Assignment 2

Fall 2014
CS595 Web Science
Dr. Michael Nelson

Mathew Chaney

October 1, 2014

Contents

List of Figures

Listings

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

- None: yet

2.3 Answer