For each of the 1000 URIs, it gets a copy of the resources and saves the raw HTML content into a file: lines 29-43.
2. Each raw file is processed to remove all HMTL markup. A system call using the *lynx* command performs the HTML markup removal: 45-50.

Listing 1: ProcessRawURI.py

```
1 import os
2 import requests
3 import sys
4 from time import strftime, localtime, time
5 from subprocess import call
6 from hashlib import md5
7
```

Problem 1 continued on next page...

```python
__author__ = 'Plinio H. Vargas'
__date__ = 'Fri,  Feb 12, 2016 at 16:39:41'
__email__ = 'pvargas@cs.odu.edu'


PACKAGE_PARENT = '..'
SCRIPT_DIR = os.path.dirname(os.path.realpath(os.path.join(os.getcwd(), os.path.
    expanduser(__file__))))
sys.path.append(os.path.normpath(os.path.join(SCRIPT_DIR, PACKAGE_PARENT)))

uri_linkfile = '../a2/linkfiles.txt'
# uri_linkfile = 'rankedURI'
work_dir = 'URI-work-files/'
# work_dir = 'query-work-files/'
p_extension = '.processed'

# record running time
start = time()
print('Starting Time: %s' % strftime("%a,  %b %d, %Y at %H:%M:%S", localtime()))

# get resources from sample URIs
counter = 0
with open(uri_linkfile, 'r') as file:
    for uri in file:
        counter += 1
        uri = uri.strip()
        r = requests.get(uri)
        print(r.text)

        # create hash for URI
        out = md5(uri.encode()).hexdigest()

        # save into working_directory
        with open(work_dir + out, 'w') as raw_file:
            raw_file.write(r.text)

        print(counter, uri, out)

# remove HTML markup from processed files in processed dir
raw_files = [x for x in os.walk(work_dir)][0][2]
for raw_file in raw_files:
    # place query-work-files-processed file into processed directory
    with open(work_dir + raw_file + p_extension, 'w') as file:
        call(["lynx", "-dump", "-force_html", work_dir + raw_file], stdout=file)



print('\nEnd Time:  %s' % strftime("%a,  %b %d, %Y at %H:%M:%S", localtime()))
print('Execution Time: %.2f seconds' % (time()-start))
```

ProcessRawURI.py is attached to this document.

1.2 Solution

All raw and processed files were stored into directory ./URI-work-files

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	5.510	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 denominator 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.

Don't forget the log base 2 for IDF, and mind your significant digits!

2.1 Approach

2.1.1 The query term used to select our 10 documents was "football". A Python application **MyGrep.py** was developed to display the frequency count in which our term "football" appeared within each processed document.

Listing 2: MyGrep.py

```
# get processed resources
processed_files = [x for x in [x for x in os.walk(raw_dir)][0][2] if x[-10:]
    == '.processed']
for file in processed_files:
    with open(raw_dir + file, 'r', encoding='iso-8859-1') as p_file:
        words = re.findall('\w+', p_file.read().lower())
        cnt = Counter()
        for word in words:
            cnt[word] += 1

        print(file, sum(cnt.values()), cnt['football'])
```

Line 24 places into array **processed_files** all files in our working directory with extension "processed" in its filename. The remaining of above partial code iterates through each file and display a hashed filename, word counts and the frequency of our term. Below, tail end result of executing **MyGrep.py**:

Figure 1: Querying "football" on Processed Files

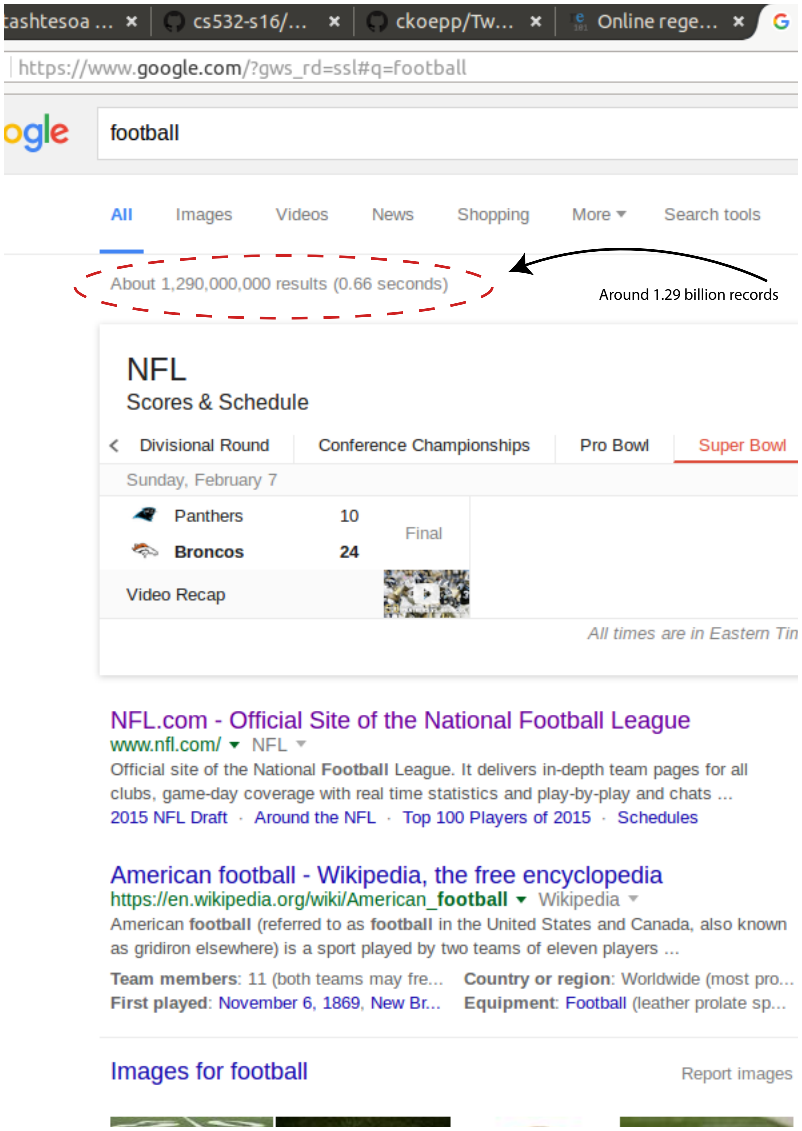
```
1deb1fd27da216c002a66a12afa7f240.processed 60 0
1b5b572e91fff302c72fede1953390cc.processed 2388 0
df44989f4b199500ecfbee2ac4700c2f.processed 2297 0
e884606825dc4bfc720ed8a179dde5f4.processed 4397 0
cbc4499880d666e056398b219cf1c800.processed 30 0
a4de2b1569c7fb9fbd37139ffb6e58c3.processed 4336 2
7b21588f35b9bd8ce5384ab0bd319e6e.processed 269 0
c2e06f65fbc5bc54c039c690ce313b03.processed 1093 0
351b4006448877b7bfe1acb3973517f7.processed 2568 0
bb2d991278d0815524179e1fa5374505.processed 3400 0
78b1e914582e37135be1676fc75cba99.processed 2477 0
fa3fae6bd45f974305540f33721e7f63.processed 24414 7
5b4058b62dbbf161521b63bb230455b1.processed 2463 0
1906a0db40b28d028b74ca84dcf1f70b.processed 2845 6
633babe8fd450769c5ad7a8eaeabb70f.processed 1153 0
13cafb82717bdd4c265deab2b1a8981b.processed 2902 0
e3b6392348b1695a89718bdece9e6e2b.processed 2687 28
590a9b6bdc592cb1e564212ecb99f97d.processed 2868 0
0d43d0c0656d65442001287b3bc38645.processed 2524 0
81b74aa80d54af18eb5688ca5636fee5.processed 22880 7
5ab0920417dd8c6b050d05eed99ea802.processed 2488 0
4a0db98e86714b95fb59af5af4cc6f5b.processed 2467 0
d4551fea2013123d359a36af07f3c6e9.processed 2434 0
b1a5b6946cc705b7425769d50afb1313.processed 7166 1
e7b5a7e8468b7c978f975cec008bc86d.processed 1392 0

End Time: Tue, Feb 16, 2016 at 05:59:22
Execution Time: 7.95 seconds
```

The query resulted in 1,000 line output corresponding to: filanme processed, number of words in the file and "football" term count. For example, in Figure 1 the yellow highlighted output fa3fae6bd45f974305540f33721e7f63.processed refers to a hashed URI collected from assignment 2, without any HTML markups, containing 24,414 words; 7 of them have are "football".

Then, a selection of 10 line items were made from the output making sure the term count was different than 0. The selection was saved in a file: **rankedURI**.

2.2.2 To find the Inverse Document Frequency(**IDF**) we considered Google, using 20B for our total docs in corpus, and obtaining 1.29B for our docs with the term from the Google search "football".

Figure 2: Google search for “football”



Result from google search of term “football” result in 1.29B records.

total docus in corpus =49.5B [2]
docs with term = 1.29B [3]
Then,

$$IDF(football) = \log_2\left(\frac{total\ docs\ in\ corpus}{docs\ with\ term}\right) \tag{1}$$

$$IDF(football) = \log_2\left(\frac{49.5B}{1.29B}\right) \tag{2}$$

$$IDF(football) \approx 5.262 \tag{3}$$

2.2 Solution

Finally, the solution to our problem is Table 1. A Python application < **CalculateRanking.py** > was developed to help calculate our 10 URIs' TFIDF score.

Listing 3: CalculateRanking.py

```
39 hash_table = {}
40 for line in (open(uri_linkfile, 'r')):
41     hash_table[md5(line.strip().encode()).hexdigest()] = line.strip()
42 
43 # get processed resources
44 rank_files = [line.strip() for line in (open(uri_rankfile, 'r'))]
45 for file in rank_files:
46     with open(work_dir + file.strip(), 'r', encoding='iso-8859-1') as p_file:
47         words = re.findall('\w+', p_file.read().lower())
48         cnt = Counter()
49         for word in words:
50             cnt[word] += 1
51 
52         TF = cnt[term] / sum(cnt.values())
53         rank_tuples.append((file, sum(cnt.values()), cnt[term], TF, IDF, TF * IDF))
54 
55         #print('File:%s info: Words-in-file: %d, term-count: %d, TF: %.4f, IDF: %.4f, TF-IDF: %.4f' %
56         #      (hash_table[file[:-10]], sum(cnt.values()), cnt[term], TF, IDF, TF * IDF))
57 print('\nTFIDF  TF     IDF    URI\n-----  --     ---    ---')
58 for row in sorted(rank_tuples, key=lambda tfidf: tfidf[5], reverse=True):
59     print('%.3f  %.3f  %.3f  %s\\\\' % (row[5], row[3], IDF, hash_table[row[0][:-10]]))
```

Lines 39-41 places our 1000 collected URIs into a hashed dictionary. Lines 43-53 iterates through our 10 selected processed files located in **rankedURI**, getting word count and TF for each file. Line 53 places into an array the calculated values for each URI: TF, IDF, TFIDF. **Note**: IDF was previously calculated. Since we have only one term, IDF is a constant.

Table 1: 10 Hits for the term "football", ranked by TFIDF

IDX	TFIDF	TF	IDF	URI
1	0.073	0.014	5.262	http://www.liverpoolecho.co.uk/sport/football/football-news/how-much-fenway-sports-group-10835670
2	0.055	0.010	5.262	http://www.thekeyplay.com/virginia-tech-football-recruiting/2016/02/11575/hokies-flip-odu-commitment-tyree-rodgers
3	0.051	0.010	5.262	http://www.andthevalleyshook.com/2016/1/31/10879578/lsu-football-recruiting-lindsey-scott-commits
4	0.047	0.009	5.262	http://www.theguardian.com/football/in-bed-with-maradona/2016/feb/03/english-footballers-europe-ashley-cole?CMP=share_btn_tw
5	0.039	0.007	5.262	http://sport360.com/article/other/us-sports/161389/where-to-watch-superbowl-50-in-dubai-and-abu-dhabi-carolina-panthers-vs-denver-broncos/
6	0.035	0.007	5.262	http://www.cbssports.com/nfl/eye-on-football/25473188/colts-owner-jim-irsay-says-he-asked-peyton-manning-to-retire-as-a-colt
7	0.025	0.005	5.262	http://www.si.com/nba/2016/02/04/lebron-james-cam-newton-super-bowl-50
8	0.025	0.005	5.262	http://www.pmnewsnigeria.com/2016/02/04/task-force-arrests-suspects-over-oil-pipe-blasts-in-niger-delta/
9	0.022	0.004	5.262	http://pilotonline.com/sports/college/old-dominion/football/odu-appears-to-have-solid-recruiting-class-as-it-battles/.../
10	0.019	0.004	5.262	http://www.si.com/nba/2016/02/04/stephen-curry-warriors-wizards-lebron-james-john-wall

The column labeled IDX has an invert relationship with TFIDF. The lower IDX, the higher is TFIDF. Similarly, the higher the TFIDF value the more significant is the relationship of our term "football" in comparison with other URIs in the table.

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://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 is only 10. 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).

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.

3.1 Approach

Table 2 was completed utilizing `http://www.seocentro.com/tools/search-engines/pagerank.html` free PR estimate:

3.2 Solution

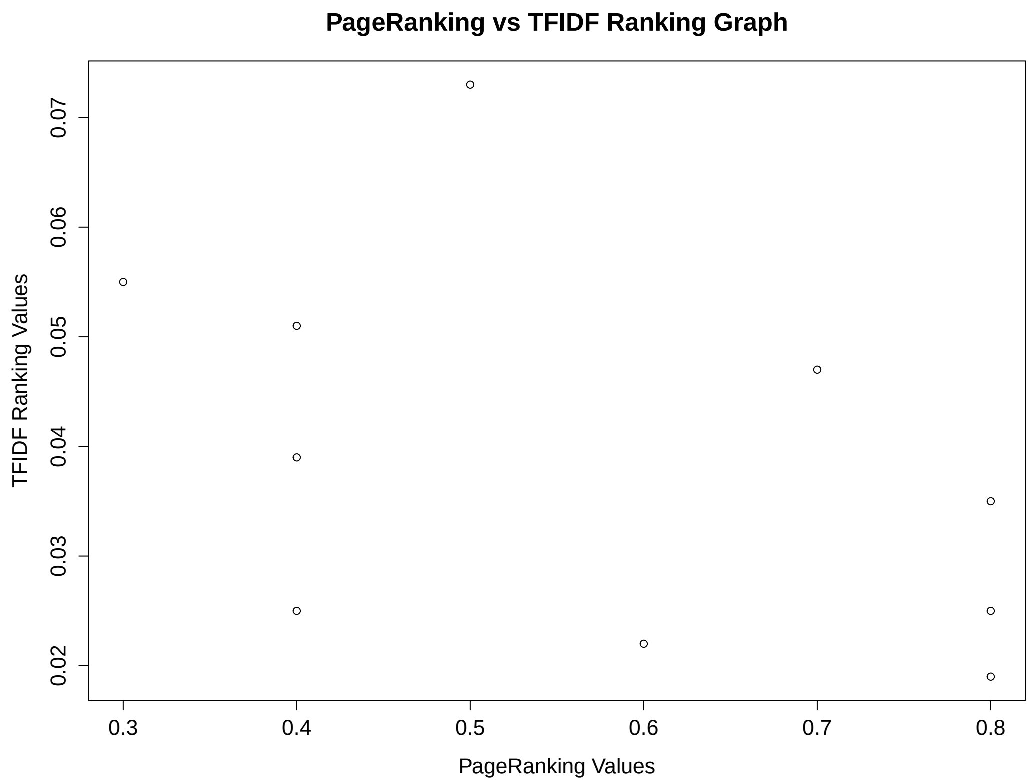
Table 2: 10 Hits for the term "football", ranked by PageRank

IDX	PageRank	URI
6	0.8	http:// www.cbssports.com/
7	0.8	http:// www.si.com/
10	0.8	http:// www.si.com/
4	0.7	http:// www.theguardian.com/
9	0.6	http:// pilotonline.com/
1	0.5	http:// www.liverpoolecho.co.uk/
3	0.4	http:// www.andthevalleyshook.com/
5	0.4	http:// sport360.com/
8	0.4	http:// www.pmnewsnigeria.com/
2	0.3	http://www.thekeyplay.com/

IDX from Table 2 has a sequence very different than Table 1. It seems there is a disparity between IDX and PageRank. Correlation between values in Table 1 and 2 are discussed on the next section.

3.3 Solutions 2 and 3 Comparison

Figure 3: TFIDF Ranking vs PageRanking Comparison



Plots are all over the graph. There is not a plot grouping or a pattern identifying a relationship between PageRanking and TFIDF Ranking. This is expected since TFIDF Ranking is based on TF and it is related to the URI. The information for PageRanking is not taking in consideration our term "football" and it is not ranking the URI, but rather its root domain.

Problem 4 Extra Credit

Compute the Kendall $\mathcal{T}_b$ score for both lists (use "b" because there will likely be tie values in the rankings). Report both the $\mathcal{T}$ value and the "p" value.

4.1 Manual Calculation

See:
`http://stackoverflow.com/questions/2557863/measures-of-association-in-r-kendalls-tau-b-and-tau-c`
`http://en.wikipedia.org/wiki/Kendall_tau_rank_correlation_coefficient#Tau-b`
`http://en.wikipedia.org/wiki/Correlation_and_dependence`

According to [1]

$$\tau_B = \frac{n_c - n_d}{\sqrt{(n_0 - n1)(n_0 - n_2)}} \tag{4}$$

$$\tag{5}$$

where

$$n_0 = n(n-1)/2 \tag{6}$$

$$n_1 = \sum_i t_i(t_i - 1)/2 \tag{7}$$

$$n_2 = \sum_j u_j(u_i - 1)/2 \tag{8}$$

n_c = Number of concordant pairs
n_d = Number of discordant pairs
t_i = Number of tied values in the i^{th} group of ties for the first quantity
u_j = Number of tied values in the j^{th} group of ties for the second quantity

Table 3: τ Values for PageRank vs TFIDF

PageRank	TFIDF	n_c	n_d	t_i	$t_i(t_i-1)/2$	u_j	$u_j(u_j-1)/2$	URI
0.8	0.035	4	5	2	1	0	0	http://www.cbssports.com/
0.8	0.025	2	6	1	0	1	0	http://www.si.com/
0.8	0.019	0	7	0	0	0	0	http://www.si.com/
0.7	0.047	3	3	0	0	0	0	http://www.theguardian.com/
0.6	0.022	0	5	0	0	0	0	http://pilotonline.com/
0.5	0.073	4	0	0	0	0	0	http://www.liverpoolecho.co.uk/
0.4	0.051	2	1	2	1	0	0	http://www.andthevalleyshook.com/
0.4	0.039	1	1	1	0	0	0	http://sport360.com/
0.4	0.025	0	1	0	0	0	0	http://www.pmnewsnigeria.com/
0.3	0.055							http://www.thekeyplay.com/
Total		16	29		2		0	

$n_0 = n(n-1)/2 = 10(9)/2 = 5(9) = 45$
$n_1 = \sum_i t_i(t_i-1)/2 = 2$ $\qquad n_2 = \sum_j u_j(u_i-1)/2 = 0$

$$\tau_B = \frac{n_c - n_d}{\sqrt{(n_0 - n1)(n_0 - n_2)}} = \frac{16 - 29}{\sqrt{(45-2)(45-0)}} = -\frac{13}{\sqrt{1935}} \approx -0.296$$

4.2 R_Studio Results

The data comparison between RankPage and TFIDF values is contained in file <**page-tfidf-data**>. Uploading the data into "R" yields the following result:

```
> Kendall::Kendall(y,x)
tau = -0.435, 2-sided pvalue =0.11572
```

4.3 Conclusion

According to [1]:

- If the agreement between the two rankings is perfect (i.e., the two rankings are the same) the coefficient has value 1.
- If the disagreement between the two rankings is perfect (i.e., one ranking is the reverse of the other) the coefficient has value -1.
- If X and Y are independent, then we would expect the coefficient to be approximately zero.

Then, by the predicates above, we can conclude PageRanking and TFIDF Ranking variables are independent since the coefficient results are not close to 1 or -1, neither in the manual calculation (4.1), nor the "R" calculation (4.2). Additionally, Figure 3 demonstrates in the graph how scattered the plots are between their values.

References

[1] Graph structure in the web. (n.d.) Retrieved February 15, 2016, from `https://en.wikipedia.org/wiki/Kendall_rank_correlation_coefficient#Tau-b`

[2] Daily Estimate Size of the World Wide Web. (n.d.) Retrieve February 17, 2015, from `http://www.worldwidewebsize.com/`

[3] Football Google Search. (n.d.) Retrieve February 17, 2015, from `http://www.google.com/?gws_rd=ssl#q=football/`