Assignment 8

Fall 2014
CS595 Web Science
Dr. Michael Nelson

Mathew Chaney

November 26, 2014

Contents

Listings

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 obtain a basic dataset for this task a list of URIs was required for processing. This list was created using python `get_uris.py` script. 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 Programming Collective Intelligence book [1]. The `get_uris` `main` function in Listing 1 was the driver that called `get_atom` funciton from Listing 2 to extract the atom [2] formatted blog entries and if one existed for that particular blog it was added to the URIs set for storage with the `add_uri` function, shown in Listing 3.

```
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

```
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 2: 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 3: add_uri function

This code uses the value stored in the `default` variable as seed for obtaining random blogs. Using the BeautifulSoup library [3], the contents of the blog were parsed and if an atom feed was found that link was extracted and then saved as part of the set of URIs in the `blog_uris` file.

The contents of each blog were downloaded and processed by the code shown in Listing 4 and the `get_titles`, `get_words` and `get_next` functions found in Listing 5. This code loops over the URIs that were downloaded with the `get_uris.py` script, loops over each entry and extracts all the words in each entry's title. These words were 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 6.

```
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 4: looping over the URIs

```
def get_next(d):
    for item in d.feed.links:
        if item['rel'] == u'next':
            return item['href']
    return None

def getwords(text):
    txt = re.compile(r'<[^>]+>').sub('', text)
    words = re.compile(r'[^A-Z^a-z]+').split(txt)
    return [word.lower() for word in words if word != '']

def get_titles(uri):
    print('processing {}'.format(uri))
    next = uri
    wc = {}
    pages = 0
    while next is not None:
        d = feedparser.parse(next)
        for e in d.entries:
            words = getwords(e.title.encode('utf-8'))
            for word in words:
                wc.setdefault(word, 0)
                wc[word] += 1
        pages += 1
        next = get_next(d)
        print('next {}'.format(next))
    title = d.feed.title.encode('utf-8')
    subtitle = d.feed.subtitle[:50].encode('utf-8')
    print('finished: {}: {}'.format(title, subtitle))
    return uri, title, subtitle, pages, wc
```

Listing 5: processing each blog

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

Listing 6: bounding the terms

To build the histogram the `pagecounts` file was parsed by the R script in Listing 7 and saved as a pdf, which is shown in Figure 1.

```
#! /usr/bin/Rscript

data <- read.table("pagecounts", sep="\t", header=TRUE, comment.char="")
counts <- table(data$pages)
pdf("hist.pdf")
barplot(counts, ylab="Number of Blogs", xlab="Page Count", main="Page Count per Blog")
dev.off()
```

Listing 7: building the histogram

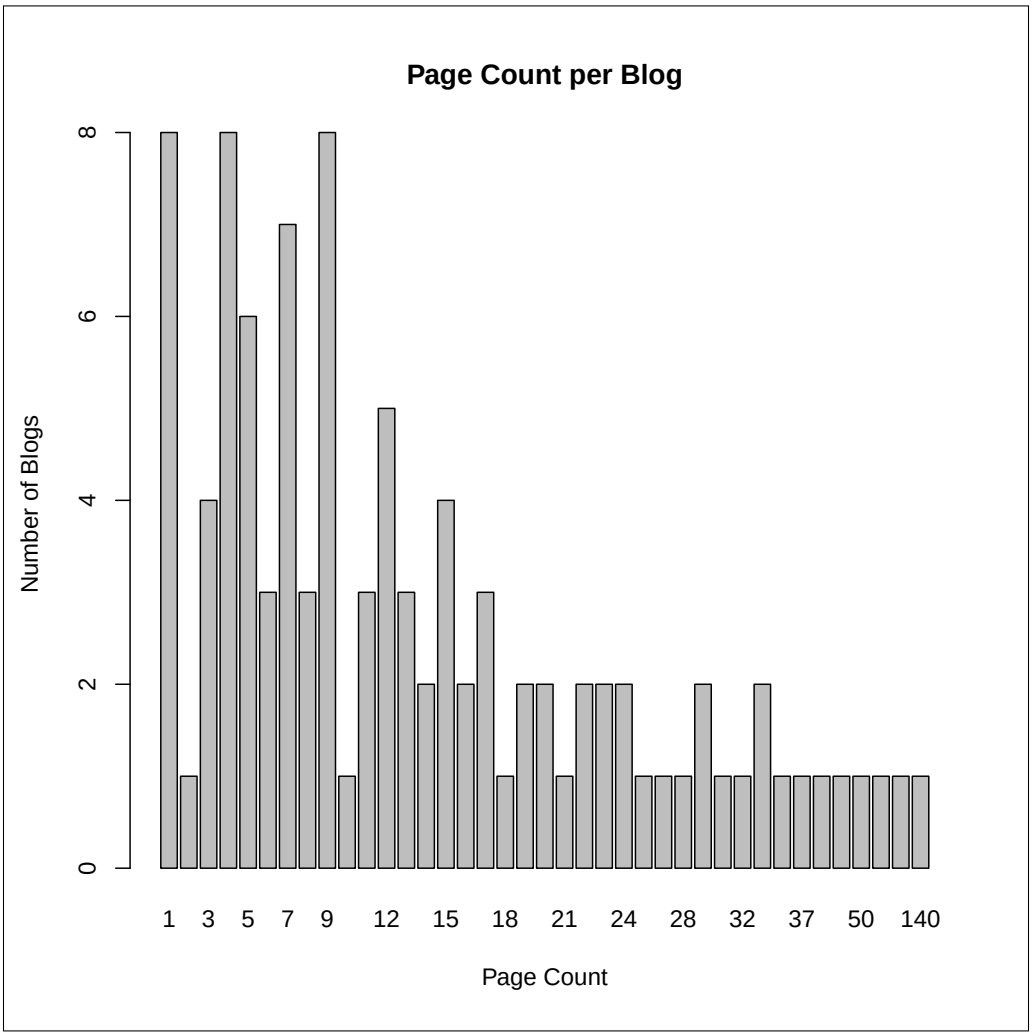

Figure 1: Page Count per Blog

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 interations 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

```
#! /usr/bin/env python

import requests
import sys
from bs4 import BeautifulSoup

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

def get_atom(uri):
    try:
        r = requests.get(uri)
    except Exception, e:
        return None
    soup = BeautifulSoup(r.text)
    links = soup.find_all('link', {'type':'application/atom+xml'})
    if links:
        return str(links[0]['href'])
    return None

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

if __name__ == '__main__':
    uris = set()
    with open('blog_uris', 'a') as outfile:
        if len(sys.argv) > 1 and sys.argv[1] == 'new':
            for must_have in must_haves:
                uri = get_atom(must_have)
                add_uri(uri, uris, outfile)
        else:
            with open('blog_uris') as infile:
                [uris.add(line.strip()) for line in infile]
        while len(uris) < 100:
            uri = get_atom(default)
            add_uri(uri, uris, outfile)
```

Listing 8: get_uris.py

```
import feedparser
import futures
import md5
import re
import sys
import json

def get_next(d):
    for item in d.feed.links:
        if item['rel'] == u'next':
            return item['href']
    return None

def getwords(text):
    txt = re.compile(r'<[^>]+>').sub('', text)
    words = re.compile(r'[^A-Z^a-z]+').split(txt)
    return [word.lower() for word in words if word != '']

def get_titles(uri):
    print('processing {}'.format(uri))
    next = uri
    wc = {}
    pages = 0
    while next is not None:
        d = feedparser.parse(next)
        for e in d.entries:
            words = getwords(e.title.encode('utf-8'))
            for word in words:
                wc.setdefault(word, 0)
                wc[word] += 1
        pages += 1
        next = get_next(d)
        print('next {}'.format(next))
    title = d.feed.title.encode('utf-8')
    subtitle = d.feed.subtitle[:50].encode('utf-8')
    print('finished: {}: {}'.format(title, subtitle))
    return uri, title, subtitle, pages, wc

if __name__ == '__main__':
    with open('blog_uris') as infile:
        uris = [line.strip() for line in infile]
    if len(sys.argv) == 2 and sys.argv[1] == 'get':
        with futures.ThreadPoolExecutor(max_workers=8) as executor:
            uri_futures = [executor.submit(get_titles, uri) for uri in uris]
            for future in futures.as_completed(uri_futures):
                uri, title, subtitle, pages, wc = future.result()
                with open('wcs/' + md5.new(uri).hexdigest(), 'w') as out:
                    out.write(title + ': ' + subtitle + '\t' + str(pages) + '\t')
                    json.dump(wc, out)
    else:
        apcount = {}
        wordcounts = {}
        pagecounts = {}
        for uri in uris:
            with open('wcs/' + md5.new(uri).hexdigest()) as infile:
                try:
                    lines = infile.read().split('\t')
                    title = lines[0]
                    pages = int(lines[1])
                    wc = json.loads(lines[2])
                except Exception, e:
                    print('*** {} generated an exception: {}'.format(uri, e))
                    continue
            wordcounts[title] = wc
            pagecounts[title] = pages
            for word, count in wc.items():
                apcount.setdefault(word, 0)
                apcount[word] += count
        wordlist = []
        for w, bc in sorted(apcount.items(), key=lambda x: x[1], reverse=True):
            frac = float(bc) / len(uris)
            if frac > 0.1 and frac < 0.5:
                wordlist.append(w)
        if len(sys.argv) == 2 and sys.argv[1] == 'pages':
            with open('pagecounts', 'w') as outfile:
                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 9: 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.

[3] Leonard Richardson. Beautiful Soup. http://www.crummy.com/software/BeautifulSoup/, 2014.