Assignment 2

Fall 2014
CS595 Web Science
Dr. Michael Nelson

Mathew Chaney

October 2, 2014

Contents

1 Question 1 **3**
1.1 Question . . . 3
1.2 Resources . . . 3
1.3 Answer . . . 3

2 Question 2 **5**
2.1 Question . . . 5
2.2 Resources . . . 5
2.3 Answer . . . 5

3 Question 3 **9**
3.1 Question . . . 9
3.2 Resources . . . 9
3.3 Answer . . . 9

List of Figures

Listings

1 get_html.py . . . 4
2 count_terms function . . . 6
3 get_uris functions . . . 6
4 Loading the uri map . . . 6
5 Getting filename from URI . . . 6
6 Stripping html tags from content . . . 7
7 Processing TF, IDF & TFIDF . . . 7
8 Writing results to uri_frequencies file . . . 7
9 uri_frequencies file . . . 8
10 page_ranks file . . . 9

1 Question 1

1.1 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:

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

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/
```

1.2 Resources

- md5: https://docs.python.org/2/library/md5.html
- requests: http://docs.python-requests.org/en/latest/
- futures: https://pypi.python.org/pypi/futures
- BeautifulSoup: http://www.crummy.com/software/BeautifulSoup/bs4/doc/

1.3 Answer

Using the python script in Listing 1, 1000 unique URIs were dereferenced and their raw contents were stored in the `html/raw/` folder as a file with the filename as the md5-hashed URI. These were then stripped of all html elements and their processed contents were stored in the `html/processed/` folder as the same md5-hashed filename.

```
#! /usr/bin/python

import requests
import futures
import md5
from bs4 import BeautifulSoup
import pickle

def convert(uri):
    return md5.new(uri).hexdigest()

def get_html(uri):
    print('Getting {}'.format(uri))
    response = requests.get(uri)
    return response.url, response.status_code, response.content

if __name__ == '__main__':
    with open('uris') as infile:
        uris = [uri.rstrip('\n') for uri in infile]

    with futures.ThreadPoolExecutor(max_workers=8) as executor:
        uri_futures = [executor.submit(get_html, uri) for uri in uris]
        for future in futures.as_completed(uri_futures):
            try:
                uri, status_code, content = future.result()
            except Exception as exc:
                print('{} generated an exception: {}'.format(uri, exc))
                continue
            if status_code == 200:
                hashed_uri = convert(uri)
                print('Writing {} as {}'.format(uri, hashed_uri))
                try:
                    with open('html/raw/' + hashed_uri, 'w') as outfile:
                        outfile.write(uri + '\n')
                        outfile.write(content)
                    with open('html/processed/' + hashed_uri, 'w') as outfile:
                        outfile.write(uri + '\n')
                        outfile.write(BeautifulSoup(content).get_text().encode('utf8'))
                except Exception as e:
                    print '**** ERROR **** --- ' + uri
                    print e
            else:
                print('Not writing {}, bad status code: {}'.format(uri, status_code))
```

Listing 1: get_html.py

2 Question 2

2.1 Question

```
2.  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:

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

TFIDF   TF      IDF     URI
-----   --      ---     ---
0.150   0.014   10.680  http://foo.com/
0.085   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.

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

2.2 Resources

- word counting: http://stackoverflow.com/questions/17507876/trying-to-count-words-in-a-string-python
- pickle: https://docs.python.org/2/library/pickle.html

2.3 Answer

First, the function `count_terms` was used to count the term frequency for a given term in all documents.

```
12 def count_terms(term, file_list=os.listdir('html/processed')):
13     for filename in file_list:
14         with open('html/processed/' + filename) as infile:
15             uri = infile.readline().strip()
16             text = infile.read()
17             count = text.count(term)
18             if count > 0:
19                 print('{} {}'.format(count, uri))
20                 return count, uri
21     return None, None
```

Listing 2: count_terms function

Ten of the results were chosen at random and stored in the `uri_counts` file. In order to easily identify which file corresponds to which URI, since the filename is the non-reversible md5-hashed URI string, a mapping from URI to filename was created using the functions in Listing 3 and serialized in the `uri_map` file using the `pickle` library.

```
23 def get_uri(uri):
24     for filename in os.listdir('html/processed/'):
25         with open('html/processed/' + filename) as infile:
26             if uri in infile.readline():
27                 return uri, filename
28     return None, None
29
30 def get_uris():
31     uri_file = {}
32     for uri in open('uris').read().split('\n'):
33         uri, filename = get_uri(uri)
34         if not uri:
35             continue
36         uri_file[uri] = filename
37     return uri_file
```

Listing 3: get_uris functions

Reading from the file was done with the line in Listing 4. This loaded the serialized URI to filename map for future use.

```
10 uri_map = pickle.load(open('uri_map', 'rb'))
```

Listing 4: Loading the uri map

To proceed with processing each of the files to find Term Frequency (TF), Inverse Document Frequency (IDF) and the product of the two (TFIDF), each URI's corresponding file was found using the `get_filename` function found in Listing 5.

```
39 def get_filename(uri):
40     if uri_map.has_key(uri):
41         return uri_map[uri]
42     return None
```

Listing 5: Getting filename from URI

Then, they were stripped of html tags using the `strip_html` function in Listing 6.

```
44 def strip_html(filename):
45     if not filename:
46         print 'invalid filename'
47         return
48     with open('html/processed/' + filename) as infile:
49         # To remove URI in first line
50         infile.readline()
51         # Removing all punctuation
52         strs = infile.read()
53         r = re.compile(r'[{}]'.format(punctuation))
54         content = r.sub(' ', strs)
55         return content
```

Listing 6: Stripping html tags from content

And finally the frequencies were calculated for each URI using the functions in Listing 7.

```
57 def get_tf(content, term):
58     return float(content.count(term)) / float(len(content.split()))
59
60 def get_idf(term):
61     present = set()
62     absent = set()
63     for uri, filename in uri_map.iteritems():
64         content = strip_html(filename)
65         if not content:
66             continue
67         if term in content:
68             present.add(uri)
69         else:
70             absent.add(uri)
71     return math.log(float(len(absent)) / float(len(present)), 2)
72
73 def process_uri(uri, term):
74     tf = get_tf(strip_html(get_filename(uri)), term)
75     tfidf = tf * idf
76     return tf, tfidf
77
78 idf = get_idf('shadow')
```

Listing 7: Processing TF, IDF & TFIDF

These frequencies were then written to the `uri_frequencies` file using the code in 8.

```
107         with open('uri_counts') as infile:
108             uris = uris = [line.split()[1] for line in infile.read().split('\n')]
109             with open('uri_frequencies', 'w') as outfile:
110                 outfile.write('{:<7} {:<7} {:<7} {:<7}\n'.format('TFIDF', 'TF', 'IDF', 'URI'
                        ))
111                 for uri in uris:
112                     tf, tfidf = process_uri(uri, term)
113                     outfile.write('{:5.4f}  {:5.4f}  {:5.4f}  {}\n'.format(tfidf, tf, idf,
                            uri))
```

Listing 8: Writing results to uri_frequencies file

And here are the results:

```
1  TFIDF    TF         IDF        URI
2  ------   --         ---        ---
3  0.0220   0.0065   3.3825   http://news.google.com/
4  0.0159   0.0047   3.3825   http://www.easkme.com/2014/07/mail-merge-in-gmail.html#.VB8GnkIk6rA.
       facebook
5  0.0113   0.0033   3.3825   http://btc-news-bot.tumblr.com/post/98066358831/we-need-to-do-a-
       better-job-explaining-bitcoin-in-ways#_=_
6  0.0109   0.0032   3.3825   http://musicisthedrug-revolution.tumblr.com/post/98067184737/
       description-you-know-what-time-it-is-world-cup#_=_
7  0.0099   0.0029   3.3825   http://www.ebay.com/itm/4-5-Android-Smartphone-Dual-Sim-Dual-Core-
       Unlocked-WIFI-3G-GSM-Cell-phone-AT-T-/271610837947?pt=Cell_Phones&hash=item3f3d447bbb
8  0.0092   0.0027   3.3825   http://www.blogdeizquierda.com/2014/09/mundo-bitcoin-banqueros-de-eu
       -ven.html?utm_source=twitterfeed&utm_medium=twitter
9  0.0064   0.0019   3.3825   http://www.valuewalk.com/2014/09/best-apps-apple-iphone-6/
10 0.0064   0.0019   3.3825   http://www.datelinemovies.com/2014/07/bloopers-for-season-4-game-of-
       thrones.html#sthash.yYuV0eDx.uxfs
11 0.0027   0.0008   3.3825   http://abusidiqu.com/its-all-scripted-ebola-virus-is-a-biological-
       weapon-from-the-us-read-this-shocking-report/
12 0.0019   0.0006   3.3825   http://rss-now.blogspot.com/2014/09/nasa-boeing-space-x-iss.html?
       utm_source=dlvr.it&utm_medium=twitter
```

Listing 9: uri_frequencies file

3 Question 3

3.1 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.
```

3.2 Resources

- Page Rank Checker: http://www.prchecker.info/check_page_rank.php

3.3 Answer

Using the Page Rank Checker website to input each of the URIs found in the ten selected URIs from question 2 the results in Listing 10 was determined.

```
1  PageRank     URI
2  ————————     ———
3  0.8          http://www.ebay.com/
4  0.8          http://news.google.com/
5  0.5          http://www.valuewalk.com/
6  0.2          http://www.blogdeizquierda.com/
7  0.2          http://abusidiqu.com/
8  0.0          http://www.easkme.com/
9  0.0          http://www.datelinemovies.com/
10 0.0          http://rss-now.blogspot.com/
11 0.0          http://musicisthedrug-revolution.tumblr.com/
12 0.0          http://btc-news-bot.tumblr.com/
```

Listing 10: page_ranks file

In looking at the similarities and differences in the results of question 2 and question 3 it seems that page rank is unrelated to term frequency measurements. This is logical because the search term isn't taken as an input when calculating page rank. Also, finding page rank has a different goal than

measuring search term relevance. It is used to objectively find which pages have a higher probability of a user randomly navigating to the page, which is unrelated to the content of the pages in the given set and is a function of the graph created by links contained in the pages of the set.