

# **Assignment 2**

**Fall 2014**

**CS595 Web Science**

**Dr. Michael Nelson**

Mathew Chaney

October 1, 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

List of Figures

Listings

1 get\_html.py . . . . . 4

# 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.

```

1  #!/usr/bin/python
2
3  import requests
4  import futures
5  import md5
6  from bs4 import BeautifulSoup
7
8  def convert(uri):
9      return md5.new(uri).hexdigest()
10
11 def get_html(uri):
12     print('Getting {}'.format(uri))
13     response = requests.get(uri)
14     return response.url, response.status_code, response.content
15
16 if __name__ == '__main__':
17     with open('uris') as infile:
18         uris = [uri.rstrip('\n') for uri in infile]
19
20     with futures.ThreadPoolExecutor(max_workers=8) as executor:
21         uri_futures = [executor.submit(get_html, uri) for uri in uris]
22         for future in futures.as_completed(uri_futures):
23             try:
24                 uri, status_code, content = future.result()
25             except Exception as exc:
26                 print('{} generated an exception: {}'.format(uri, exc))
27                 continue
28             if status_code == 200:
29                 hashed_uri = convert(uri)
30                 print('Writing {} as {}'.format(uri, hashed_uri))
31                 map_file.write('{} {}{}\n'.format(uri, hashed_uri))
32                 with open('html/raw/' + hashed_uri, 'w') as outfile:
33                     outfile.write(content)
34                 with open('html/processed/' + hashed_uri, 'w') as outfile:
35                     outfile.write(BeautifulSoup(content).get_text())
36             else:
37                 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 denominator 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