# Web Science: Assignment #7

*Alexander Nwala*

**Apurva Modi**

Wednesday, May 1, 2019

# Contents

**Problem 1** **3**

**Problem 2** **8**

**Problem 3** **10**

**Problem 4** **11**

# 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. Note that this method randomly chooses blogs and each student will separately do this process, so it is unlikely that these 98 blogs will be shared among students. In other words, no sharing of blog data. Upload to github your code for grabbing the blogs and provide a list of blog URIs, both in the report and in github.

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) 1000 terms, this is *after* the criteria on p. 32 (slide 8) has been satisfied. Remember that blogs are paginated.

**SOLUTION :**

1. I have used the following link to extract the blogs

   ```
   http://www.blogger.com/profile-find.g?t=m&q= + query (which is food in my case)
   ```

2. Created a python script **1/Assignment7_1.py** to crawl and get 104 blog URLs related to food.

3. Added the given two blogs in to the file blogList.txt.

4. Created the python script **1/blogData.py** to iterate through the URLs from the Outputs/blogList.txtl

5. Finally, the same script **1/blogData.py** is used to create the **blogdata.txt** showing the blog matrix for extracted blogs.

Listing 1: Assignment7_1.py

```
 import os
import requests
from bs4 import BeautifulSoup
from bs4.element import Comment

def fetch_blogs(query, limit):
    list_blogger_profiles = []
    list_blogger_urls = []
    output_directory = "Outputs/"
    blogger_url = "http://www.blogger.com/profile-find.g?t=m&q=" + query
    if not os.path.isdir(output_directory):
        os.mkdir(output_directory)
    while len(list_blogger_urls) < limit:
        response = requests.get(blogger_url)
        print(blogger_url)
```

```
        if response.status_code == 200:
            soup = BeautifulSoup(response.text, 'html.parser')
            profiles_tag_selector = soup.find_all('dl')
            for tags in profiles_tag_selector:
                profile_url_selector = tags.find('a', href=True)
                blogger_profile_url = "http://www.blogger.com/" + profile_url_selector['href'][1:]
                list_blogger_profiles.append(blogger_profile_url)
            list_blogger_urls.extend(fetch_blog_urls(list_blogger_profiles, limit))
            print(list_blogger_urls)
            list_blogger_profiles = []
            blogger_url_selector = soup.find('a', {'id': 'next-btn'},
                                             href=True)
            if blogger_url_selector:
                blogger_url = blogger_url_selector['href']
            else:
                break
            print(len(list_blogger_urls))
    print(len(list_blogger_urls))
    if os.path.isfile(output_directory + "test1.txt"):
        file_blog_urls = open(output_directory + "blogList.txt", "a+")
    else:
        file_blog_urls = open(output_directory + "blogList.txt", "w")
    for urls in list_blogger_urls:
        file_blog_urls.write(urls + "\n")
    file_blog_urls.close()

def fetch_blog_urls(list_blogger_profiles, limit):
    count = 0
    list_blogger_urls = []
    for profile in list_blogger_profiles:
        if count < limit:
            response = requests.get(profile)
            if response.status_code == 200:
                soup = BeautifulSoup(response.text, 'html.parser')
                profile_blog_link_selector = soup.find_all('span', dir=True)
                for profile_blog_links in profile_blog_link_selector:
                    if profile_blog_links['dir'] == 'ltr':
                        blog_links_selector = profile_blog_links.find_all('a', href=True)
                        for blog_links in blog_links_selector:
                            print(blog_links['href'])
                            try:
                                response = requests.head(blog_links['href'], timeout=60)
                                print(response.headers)
                                if response.encoding and (response.encoding == 'ISO-8859-1' or respon
                                    list_blogger_urls.append(blog_links['href'])
                                    print()
                                    count += 1
                            except Exception as e:
                                print(e)
    print(list_blogger_urls)
    return list_blogger_urls

fetch_blogs("Food", 100)
```

```
file_open = open("Outputs/blogList.txt", "a+")
file_open.write("http://f-measure.blogspot.com/" + "\n")
file_open.write("http://ws-dl.blogspot.com/" + "\n")
file_open.close()
```

Listing 2: blogData.py

```

import feedparser
import re
import urllib.parse
import requests
from bs4 import BeautifulSoup as bs4

def getwordcounts(url):
    ’’’
    Returns title and dictionary of word counts for an RSS feed
    ’’’
    # Parse the feed
    d = feedparser.parse(url)
    wc = {}
    # Loop over all the entries
    for e in d.entries:
        if ’summary’ in e:
            summary = e.summary
        else:
            summary = e.description
        # Extract a list of words
        words = getwords(e.title + ’ ’ + summary)
        for word in words:
            wc.setdefault(word, 0)
            wc[word] += 1
    return (d.feed.title, wc)


def getwords(html):
    # Remove all the HTML tags
    txt = re.compile(r’<[^>]+>’).sub(’’, html)

    # Split words by all non-alpha characters
    words = re.compile(r’[^A-Z^a-z]+’).split(txt)

    # Convert to lowercase
    return [word.lower() for word in words if word != ’’]

def findfeed(site):
    raw = requests.get(site).text
    result = []
    possible_feeds = []
    html = bs4(raw)
    feed_urls = html.findAll("link", rel="alternate")
    for f in feed_urls:
        t = f.get("type",None)
        if t:
            if "rss" in t or "xml" in t:
                href = f.get("href",None)
                if href:
                    possible_feeds.append(href)
    parsed_url = urllib.parse.urlparse(site)
    base = parsed_url.scheme +"://"+parsed_url.hostname
    atags = html.findAll("a")
    for a in atags:
        href = a.get("href",None)
        if href:
            if "xml" in href or "rss" in href or "feed" in href:
                possible_feeds.append(base+href)
    for url in list(set(possible_feeds)):
        f = feedparser.parse(url)
        if len(f.entries) > 0:
            if url not in result:
                result.append(url)
    return(result)

apcount = {}
wordcounts = {}
feedlist = [line for line in open("Outputs/blogList.txt")]
#feedlistLen = 1
for feedurl in feedlist:
    try:
        rssFeedUrl = findfeed(feedurl)
        (title, wc) = getwordcounts(rssFeedUrl[0])
        #(title, wc) = getwordcounts(feedurl)
        wordcounts[title] = wc
        for (word, count) in wc.items():
            apcount.setdefault(word, 0)
            if count > 1:
                apcount[word] += 1
    except:
        print('Failed to parse feed %s' % feedurl)

wordlist = []
for (w, bc) in apcount.items():
    frac = float(bc) / len(feedlist)
    #frac = float(bc) / feedlistLen
    if frac > 0.1 and frac < 0.5:
        wordlist.append(w)
out = open("blogdata.txt", 'w')
out.write('Blog')
for word in wordlist:
    out.write('\t%s' % word)
out.write('\n')
for (blog, wc) in wordcounts.items():
    print(blog)
    out.write(blog)
    for word in wordlist:
        if word in wc:
            out.write('\t%d' % wc[word])
        else:
            out.write('\t0')
    out.write('\n')
out.close()
```

# Problem 2

Create an ASCII and JPEG dendrogram that clusters (i.e., HAC) the most similar blogs ;see slides 13 and 14. Include the JPEG in your report and upload the ascii file to github (it will be too unwieldy for inclusion in the report).
**SOLUTION :**

1. I coded a python script **2/denodo.py**, to call the methods from **clusters.py**
2. The script creates a ASCII text file **ASCII** and draws a dendogram **Dendrogram.jpg**
3. I made use of the tutorial and code available on Github under Programming Collective Intelligence.

Listing 3: denodo.py

```

import clusters
import sys


def createDendrogram():
    blogs, colnames, data = clusters.readfile('blogdata.txt')
    cluster = clusters.hcluster(data)
    clusters.drawdendrogram(cluster, blogs, jpeg='Dendrogram.jpg')
    f = open("ASCII.txt", 'w')
    sys.stdout = f
    clusters.printclust(cluster, labels=blogs)
    f.close()
    sys.stderr.close()


if __name__ == "__main__":
    createDendrogram()
```

The below picture displays the generated dendogram.

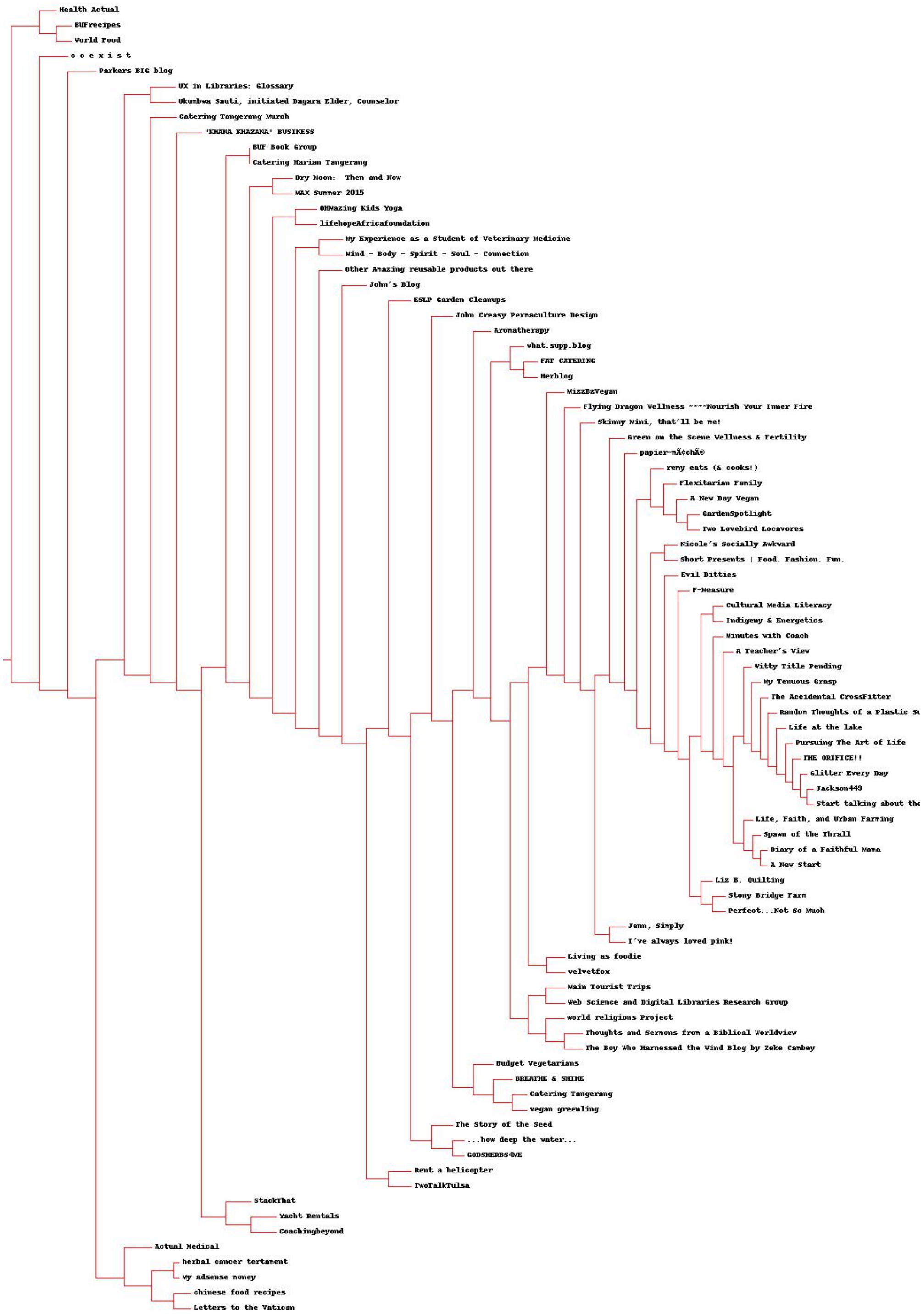


Figure 1: Dendogram

# Problem 3

Cluster the blogs using K-Means, using k=5,10,20. (see slide 25). Print the values in each centroid, for each value of k. How many iterations were required for each value of k?
**SOLUTION :**

1. I coded a python script **kmeanclusturing.py** to call the methods from **clusters.py**
2. I also made use of the tutorial and code available on Github under Programming Collective Intelligence.

Listing 4: kmeanclusturing.py

```
import clusters
def kMean():
    kMeanValues = [5, 10, 20]
    blogs, colnames, data = clusters.readfile(’blogdata.txt’)
    for i in kMeanValues:

        kclust, itercount = clusters.kcluster(data, k=i)
        print(kclust)
        f = open("kclust_%d.txt" % i, ’w’)
        f.write("Total Number Of Iterations: %d \n" % itercount)
        print(len(kclust))
        clusterCount = 1
        for cluster in kclust:
            i=1
            f.write("---\n")
            f.write("Cluster %d \n" % clusterCount)
            for blogid in cluster:
                f.write(str(i)+".\t"+blogs[blogid] + "\n")
                i+=1
            f.write("\n")
            clusterCount+=1
if __name__ == "__main__":
    kMean()
```

Results have been uploaded in to GitHub under foler **3/kclust_5.txt,3/kclust_10.txt and 3/kclust_20.txt**. It took 11 iterations to create 5 clusters, 5 iterations for 10 cluster and 4 iterations for 20 clusters.

# Problem 4

Use MDS to create a JPEG of the blogs similar to slide 29 of the week 11 lecture. How many iterations were required?

**SOLUTION :**

Created a python script **4/mdsIteration.py**

Listing 5: 4/mdsIteration.py

```
import clusters



blognames, words, data = clusters.readfile('blogdata.txt')
coords, itercount = clusters.scaledown(data)
clusters.draw2d(coords, labels=blognames, jpeg='mds.jpg')
print ('Iteration count: %d' % itercount)
```

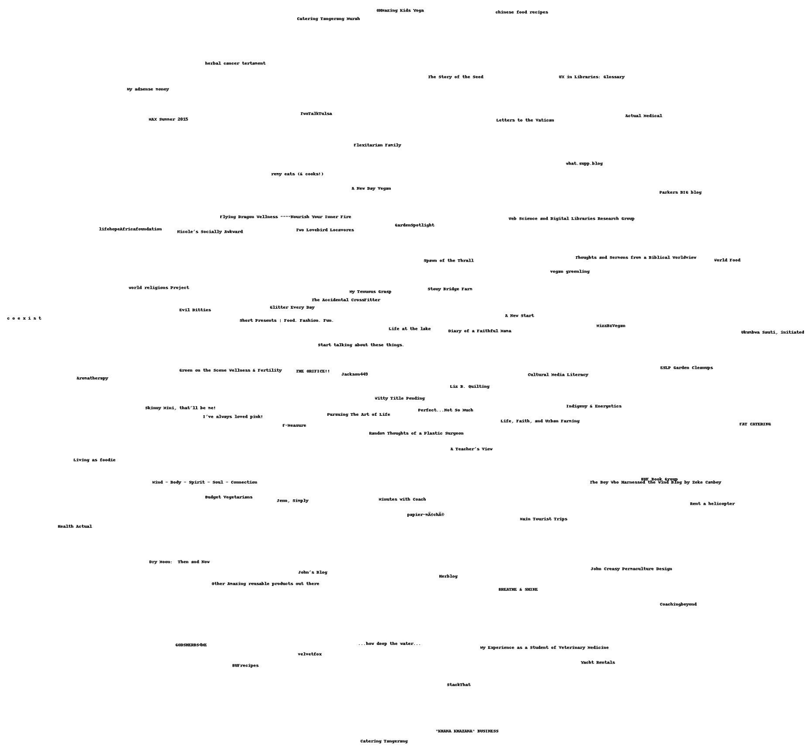
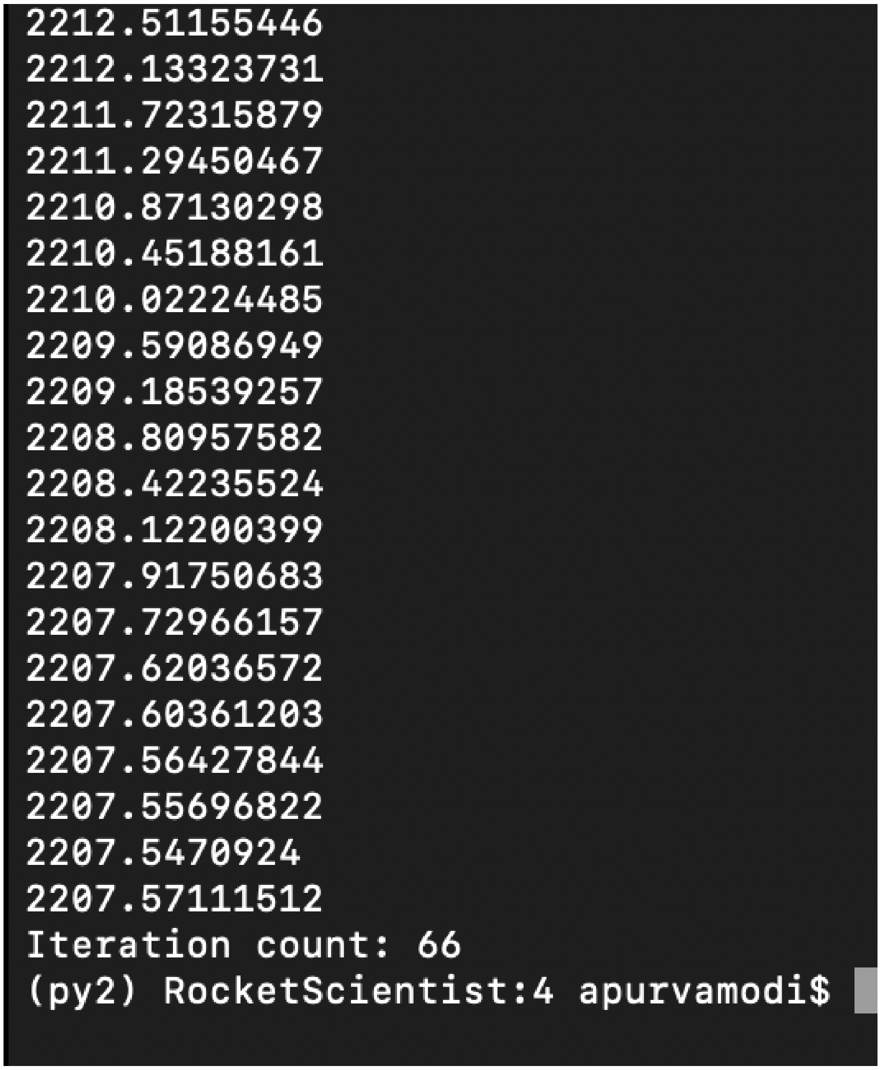


Figure 2: MDS

Result was generated with 66 iterations.



```
2212.51155446
2212.13323731
2211.72315879
2211.29450467
2210.87130298
2210.45188161
2210.02224485
2209.59086949
2209.18539257
2208.80957582
2208.42235524
2208.12200399
2207.91750683
2207.72966157
2207.62036572
2207.60361203
2207.56427844
2207.55696822
2207.5470924
2207.57111512
Iteration count: 66
(py2) RocketScientist:4 apurvamodi$
```

Figure 3: INumber of iterations

## References

1. https://github.com/arthur-e/Programming-Collective-Intelligence/tree/master/chapter3

2. https://en.wikipedia.org/wiki/Cluster_analysis

3. https://en.wikipedia.org/wiki/Dendrogram

4. https://en.wikipedia.org/wiki/K-means_clustering