Introduction to Web Science/595: Assignment #9

Dr. Nelson

Francis Pruter

Thursday, 04 Decemeber, 2014

Contents

Problem 1

```
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).
```

SOLUTION
First, I downloaded 100 URIs to include the 2 required URIs. Below is the code used:

Listing 1: getBlogs.py

```python
#!/usr/bin/python
import requests

next_blog = "https://www.blogger.com/next-blog?navBar=true&blogID=3471633091411211117"

ATOMlink = "/feeds/posts/default"

blogs=[]

blogs.append("http://f-measure.blogspot.com/")
blogs.append("http://ws-dl.blogspot.com/")

#get 100 unique blogs
while len(blogs) < 100:
    try:
      #verify blog URI still works
      r = requests.get(next_blog)
      if r.status_code == 200:
        if not r.url in blogs:
          blogs.append(r.url[:len(r.url)-17])
          print len(blogs)
    except requests.ConnectionError:
      pass

with open("blogURI", "w+") as fout:
    for uri in blogs:
      fout.write(uri+'\n')
```

Problem 1 continued on next page. . .

Next, I utlized and modified generatefeedvector.py [1] provided from the book. I modified teh getwordcounts function to include a loop to ensure I gather the words from every page in the blog and keep a count of how many pages and returns it. Lines 38-42 is a loop that get the links from the website and looks for "rel=next". If found, there is a next page and gets the words from the next page.

Listing 2: getBlogs.py

```
   def getwordcounts(url):
10   '''
     Returns title and dictionary of word counts for an RSS feed
   <link rel='next' type='application/atom+xml' href='http://www.blogger.com/feeds
       /8028607259735637511/posts/default?start-index=26&amp;max-results=25'/>
     '''
     wc = {}
15   next = True
     pagecount = 0

   #continue until there are no next page
     while next:
20     pagecount+=1
       next = False
       # Parse the feed
       d = feedparser.parse(url)
       # Loop over all the entries
25     for e in d.entries:
           if 'summary' in e:
               summary = e.summary
           else:
               summary = e.description
30
           # Extract a list of words
           words = getwords(e.title + ' ' + summary)
           for word in words:
               wc.setdefault(word, 0)
35             wc[word] += 1

   #look for next page
       for link in  d['feed']['links']:
         if link['rel'] == 'next':
40         url = link['href']
           #print url
           next = True

     return (d.feed.title, wc, pagecount)
```

After that, I modified lines 84-87 to 89-92. The modification gathers the 500 most "popular" terms.

Listing 3: getBlogs.py

```

   #for (w, bc) in apcount.items():
85 #    frac = float(bc) / len(feedlist)
   #    if frac > 0.1 and frac < 0.5:
   #        wordlist.append(w)

```

```
for w, bc in sorted(apcount.items(), key=lambda kv: kv[1], reverse=True):
    frac = float(bc) / len(feedlist)
    if frac > 0.1 and frac < 0.5 and len(wordlist) < 500:
        wordlist.append(w)
```

Below is a histogram of how many pages each blog has:

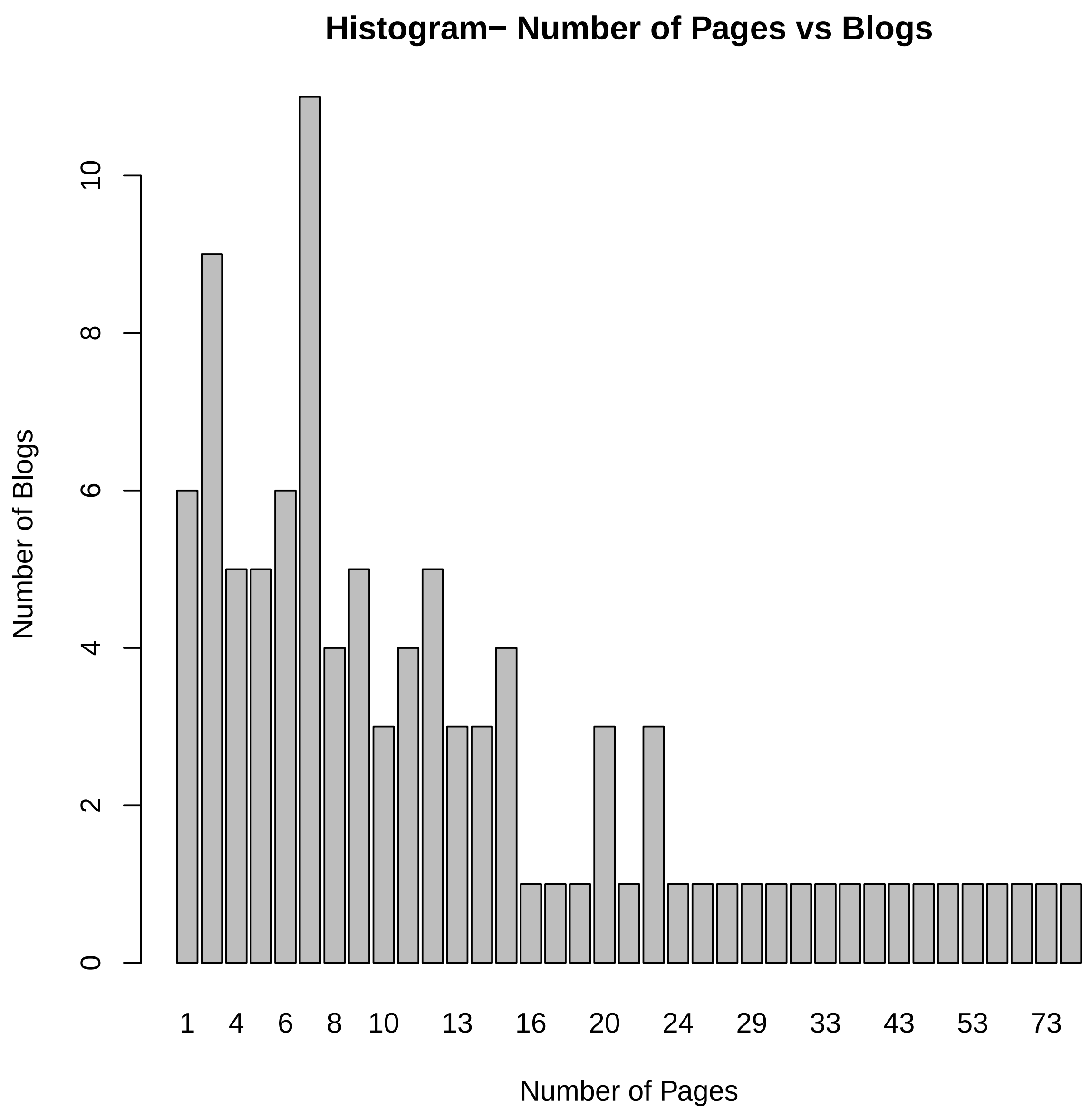

Figure 1: Hist.pdf

Problem 2

```
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).
```

SOLUTION
I used the code from class and clusters.py [1] from the book.

Listing 4: drawdendrogram.py

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

import clusters
blognames, words, data = clusters.readfile('blogdata1.txt')
clust=clusters.hcluster(data)

#Question 2
clusters.printclust(clust, labels=blognames)
clusters.drawdendrogram(clust, blognames, jpeg='dengrogram.jpg')
```

Figure 2 on the next page is the JPEG created. The ASCII dendrogram is

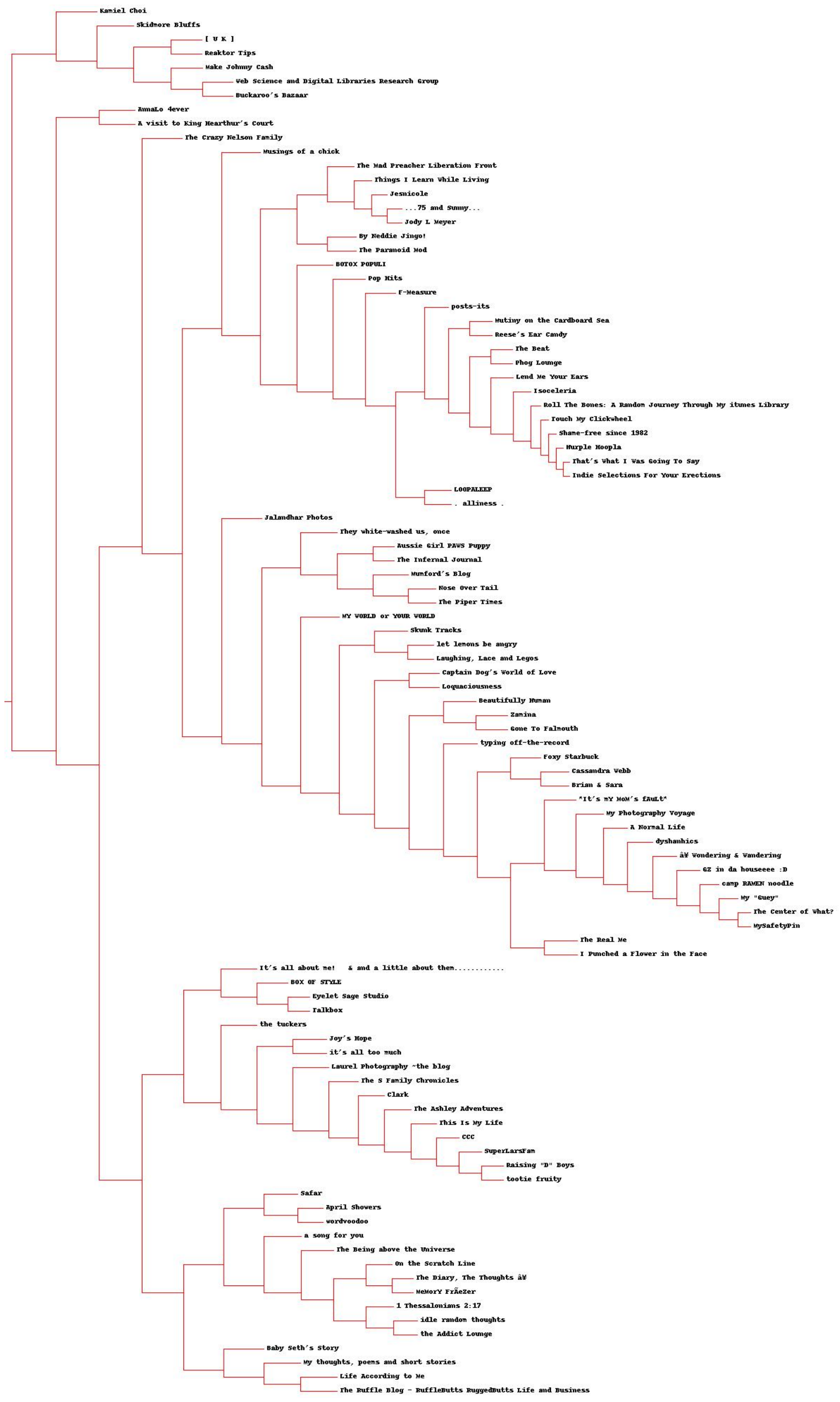

Figure 2: Dengrogram.jpg

Problem 3

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

SOLUTION

I used the code from class and clusters.py from the book.

K	Iterations
5	4
10	5
20	5

Below is the code used:

Listing 5: drawdendrogram.py

```
#Question 3
print "K = 5"
kclust5 = clusters.kcluster(data, k=5)
print "\nK = 10"
kclust10 = clusters.kcluster(data, k=10)
print "\nK = 20"
kclust20 = clusters.kcluster(data, k=20)
```

I redirected the out output to “kclust.txt” using:

```
python clusters.py > kclust.txt
```

Problem 4

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

SOLUTION

Below is the code used and JPEG created after 212 iterations:

Listing 6: drawdendrogram.py

```python
coords=clusters.scaledown(data)
clusters.draw2d(coords, blognames, jpeg='MDS.jpg')
```

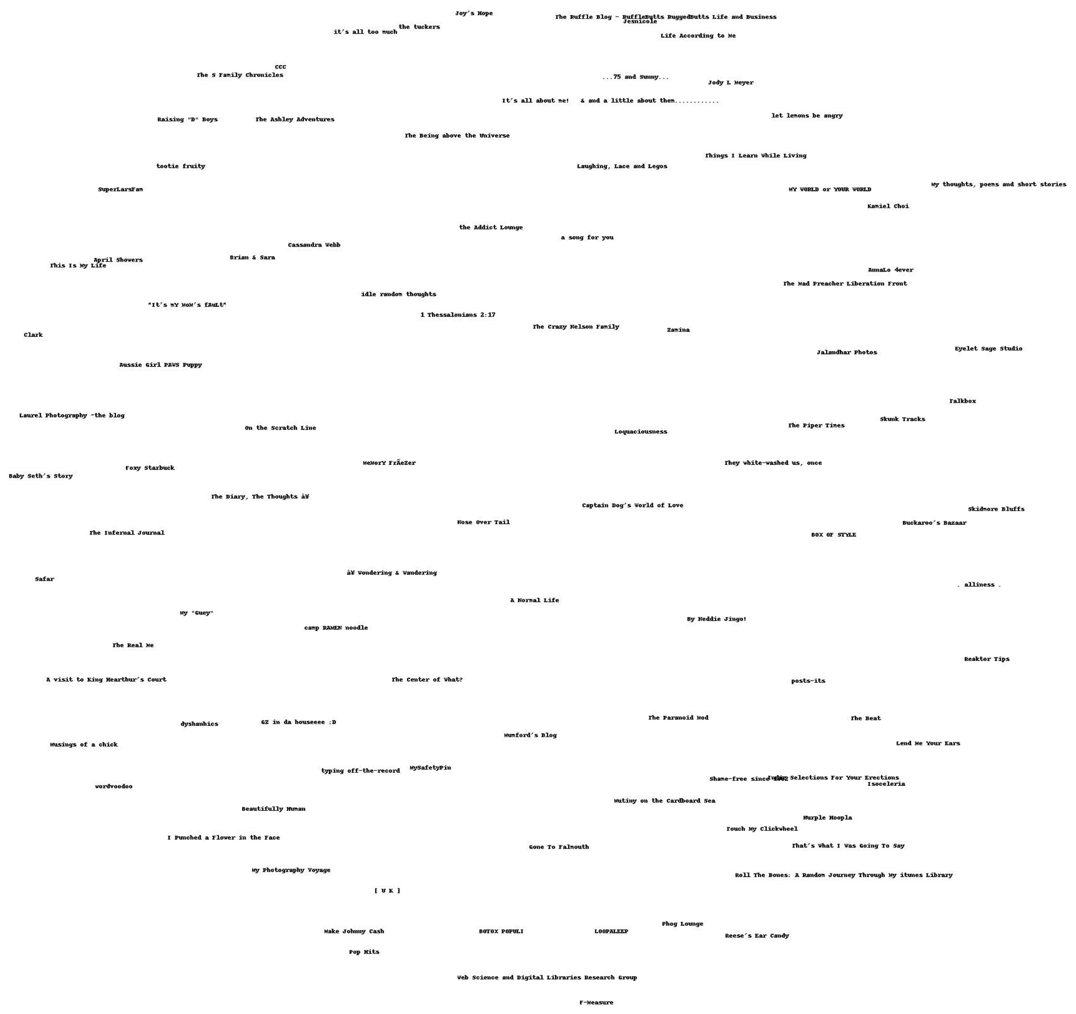

Figure 3: MDS.jpg

References

[1] K. Arthur Endsley. Programming-collective-intelligence. https://github.com/arthur-e/Programming-Collective-Intelligence/blob/master/chapter3/, 2012.