Old Dominion Univeristy

CS 495: Introduction to Web Science
Instructor: Michael L. Nelson, Ph.D
Fall 2014 4:20pm - 7:10pm R, ECSB 2120

Assignment # 3

GeorgeC. Micros UIN: 00757376

Honor Pledge

I pledge to support the Honor System of Old Dominion University. I will refrain from any form of academic dishonesty or deception, such as cheating or plagTFIDF TF IDF URIiarism. I am aware that as a member of the academic community it is my responsibility to turn in all suspected violations of the Honor Code. I will report to a hearing if summoned.

Signed _____

October 2, 2014

George C. Micros

Written Assignment 3

Fall 2014
CS 495: Introduction to Web Science
Dr. Michael Nelson

October 2, 2014

Contents

Chapter 1
Written Assignment 3

1.1 Question 1

1.1.1 The Question

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 41d5f125d13b4bb554e6e31b6b591eeb

(“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:
Keep both files for each URI (i.e., raw HTML and processed). If you’re feeling ambitious, “boilerpipe” typically does a good job for removing templates:

`https://code.google.com/p/boilerpipe/`

1.1.2 The Answer

In this weeks assignment the 1000 unique URLs collected in assignment 2 were used to perform a search over the collection for a specific term and then rank the results using specified metrics, namely TFIDF and PageRank.

In the assignment instructions it si advised that the temptation to store information using URLs as file descriptors. This is because URLs contain special characters which are less than friendly to the operating system. In attempt to prevent a horrible mistake and plan ahead the first step was to create a lookup table that contained each URL and the identified for webpage filename. The proposed identifier was the md5 checksum. Once a lookup table was created the process of collecting the webpages and extracting thie text could proceed.

The webpages were retrieved using cURL and sending a GET request. The response was stored in a directory and named the che cksum of the URL. Once these pages were requested another scirpt sent the same request, but the output was piped into lynx to extract the text and then stored in a different directory with the same hashed URL value.

```
1 #! /bin/bash
2
3 rm LUT
4 while read -r line
5 do
6     line=$line
7     hash=$(echo $line | md5sum | awk '{print $1 }')
8     echo $hash, $line >> LUT
9 done < ./data/temp
```

Listing 1: Bash script that creates a lookup tables of URLs and their hashes

```
1 #! /bin/bash
2
3 rm -rf pages/
4 mkdir pages/
5
6 while read -r line
7 do
```

```
8       line=$line
9       hash=$(echo $line | md5sum | awk '{print $1}')
10      curl -X GET $line > ./pages/$hash
11
12  done < ./data/temp
```

Listing 2: Bash script to perform GET on URLs

```
1   #! /bin/bash
2
3   rm -rf processd
4   mkdir processd
5
6   while read -r line
7   do
8       line=$line
9       hash=$(echo $line | md5sum | awk '{print $1}')
10      curl -X GET $line | lynx -stdin -dump -force_html -nolist > ./processd/$hash.proc
11  done < ./data/temp
```

Listing 3: Bash script to extract text from webpage, ignoring links

1.2 Question 2

1.2.1 The Question

Choose a query term (e.g., "shadow") that is not a stop word (see week 4 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 4 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:

	TFIDF	TF	IDF	URL
·	0.150	0.014	10.680	http://foo.com
	0.085	0.008	10.680	http://bar.com

Table 1.1: Table 1. 10 Hits for the term "shadow", ranked by TFIDF.

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

1.2.2 The Answer

The search term used was "internet". In the collection of the 1000 URLs approximately 100 contained the term "internet". Utilizing mainly the magical capabilities of "grep" the number of times the term occurred and the number of words in each document were determined. These two numbers define the *term frequency* (TF) of each document.

$$TF = \frac{\text{number of term occurances}}{\text{number of words in document}}$$

The *inverse document frequency* (IDF) of the query is defined as the log base 2 of the number of webpages in the corpus over the number of webpages containing the term.

$$TF = \log_2\left(\frac{\text{number of webpages in corpus}}{\text{number of webpagescontainign the term}}\right)$$

This factor is constant for each search term and contributes as a weight function when multiple terms as queried together. For the size fo the internet the figure of 40,000,000,000 was used, based on Google estimates at the time of this project. The number of documents containing the term for the 701,000,000 URLs according to Google. However, given that we are searching a single term the IDF factor is constant for all webpages and does not affect the overall ranking.

One challenging fact was that logarithms are complicated to evaluate, especially in Bash which only supports the natural logarithm. No worries after years of math courses I finally have a use for the change of base formula.

$$\log_a b = \frac{\ln b}{\ln a}$$

Therefore we now have the necessary tools to evaluate the logarithm of any base in Bash, specifically we are interested in base 2. The IDF was computed and multiplied with TF to give the TFIDF value for each URL. The descending list of the 10 hits is listed below.

```
#! /bin/bash

#cnt=$(grep internet ./processd/* | awk -F: '{ print $1}' | uniq -c | awk '{ print $1}' | wc -l)
term=$(grep internet ./processd/* | awk -F: '{ print $1}' | uniq -c | awk '{ print $1}'> term.txt)
dwc=$(wc -w `grep internet ./processd/* | awk -F: '{print $1}' | uniq -c |awk '{ print$2}'` | awk
    '{print $1}' | head -n -1 > dwc.txt)
urls=$(grep internet ./processd/* |awk -F: '{print $1}' |uniq -c |  awk -F/ '{print $3}' | awk -F.
    '{print $1}' | grep -f - ./LUT | awk -F, '{print $2}'>urls.txt >urls.txt)
paste term.txt dwc.txt urls.txt > 5.txt
size=40000000000
cnt=701000000
idf=$(echo ";l($size/$cnt)/l(2)"| bc -l)

rm -f table
while read -r a b c
do
    res=$(echo "$a/$b" | bc -l)
    num=$(echo "$res*$idf" | bc -l);
    #echo $res $c
    echo $num $res $idf $c >> table
done <5.txt

head -n 10 table | sort -r -k1 | xargs printf "%.8f, %.8f, %.8f, %s\n" >tfidf
head -n 10 table | sort -r -k1 | awk '{print $4}' | awk -F/ '{print $3}' >  urlsRes


head -n 10 table | sort -r -k1 | xargs printf "%.8f & %.8f & %.8f & \\ url{%s} \\\\\ \n"  > t1
```

Listing 4: Bash script to retrieve and rank hits for the term “internet”

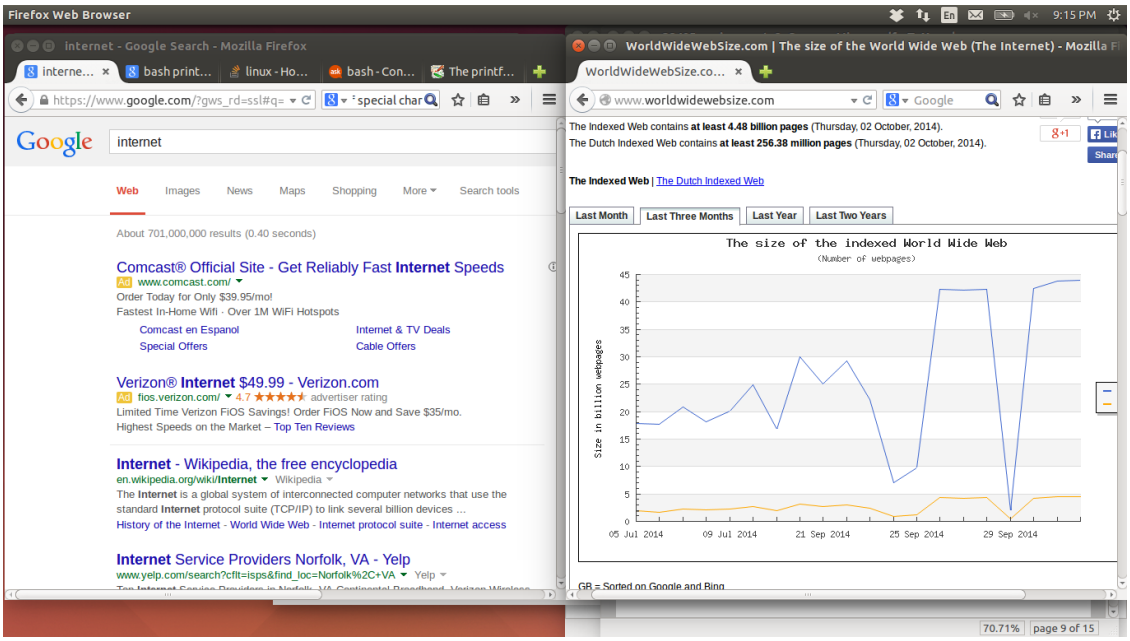

Fig. 1.1: Size of corpus and result set

.

TFIDF	TF	IDF	URLS
0.02297024	0.00393701	5.83444175	http://www.whitehouse.gov/
0.02286095	0.00391828	5.83444175	http://pastebin.com/raw.php?i=n1qTeikM
0.01246676	0.00213675	5.83444175	http://radar.oreilly.com/2014/02/oobleck-security.html
0.01185862	0.00203252	5.83444175	http://www.slideshare.net/timoreilly/government-for-the-people-by-the-people-in-the-21st-century
0.00791111	0.00135593	5.83444175	http://solidcon.com/solid2014/public/schedule/list
0.00340399	0.00058343	5.83444175	http://www.economist.com/news/united-states/21614225-bright-foreigners-study-america-shame-they-cant-stay-coming-and-going
0.00201814	0.00034590	5.83444175	https://stopthesecrecy.net/
0.00201326	0.00034507	5.83444175	http://www.ed.gov/connected
0.00200084	0.00034294	5.83444175	http://solidcon.com/solid2014/public/schedule/speakers?cmp=tw-na-confreg-home-sld14_solid_twitter_posts
0.00058581	0.00010041	5.83444175	http://violentmetaphors.com/2014/03/25/parents-you-are-being-lied-to/

Table 1.2: Table 2. 10 Hits for the term “internet”, ranked by TFIDF.

1.3 Question 3

1.3.1 The Question

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:

.

PR	URL
0.9	http://foo.com
0.5	http://bar.com

Table 1.3: Table 2. 10 hits for the term "shadow", ranked by PageRank.

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

1.3.2 The Answer

The other metric that was to be determied as used to sort the list of hits was the PageRank. In the order to avoid the manual labor incorporated with copying each URL individually a Perl script was used to automate this task. The results of this script are consistent with the results of the other websites.

```
#! /bin/bash

rm prRes

while read -r line
do
 a="http://"
    s=$(./page.pl $a$line)

    s=$(echo "scale=1; $s/10" | bc -l)
    echo $s, $line >> prRes
done < "urlsRes"

sort -nr -k1 -t,  prRes > PR.txt
```

Listing 5: Bash script to run CarbonDate and store results

```
#!/usr/bin/perl

$webpage = $ARGV[0];


use WWW::Google::PageRank;
my $pagerank = WWW::Google::PageRank->new;
my $score = $pagerank->get($webpage);
if($score == "")
```

```
{
        $score = 0;
}

print scalar($score), "\n";
```

Listing 6: Python script to find “Estimated Creation Date” in JSON file

PR	URL
1.0	www.whitehouse.gov
.8	www.slideshare.net
.8	www.ed.gov
.8	www.economist.com
.6	radar.oreilly.com
.6	pastebin.com
.4	violentmetaphors.com
0	stopthesecrecy.net
0	solidcon.com
0	solidcon.com

Table 1.4: Table 3. 10 Hits for the term “internet”, ranked by PageRank.

In both rankings the whitehouse website is the first result. Its TF was fairly close to that of pastebin.com, ut the pagerank of pastebin is significantly lower. Despite the first site remainign the same with the different matrics it is obvious that the order anf the ranking is not perserved as well as the relative poisiton with respet to each other.

1.4 Question 4

1.4.1 The Question

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_tau_rank_correlation_coefficient#Tau-b

http://en.wikipedia.org/wiki/Correlation_and_dependence

1.4.2 The Answer

The Tau value is 0.898 and the p-val 0.002205

```
#! /usr/bin/Rscript

library(Kendall)

a <- read.csv("tfidf");
b <- read.csv("PR.txt");

a <- a[,1];
b <- b[,1];


cor(a,b,method="kendall")
Kendall(a,b)
```

Listing 7: R script that computes the Tau and p value of the Kendall Tau correlation

References

1. https://stat.ethz.ch/pipermail/r-help//2012-August/333656.html
2. http://digitalpbk.com/perl/perl-script-check-google-pagerank
3. http://www.google.com
4. http://jakevdp.github.io/blog/2012/10/14/scipy-sparse-graph-module-word-ladders/
5. http://curl.haxx.se/docs/httpscripting.html
6. http://www.crummy.com/software/BeautifulSoup/bs4/doc/
7. http://www.rmi.net/~lutz/
8. http://www.cs.cornell.edu/home/kleinber/networks-book/
9. http://thomassileo.com/blog/2013/01/25/using-twitter-rest-api-v1-dot-1-with-python/
10. http://www.cs.odu.edu/~mklein/cs796/lecture/