CS532 Web Science: Assignment 3

Finished on February 18, 2016

Dr. Michael L. Nelson

Naina Sai Tipparti
ntippart@cs.odu.edu

Contents

Listings

List of Figures

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

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 [4]-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. For reference, the URIs were written as the first line of each of their content files.[8][2][1]

```
#! /usr/bin/python

import requests
import concurrent.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('links') as infile:
        uris = [uri.rstrip('\n') for uri in infile]

    with concurrent.futures.ThreadPoolExecutor(max_workers=8) as executor:
        uri_futures = [executor.submit(get_html, uri) for uri in uris]
        for future in concurrent.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 + '.processed.txt', '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

Problem 2

Question

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 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

First, the function `count_terms` was used to count[9] the term frequency for the given term "shadow" in all documents.

```
def count_terms(term, file_list=os.listdir('html/processed')):
    for filename in file_list:
        with open('html/processed/' + filename + '.processed.txt') as infile:
            uri = infile.readline().strip()
            text = infile.read()
            count = text.count(term)
            if count > 0:
                print('{} {}'.format(count, uri))
                return count, uri
```

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`[7] library.

```
def get_uri(uri):
    for filename in os.listdir('html/processed/'):
        with open('html/processed/' + filename + '.processed.txt') as infile:
            if uri in infile.readline():
                return uri, filename
    return None, None

def get_uris():
    uri_file = {}
    for uri in open('links').read().split('\n'):
        uri, filename = get_uri(uri)
        if not uri:
            continue
        uri_file[uri] = filename
```

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.

```
map_file = 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.

```
def get_filename(uri):
    if uri_map.has_key(uri):
        return uri_map[uri]
```

Listing 5: Getting filename from URI

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

```
def strip_html(filename):
    if not filename:
        print 'invalid filename'
        return
    with open('html/processed/' + filename + '.processed.txt') as infile:
        # To remove URI in first line
        infile.readline()
        # Removing all punctuation
        strs = infile.read()
        r = re.compile(r'[{}]'.format(punctuation))
        content = r.sub(' ', strs)
```

Listing 6: Stripping HTML tags from content

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

```
def get_tf(content, term):
    return float(content.count(term)) / float(len(content.split()))

def get_idf(term):
    present = set()
    absent = set()
    for uri, filename in uri_map.iteritems():
        content = strip_html(filename)
        if not content:
            continue
        if term in content:
            present.add(uri)
        else:
            absent.add(uri)
    return math.log(float(len(absent)) / float(len(present)), 2)

def process_uri(uri, term):
    tf = get_tf(strip_html(get_filename(uri)), term)
    tfidf = tf * idf
    return tf, tfidf
```

Listing 7: Calculating TF, IDF & TFIDF

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

```
        term = sys.argv[2]
        with open('uri_counts') as infile:
            uris = uris = [line.split()[1] for line in infile.read().split('\n')]
            with open('uri_frequencies', 'w') as outfile:
                outfile.write('{:<7} {:<7} {:<7} {:<7}\n'.format('TFIDF', 'TF', 'IDF', 'URI'
                    ))
                for uri in uris:
                    tf, tfidf = process_uri(uri, term)
```

Listing 8: Writing results to uri_frequencies file

And the results can be seen in Listing 9:

```
TFIDF   TF      IDF     URI
-----   --      ---     ---
0.0220  0.0065  3.3825  http://news.google.com/
0.0159  0.0047  3.3825  http://www.easkme.com/2014/07/mail-merge-in-gmail.html#.VB8GnkIk6rA.
    facebook
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#_=_
0.0109  0.0032  3.3825  http://musicisthedrug-revolution.tumblr.com/post/98067184737/
    description-you-know-what-time-it-is-world-cup#_=_
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
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
0.0064  0.0019  3.3825  http://www.valuewalk.com/2014/09/best-apps-apple-iphone-6/
0.0064  0.0019  3.3825  http://www.datelinemovies.com/2014/07/bloopers-for-season-4-game-of-
    thrones.html#sthash.yYuV0eDx.uxfs
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/
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

Problem 3

Question

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 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

Using the Page Rank[6] Checker website to input each of the URIs found in the ten selected URIs from question 2 the results in Listing 10 was determined.

```
PageRank    URI
---------   ---
0.8         http://www.ebay.com/
0.8         http://news.google.com/
0.5         http://www.valuewalk.com/
0.2         http://www.blogdeizquierda.com/
0.2         http://abusidiqu.com/
0.0         http://www.easkme.com/
0.0         http://www.datelinemovies.com/
0.0         http://rss-now.blogspot.com/
0.0         http://musicisthedrug-revolution.tumblr.com/
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.

Problem 4

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

Answer

Using R, I input the values of TFIDF and Page rank into two separate vectors. Using the package[5] Kendall[3], it calculated a tau of 0.866 and p of 0.0018457. With a tau number so close to +1, there is a high association between the two measured quantities. In the case of ties, both packages "Kendall and cor produce the same result but cor.test produces a p-value which is not as accurate".

```
#! /usr/bin/Rscript

library(Kendall)
data <- read.table("D:/cs532/a3/q4/kendall", header=TRUE)
tfidf <- data[1]
pagerank <- data[2]
cor(tfidf, pagerank, method="kendall")
Kendall(tfidf,pagerank)
```

Listing 11: kendall.r

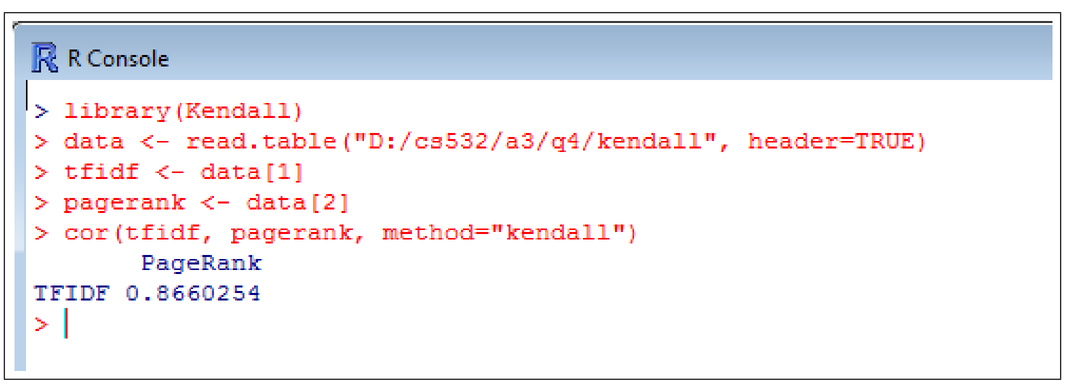

Figure 1: R script that computes the Tau and p value of the Kendall Tau correlation

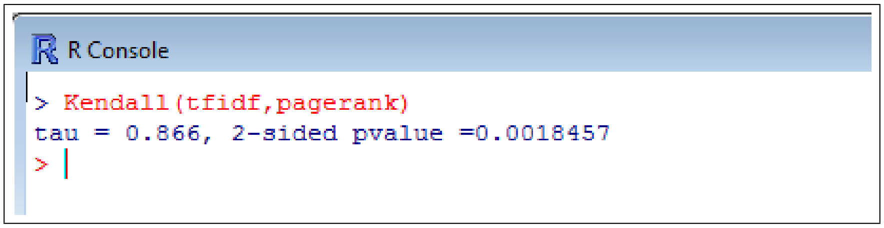

Figure 2: Tau and p value of the Kendall

References

[1] Beautiful soup documentation. http://www.crummy.com/software/BeautifulSoup/bs4/doc/.

[2] futures 3.0.5. https://pypi.python.org/pypi/futures.

[3] The size of the world wide web (the internet). http://en.wikipedia.org/wiki/Kendall_tau_rank_correlation_coefficient#Tau-b.

[4] 4.3. md5md5 message digest algorithm. https://docs.python.org/2/library/md5.htmls.

[5] Package kendall. http://cran.r-project.org/web/packages/Kendall/Kendall.pdf.

[6] Check page rank of web site pages instantly. http://www.prchecker.info/check_page_rank.php.

[7] 11.1. pickle - python object serialization. 11.1. pickle - Python object serialization.

[8] Requests: Http for humans. https://docs.python.org/2/library/md5.htmls.

[9] Trying to count words in a string. http://stackoverflow.com/questions/17507876/trying-to-count-words-in-a-string-python.