Assignment 3

CS 595: Introduction to Web Science

Fall 2013

Shawn M. Jones

Finished on October 3, 2013

1

Question

```
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
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:

% 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).
```

Answer

Downloading the URIs was done relatively easily using the script shown in Listing 1.

The script is run like so:

```
./downloadURIs.sh urilist-final.txt shalist-final.txt collection
    failedfile.txt
```

The arguments to the script are:

1. file containing URIs, one per line (`urilist-final.txt`)
2. file used to associate SHA-1 hashes with each URI, written to as each URI is processed, with the SHA-1 hashes being part of the filenames of the downloaded representations of each URI (`shalist-final.txt`)
3. directory name to store the downloaded representations (`collection`)
4. file used to store URIs that fail to download (`failedfile.txt`)

Once the script is run, the directory used in the third argument (`collection`) contains filenames, such as `f8017e9dd34d714681d55693689736d5d3f56021.raw` and `f8017e9dd34d714681d55693689736d5d3f56021.processed`. Files with a `.raw` extension contain the raw representation downloaded on line 32. Files containing a `.processed` contain the representation stripped of any HTML, done on line 42.

The file used in the second argument (`shalist-final.txt`) is used to associate these filenames to URIs (e.g. `f8017e9dd34d714681d55693689736d5d3f56021` corresponds to `http://www.cnn.com/2013/07/19/politics/obama-zimmerman/`).

The `set -e` on line 4 causes the script to exit at the first sign of trouble. This is a simple error-handling mechanism to place in Bourne shell scripts.

Unfortunately, the curl call on line 32 produced the following errors for several URIs:

- `curl: (7) couldn't connect to host`
- `curl: (18) transfer closed with outstanding read data remaining`
- `curl: (52) Empty reply from server`

This is why lines 35-38 exist. Line 31 undoes the error-handling mechanism so we can deal with the failed status on lines 33 and 35. Any URI that fails to download the first time is stored in a *failed file* (the fourth argument) which can then be fed through the script again, as the first argument, to (hopefully) successfully download the representations at a later time. Fortunately, only one re-run was required.

```
1  #!/bin/bash
2
3  # if anything fails, QUIT ASAP!
4  set -e
5
6  input=$1
7  listfile=$2
8  workingdir=$3
9  failedfile=$4
10
11 echo "Removing ${listfile}"
12 if [ -f "${listfile}" ]; then
13     rm "${listfile}"
14 fi
15
16 echo "Cleaning out ${workingdir}"
17 if [ -d "${workingdir}" ]; then
18     rm -rf "${workingdir}"
19 fi
20
21 mkdir -p "${workingdir}"
22
23 for line in `cat ${input}`; do
24     sha=`echo "${line}" | shasum | awk '{ print $1 }'`
25     echo "working on ${line}, now ${sha}"
26     echo "${line}    ${sha}" >> ${listfile}
27
28     outfileraw="${sha}.raw"
29
30     # curl appears to be where we can expect failure, so manage
           it
31     set +e
32     curl -A "Mozilla/5.0 (compatible; MSIE 10.0; Windows NT 6.1;
           WOW64; Trident/6.0)" "${line}" >> "${workingdir}/${
           outfileraw}"
33     status=$?
34
35     if [ $status -ne 0 ]; then
36         echo ${line} >> ${failedfile}
37         echo ${line} failed to work, moving on
38     fi
39     set -e
40
41     outfileproc="${sha}.processed"
42     lynx -dump -force_html "${workingdir}/${outfileraw}" >> "${
           workingdir}/${outfileproc}"
43     sleep 1
44 done
```

```
45
46 echo "finished"
```

Listing 1: Bourne-Again Shell Program for downloading URIs from Assignment 2

2

Question

2. Choose a query term (e.g., "shadow") that is not a stop words (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:

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 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!

Answer

Searching for the term *football* yielded quite a few results. This was done with the following command, which also selects out the top 10 for use in this exercise.

```
grep -i football *.processed | awk -F: '{ print $1 }' | sort |
    uniq -c | sort -rn | head -n 10
```

Which returned:

```
  19 87c993bbce1bd82db37cecc698c699e868d83fda.processed
  14 c60fdff864fe97d0a9072bae75dbac742b8e5ef4.processed
  10 9b4a0e8f58cac8c079780008762851aff600b8e7.processed
   8 5778ff1e943bcffb35088d5356e018fe91e6b348.processed
   7 9495c088454cb1f1e0dd8578a851daf9fda109a4.processed
   6 ebf64bd74b9b4d76e9eb1914943966291a3f6f80.processed
   6 e563ce7cf28c6844fd0bd962871530ec3d4ea1c6.processed
   6 4dcb9dc20543b9b0936ad8a29ce9dacccbc782b2.processed
   6 49c3c4171caf2902b5450b5c4e80c8dc4eb0073c.processed
   5 e2ab060d02ca13e0ddc8e5c30a732390045c039b.processed
```

This gives us raw term frequency for these files, but doesn't give us the word count so we can normalize it. To get the word count for the calculation, I did the following:

```
for i in `grep -i football *.processed | awk -F: '{ print $1 }'
    | sort | uniq -c | sort -rn | head -n 10 | awk '{ print $2
    }'`; do wc -w $i ; done
```

Which returned:

```
    7339 87c993bbce1bd82db37cecc698c699e868d83fda.processed
    5634 c60fdff864fe97d0a9072bae75dbac742b8e5ef4.processed
   39130 9b4a0e8f58cac8c079780008762851aff600b8e7.processed
    3810 5778ff1e943bcffb35088d5356e018fe91e6b348.processed
     910 9495c088454cb1f1e0dd8578a851daf9fda109a4.processed
    3866 ebf64bd74b9b4d76e9eb1914943966291a3f6f80.processed
    2229 e563ce7cf28c6844fd0bd962871530ec3d4ea1c6.processed
    3109 4dcb9dc20543b9b0936ad8a29ce9dacccbc782b2.processed
    1922 49c3c4171caf2902b5450b5c4e80c8dc4eb0073c.processed
    3394 e2ab060d02ca13e0ddc8e5c30a732390045c039b.processed
```

Another step is necessary to get the URIs:

```
for i in `grep -i football *.processed | awk -F: '{ print $1 }'
    | sort | uniq -c | sort -rn | head -n 10 | awk '{ print $2 }'
     | sed 's/.processed//g'`; do grep $i ../shalist-final.txt ;
    done
```

And normalized TF is calculated for each URI like so:

$$TF(football) = TF_{raw}(football)/occurrences(football)$$

Which returns the following:

```
http://www.dailykos.com/story/2013/06/18/1216969/-D-C-Football?
    utm_source=twitterfeed&utm_medium=twitter&utm_campaign=Feed%3
    A+dailykos%2Findex+%28Daily+Kos%29      87
    c993bbce1bd82db37cecc698c699e868d83fda
http://gif.mocksession.com/2013/02/rubio-is-thirsty/
    c60fdff864fe97d0a9072bae75dbac742b8e5ef4
http://www.dailykos.com/story/2013/05/09/1207970/-Agreeing-with-
    McCain-on-Cable-bill?utm_source=twitterfeed&utm_medium=
    twitter&utm_campaign=Feed%3A+dailykos%2Findex+%28Daily+Kos%29
        9b4a0e8f58cac8c079780008762851aff600b8e7
http://www.tampabay.com/news/politics/national/mitt-romney-is-
    republican-partys-nominee-but-not-the-standard-bearer/1248507
        5778ff1e943bcffb35088d5356e018fe91e6b348
http://www.cnn.com/2013/04/23/justice/ohio-steubenville-coach/
    index.html?hpt=hp_t3      9495
    c088454cb1f1e0dd8578a851daf9fda109a4
http://host.madison.com/wsj/news/local/crime_and_courts/appeals-
    court-reverses-federal-judge-s-decision-upholds-collective-
    bargaining/article_c08d81f6-61a3-11e2-8ab7-001a4bcf887a.html
        ebf64bd74b9b4d76e9eb1914943966291a3f6f80
http://bleacherreport.com/articles/1699257-major-league-baseball
    -suspends-ryan-braun-for-remainder-of-2013-season
    e563ce7cf28c6844fd0bd962871530ec3d4ea1c6
http://concord-nh.patch.com/groups/politics-and-elections/p/rep-
    s-sieg-heil-causes-furor      4
    dcb9dc20543b9b0936ad8a29ce9dacccbc782b2
http://folksdresseduplikeeskimos.tumblr.com/      49
    c3c4171caf2902b5450b5c4e80c8dc4eb0073c
http://www.freep.com/article/20121205/NEWS15/121205082/Michigan-
    Rick-Snyder-emergency-manager-law-repeal-Lansing
    e2ab060d02ca13e0ddc8e5c30a732390045c039b
```

As the SHA-1 sums are the keys to join between these three outputs, the calculation of Term Frequency can be done easily by hand (because we only have 10 items), but where's the fun in that?

Because of the sort done on the items, they stay in order, meaning we don't need the keys to have Unix do the normalized TF calculations.

```
export i=0
for item in `grep -i football *.processed | awk -F: '{ print $1
    }' | sort | uniq -c | sort -rn | head -n 10 | awk '{ print $1
     }'`; do rawtf[$i]=$item; i=`expr $i + 1`; done

export i=0
for item in `grep -i football *.processed | awk -F: '{ print $1
    }' | sort | uniq -c | sort -rn | head -n 10 | awk '{ print $2
     }'`; do occur[$i]=`wc -w $item | awk '{ print $1 }'` i=`expr
     $i + 1`; done

for i in `seq 0 9`; do echo "scale=5; ${rawtf[$i]} / ${occur[$i
    ]}" | bc -l; done
```

Which yields:

```
.00258
.00248
.00025
.00209
.00769
.00155
.00269
.00192
.00312
.00147
```

According to de Kunder, Google has currently indexed slightly more than $46,000,000,000$ web pages[1].

Google gives $1,230,000,000$ results for the word *football*.

This gives an inverse document frequency of (using significant digits rules for logarithms [2]):

$$\log_2\left(\frac{46,000,000,000}{1,230,000,000}\right) = \log_2\left(\frac{4600}{123}\right) = \log_2(37.4) = 5.225$$

The final piece of the puzzle is to calculate TFIDF, which is merely the multiplication of the normalized term frequencies for each page with the inverse document frequency calculated above.

Again, done with the help of our handy friends in Unix:

```
for i in `seq 0 9`; do echo "scale=5; ${rawtf[$i]} / ${occur[$i
    ]} * 5.225" | bc -l; done
```

Table 1 displays the results.

TFIDF	TF	IDF	URI
0.135	0.00258	5.225	http://www.dailykos.com/story/2013/06/18/1216969/-D-C-Football?utm_source=twitterfeed&utm_medium=twitter&utm_campaign=Feed%3A+dailykos%2Findex+%28Daily+Kos%29
0.0130	0.00248	5.225	http://gif.mocksession.com/2013/02/rubio-is-thirsty/
0.0013	0.00025	5.225	http://www.dailykos.com/story/2013/05/09/1207970/-Agreeing-with-McCain-on-Cable-bill?utm_source=twitterfeed&utm_medium=twitter&utm_campaign=Feed%3A+dailykos%2Findex+%28Daily+Kos%29
0.0109	0.00209	5.225	http://www.tampabay.com/news/politics/national/mitt-romney-is-republican-partys-nominee-but-not-the-standard-bearer/1248507
0.0402	0.00769	5.225	http://www.cnn.com/2013/04/23/justice/ohio-steubenville-coach/index.html?hpt=hp_t3
0.0081	0.00155	5.225	http://host.madison.com/wsj/news/local/crime_and_courts/appeals-court-reverses-federal-judge-s-decision-upholds-collective-bargaining/article_c08d81f6-61a3-11e2-8ab7-001a4bcf887a.html
0.0141	0.00269	5.225	http://bleacherreport.com/articles/1699257-major-league-baseball-suspends-ryan-braun-for-remainder-of-2013-season
0.0100	0.00192	5.225	http://concord-nh.patch.com/groups/politics-and-elections/p/rep-s-sieg-heil-causes-furor
0.0163	0.00312	5.225	http://folksdresseduplikeeskimos.tumblr.com/
0.0077	0.00147	5.225	http://www.freep.com/article/20121205/NEWS15/121205082/Michigan-Rick-Snyder-emergency-manager-law-repeal-Lansing

Table 1: Table of URIs, TF, IDF and TF*IDF containing the word *football*

3

Question

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.

Answer

4

Question

```
=====================================================
======Question 4 is for 3 points extra credit=======
=====================================================

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-
http://en.wikipedia.org/wiki/Kendall_tau_rank_correlation_coefficient#Tau-b
http://en.wikipedia.org/wiki/Correlation_and_dependence
```

Answer

References

[1] de Kunder, M. World wide web size: Daily estimated size of the world wide web, Oct. 2013.

[2] Euler, W. Significant figures when using logs, Feb. 2005.