

Assignment 8

Fall 2014

CS595 Web Science

Dr. Michael Nelson

Mathew Chaney

November 28, 2014

Contents

| | | |
|----------|--------------------|-----------|
| 1 | Question 1 | 3 |
| 1.1 | Question | 3 |
| 1.2 | Answer | 3 |
| 2 | Question 2 | 7 |
| 2.1 | Question | 7 |
| 2.2 | Answer | 7 |
| 3 | Question 3 | 8 |
| 3.1 | Question | 8 |
| 3.2 | Answer | 8 |
| 4 | Question 4 | 9 |
| 4.1 | Question | 9 |
| 4.2 | Answer | 9 |
| 5 | Question 5 | 10 |
| 5.1 | Question | 10 |
| 5.2 | Answer | 10 |
| 6 | Appendix A | 11 |
| 7 | References | 14 |

Listings

| | | |
|----|---|----|
| 1 | main for get_uris.py | 3 |
| 2 | referenced variables in get_uris.py | 3 |
| 3 | get_atom function | 4 |
| 4 | add_uri function | 4 |
| 5 | looping over the URIs | 4 |
| 6 | processing each blog | 5 |
| 7 | bounding the terms | 5 |
| 8 | building the histogram | 5 |
| 9 | get_uris.py | 11 |
| 10 | matrix.py | 12 |

1 Question 1

1.1 Question

Create a blog-term matrix. Start by grabbing 100 blogs; include:

```
http://f-measure.blogspot.com/  
http://ws-dl.blogspot.com/
```

and grab 98 more as per the method shown in class.

Use the blog title as the identifier for each blog (and row of the matrix). Use the terms from every item/title (RSS) or entry/title (Atom) for the columns of the matrix. The values are the frequency of occurrence. Essentially you are replicating the format of the "blogdata.txt" file included with the PCI book code. Limit the number of terms to the most "popular" (i.e., frequent) 500 terms, this is *after* the criteria on p. 32 (slide 7) has been satisfied.

Create a histogram of how many pages each blog has (e.g., 30 blogs with just one page, 27 with two pages, 29 with 3 pages and so on).

1.2 Answer

To complete this assignment, a blog word count dataset was required. To start off, a list of blog URIs was obtained using the method described in class, implemented as the `get_uris.py` script. Two default blogs, F-Measure and the Old Dominion Web Science and Digital Libraries blogs, were added to the list and then, using the seed URI provided (Listing 2), the remaining 98 URIs from random blogs within the blogger.com family were added. Then, using the `matrix.py` script, the page counts for each blog were extracted and saved to a file called `pagecounts`. The `matrix.py` script is a modified version of `generatefeedvectors.py` from the book *Programming Collective Intelligence* [1].

```
27 if __name__ == '__main__':  
28     uris = set()  
29     with open('blog_uris', 'a') as outfile:  
30         if len(sys.argv) > 1 and sys.argv[1] == 'new':  
31             for must_have in must_haves:  
32                 uri = get_atom(must_have)  
33                 add_uri(uri, uris, outfile)  
34         else:  
35             with open('blog_uris') as infile:  
36                 [uris.add(line.strip()) for line in infile]  
37         while len(uris) < 100:  
38             uri = get_atom(default)  
39             add_uri(uri, uris, outfile)
```

Listing 1: main for `get_uris.py`

```
7 default = 'http://www.blogger.com/next-blog?navBar=true&blogID=3471633091411211117'  
8 must_haves = ['http://f-measure.blogspot.com/', 'http://ws-dl.blogspot.com/']
```

Listing 2: referenced variables in `get_uris.py`

The `get_uris` main function in Listing 1 was the driver that called the `get_atom` function (shown in Listing 3) to extract the atom [2] URIs from each blog and add them to the set of URIs with the `add_uri` function, shown in Listing 4.

```

10 def get_atom(uri):
11     try:
12         r = requests.get(uri)
13     except Exception, e:
14         return None
15     soup = BeautifulSoup(r.text)
16     links = soup.find_all('link', {'type': 'application/atom+xml'})
17     if links:
18         return str(links[0]['href'])
19     return None

```

Listing 3: get_atom function

```

21 def add_uri(uri, uris, outfile):
22     if uri and uri not in uris:
23         uris.add(uri)
24         outfile.write(uri + '\n')
25     print len(uris), uri

```

Listing 4: add_uri function

The contents of each blog were downloaded and processed by the code shown in Listing 5 and the `get_titles`, `get_words` and `get_next` functions found in Listing 6. This code loops over the URIs that were downloaded with the `get_uris.py` script, parses each entry and extracts all the words in each entry's title. These words were then compiled into a master list for all 100 blogs, with the top 500 words that fit into the range bounded by the code in Listing 7 being used for the final word count.

```

40 with open('blog_uris') as infile:
41     uris = [line.strip() for line in infile]
42     if len(sys.argv) == 2 and sys.argv[1] == 'get':
43         with futures.ThreadPoolExecutor(max_workers=8) as executor:
44             uri_futures = [executor.submit(get_titles, uri) for uri in uris]
45             for future in futures.as_completed(uri_futures):
46                 uri, title, subtitle, pages, wc = future.result()
47                 with open('wcs/' + md5.new(uri).hexdigest(), 'w') as out:
48                     out.write(title + ': ' + subtitle + '\t' + str(pages) + '\t')
49                 json.dump(wc, out)

```

Listing 5: looping over the URIs

```

8 def get_next(d):
9     for item in d.feed.links:
10         if item['rel'] == u'next':
11             return item['href']
12     return None
13
14 def getwords(text):
15     txt = re.compile(r'<[>]+>').sub('', text)
16     words = re.compile(r'^A-Z^a-z+').split(txt)
17     return [word.lower() for word in words if word != '']
18
19 def get_titles(uri):
20     print('processing {}'.format(uri))
21     next = uri
22     wc = {}
23     pages = 0
24     while next is not None:
25         d = feedparser.parse(next)
26         for e in d.entries:
27             words = getwords(e.title.encode('utf-8'))
28             for word in words:
29                 wc.setdefault(word, 0)
30                 wc[word] += 1
31             pages += 1
32             next = get_next(d)
33         print('next {}'.format(next))
34     title = d.feed.title.encode('utf-8')
35     subtitle = d.feed.subtitle[:50].encode('utf-8')
36     print('finished: {}: {}'.format(title, subtitle))
37     return uri, title, subtitle, pages, wc

```

Listing 6: processing each blog

```

70     for w, bc in sorted(apcount.items(), key=lambda x: x[1], reverse=True):
71         frac = float(bc) / len(uris)
72         if frac > 0.1 and frac < 0.5:
73             wordlist.append(w)

```

Listing 7: bounding the terms

To build a histogram showing the blog page counts, the `pagecounts` file was parsed by the R script in Listing 8 and saved as a pdf, which is shown in Figure 1.

```

1 #! /usr/bin/Rscript
2
3 data <- read.table("pagecounts", sep="\t", header=TRUE, comment.char="")
4 counts <- table(data$pages)
5 pdf("hist.pdf")
6 barplot(counts, ylab="Number of Blogs", xlab="Page Count", main="Page Count per Blog")
7 dev.off()

```

Listing 8: building the histogram

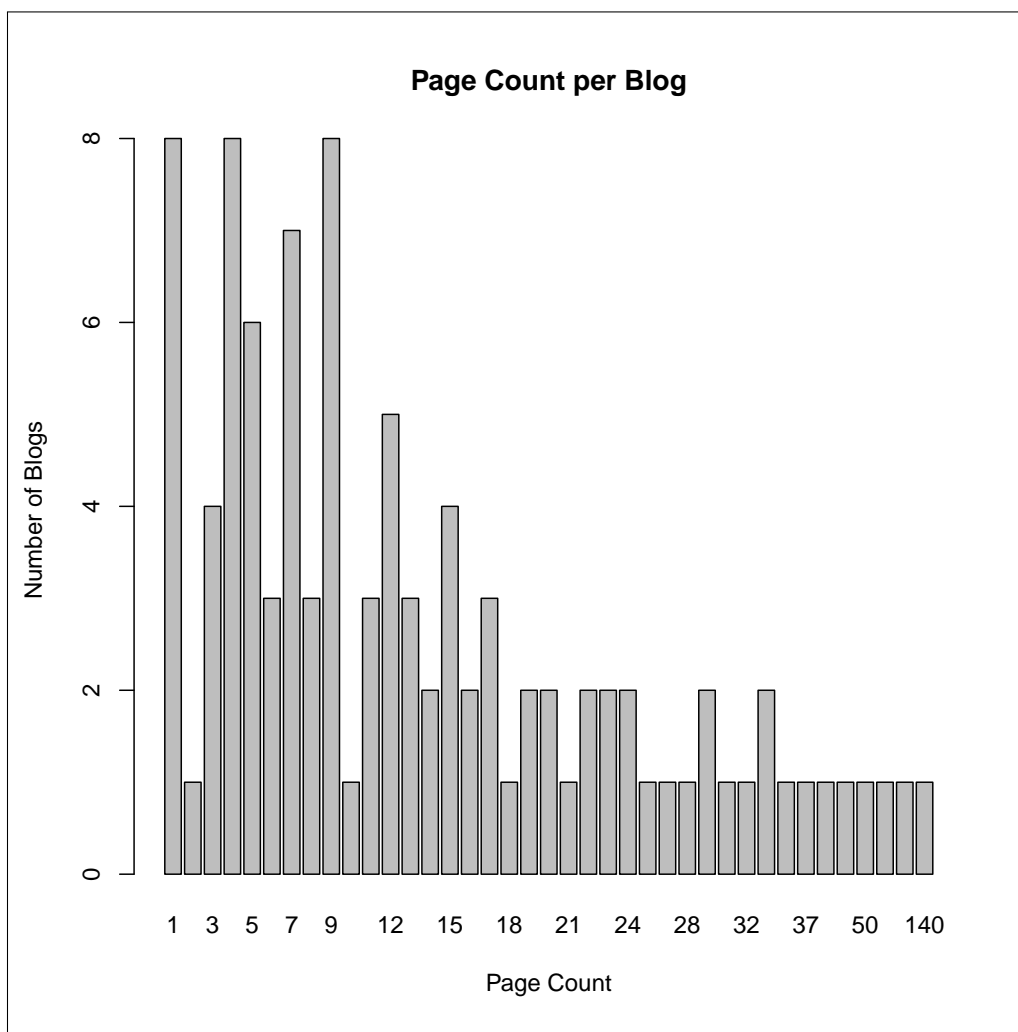


Figure 1: Page Count per Blog

2 Question 2

2.1 Question

Create an ASCII and JPEG dendrogram that clusters (i.e., HAC) the most similar blogs (see slides 12 & 13). Include the JPEG in your report and upload the ascii file to github (it will be too unwieldy for inclusion in the report).

2.2 Answer

3 Question 3

3.1 Question

Cluster the blogs using K-Means, using $k=5,10,20$. (see slide 18).
How many iterations were required for each value of k ?

3.2 Answer

4 Question 4

4.1 Question

Use MDS to create a JPEG of the blogs similar to slide 29.
How many iterations were required?

4.2 Answer

5 Question 5

5.1 Question

Re-run question 2, but this time with proper TFIDF calculations instead of the hack discussed on slide 7 (p. 32). Use the same 500 words, but this time replace their frequency count with TFIDF scores as computed in assignment #3. Document the code, techniques, methods, etc. used to generate these TFIDF values. Upload the new data file to github.

Compare and contrast the resulting dendrogram with the dendrogram from question #2.

Note: ideally you would not reuse the same 500 terms and instead come up with TFIDF scores for all the terms and then choose the top 500 from that list, but I'm trying to limit the amount of work necessary.

5.2 Answer

6 Appendix A

```
1 #!/usr/bin/env python
2
3 import requests
4 import sys
5 from bs4 import BeautifulSoup
6
7 default = 'http://www.blogger.com/next-blog?navBar=true&blogID=3471633091411211117'
8 must_haves = ['http://f-measure.blogspot.com/', 'http://ws-dl.blogspot.com/']
9
10 def get_atom(uri):
11     try:
12         r = requests.get(uri)
13     except Exception, e:
14         return None
15     soup = BeautifulSoup(r.text)
16     links = soup.find_all('link', {'type': 'application/atom+xml'})
17     if links:
18         return str(links[0]['href'])
19     return None
20
21 def add_uri(uri, uris, outfile):
22     if uri and uri not in uris:
23         uris.add(uri)
24         outfile.write(uri + '\n')
25         print len(uris), uri
26
27 if __name__ == '__main__':
28     uris = set()
29     with open('blog_uris', 'a') as outfile:
30         if len(sys.argv) > 1 and sys.argv[1] == 'new':
31             for must_have in must_haves:
32                 uri = get_atom(must_have)
33                 add_uri(uri, uris, outfile)
34         else:
35             with open('blog_uris') as infile:
36                 [uris.add(line.strip()) for line in infile]
37     while len(uris) < 100:
38         uri = get_atom(default)
39         add_uri(uri, uris, outfile)
```

Listing 9: get_uris.py

```

1 import feedparser
2 import futures
3 import md5
4 import re
5 import sys
6 import json
7
8 def get_next(d):
9     for item in d.feed.links:
10         if item['rel'] == u'next':
11             return item['href']
12     return None
13
14 def getwords(text):
15     txt = re.compile(r'<[^>]+>').sub('', text)
16     words = re.compile(r'[A-Z^a-z]+').split(txt)
17     return [word.lower() for word in words if word != '']
18
19 def get_titles(uri):
20     print('processing {}'.format(uri))
21     next = uri
22     wc = {}
23     pages = 0
24     while next is not None:
25         d = feedparser.parse(next)
26         for e in d.entries:
27             words = getwords(e.title.encode('utf-8'))
28             for word in words:
29                 wc.setdefault(word, 0)
30                 wc[word] += 1
31         pages += 1
32         next = get_next(d)
33         print('next {}'.format(next))
34         title = d.feed.title.encode('utf-8')
35         subtitle = d.feed.subtitle[:50].encode('utf-8')
36         print('finished: {}: {}'.format(title, subtitle))
37     return uri, title, subtitle, pages, wc
38
39 if __name__ == '__main__':
40     with open('blog.uris') as infile:
41         uris = [line.strip() for line in infile]
42         if len(sys.argv) == 2 and sys.argv[1] == 'get':
43             with futures.ThreadPoolExecutor(max_workers=8) as executor:
44                 uri_futures = [executor.submit(get_titles, uri) for uri in uris]
45                 for future in futures.as_completed(uri_futures):
46                     uri, title, subtitle, pages, wc = future.result()
47                     with open('wcs/' + md5.new(uri).hexdigest(), 'w') as out:
48                         out.write(title + ': ' + subtitle + '\t' + str(pages) + '\t')
49                     json.dump(wc, out)
50         else:
51             apcount = {}
52             wordcounts = {}
53             pagecounts = {}
54             for uri in uris:
55                 with open('wcs/' + md5.new(uri).hexdigest()) as infile:
56                     try:
57                         lines = infile.read().split('\t')
58                         title = lines[0]
59                         pages = int(lines[1])
60                         wc = json.loads(lines[2])
61                     except Exception, e:
62                         print('*** {} generated an exception: {}'.format(uri, e))
63                         continue
64                     wordcounts[title] = wc
65                     pagecounts[title] = pages
66                     for word, count in wc.items():
67                         apcount.setdefault(word, 0)
68                         apcount[word] += count
69             wordlist = []
70             for w, bc in sorted(apcount.items(), key=lambda x: x[1], reverse=True):
71                 frac = float(bc) / len(uris)
72                 if frac > 0.1 and frac < 0.5:
73                     wordlist.append(w)
74             if len(sys.argv) == 2 and sys.argv[1] == 'pages':
75                 with open('pagecounts', 'w') as outfile:
76                     outfile.write('blog\tpages\n')

```

```

77         for blog, pagecount in pagecounts.iteritems():
78             outfile.write("\"" + blog.replace("\"", "") + "\"" + '\t' + str(
                pagecount) + '\n')
79 if len(sys.argv) == 2 and sys.argv[1] == 'wc':
80     with open('blogdata1.txt', 'w') as out:
81         out.write('Blog')
82         for word in wordlist[:500]:
83             out.write('\t%s' % word)
84         out.write('\n')
85         for blog, wc in wordcounts.items():
86             print blog
87             out.write(blog)
88             for word in wordlist[:500]:
89                 if word in wc:
90                     out.write('\t{}'.format(wc[word]))
91                 else: out.write('\t0')
92             out.write('\n')

```

Listing 10: matrix.py

7 References

- [1] Toby Segaran. *Programming Collective Intelligence*. O'Reilly, first edition, 2007.
- [2] Internet Engineering Task Force (IETF). RFC-4287 The Atom Syndication Format. <https://tools.ietf.org/html/rfc4287>, 2005.