Web Science: Assignment #7

Alexander Nwala

Puneeth Bikkasandra

Tuesday, May 1, 2018

Contents

Problem 1 **3**

Problem 2 **8**

Problem 3 **10**

Problem 4 **12**

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 :

I have solved the problem as described in the below steps :

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

   `http://www.blogger.com/next-blog?navBar=true&blogID=3471633091411211117`

2. Created a python script **fetchBlogs.py** to crawl and get 100 blog URLs
3. Added the given two blogs in to the file.
4. Extracted the raw Html content and saved in "filename"–"blog link" format in a text file **blogListFile**
5. Created the python script **fetchFeed.py** to iterate through the raw Html content to get the feed URLs and save in to another text file **feedsList**
6. Finally, script **generateFeedVector.py** is used to create the text **blogdata** showing the blog matrix for 100 blogs.

Listing 1: fetchBlogs.py

```
import requests
blogLinks = []

blogLinks.append('http://f-measure.blogspot.com')
blogLinks.append('http://ws-dl.blogspot.com')

def get100BlogLinks():
    while (len(blogLinks) < 105) :
        try:
            url="http://www.blogger.com/next-blog?navBar=true&
            blogID=3471633091411211117"
            request = requests.get(url)
            url = request.url.strip('?expref=next-blog/')
```

Problem 1 continued on next page...

```
                if url not in blogLinks:
                    blogLinks.append(url)
            except:
                pass

    return    blogLinks

def saveRawHTML(blogs):
    blogsListFile = open('blogListFile.txt','w')
    i = 0
    for url in blogs:
        request = requests.get(url)
        rawHtmlData = request.content
        file = open('rawFiles/rawHtml%s.html' % str(i+1), 'w')
        file.write(rawHtmlData)
        file.close()
        blogsListFile.write('rawHtml%s.html' % str(i+1)+"--"+url+" \n")
        i = i + 1

if __name__ == '__main__':
    print("Length :",str(len(get100BlogLinks())))
    saveRawHTML(get100BlogLinks())
```

Below code fetches the feed from the raw Html content.

Listing 2: fetchFeed.py

```
import requests
from bs4 import BeautifulSoup as bs4
import os
blogLinks = []

def getFeed(filename):
        try:
            with open("rawFiles/" + filename) as file:
                raw = file.read()
                html = bs4(raw,"html.parser")
                feed_url = html.findAll("link",
                 rel="alternate",type="application/atom+xml")
                blogLink = feed_url[0]['href']
                print("blogLink:",blogLink)
            return blogLink

        except:
            print("feed_url",feed_url)
            pass

if __name__ == "__main__":

    with open("blogListFile.txt") as f:
        for line in f:
            print("Line",line)
            fileName = line.split('--')[0]
            print("File",fileName)
            blog = getFeed(fileName)
```

```
                print("Blog:",blog)
                blogLinks.append(blog)

    with open('feeds/feedsList.txt','w') as file:
        for link in blogLinks:
                file.write(str(link))
                file.write("\n")
```

Listing 3: generateFeedVector.py

```
  #!/usr/bin/python
# -*- coding: utf-8 -*-
import feedparser
import re

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 != '']


apcount = {}
wordcounts = {}
withoutFeed = 0
feedlist = [line for line in open('feeds/feedsList.txt')]
for feedurl in feedlist:
    try:
        (title, wc) = getwordcounts(feedurl)
        wordcounts[title] = wc
        for (word, count) in wc.items():
            apcount.setdefault(word, 0)
            if count > 1:
```

```
                apcount[word] += 1
    except:
        withoutFeed = withoutFeed + 1
        print('Failed to parse feed', feedurl)

wordlist = []
for (w, bc) in apcount.items():
    frac = float(bc) / len(feedlist)
    if frac > 0.1 and frac < 0.5:
        wordlist.append(w)
    if len(wordlist) >= 1000:
        break

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("Title: ",blog)
    out.write(blog.encode('utf-8'))
    for word in wordlist:
        if word in wc:
            out.write('\t%d' % wc[word])
        else:
            out.write('\t0')
    out.write('\n')
print("With No Feed:",withoutFeed)
```

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 :

I have solved the problem as described in the below steps :

1. Wrote a python script **createDendogram.py**, to call the methods from **clusters.py**
2. The script creates a ASCII text file **ASCII** and draws a dendogram **Dendrogram.jpg**

Listing 4: createDendogram.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 decpicts the generated dendogram.

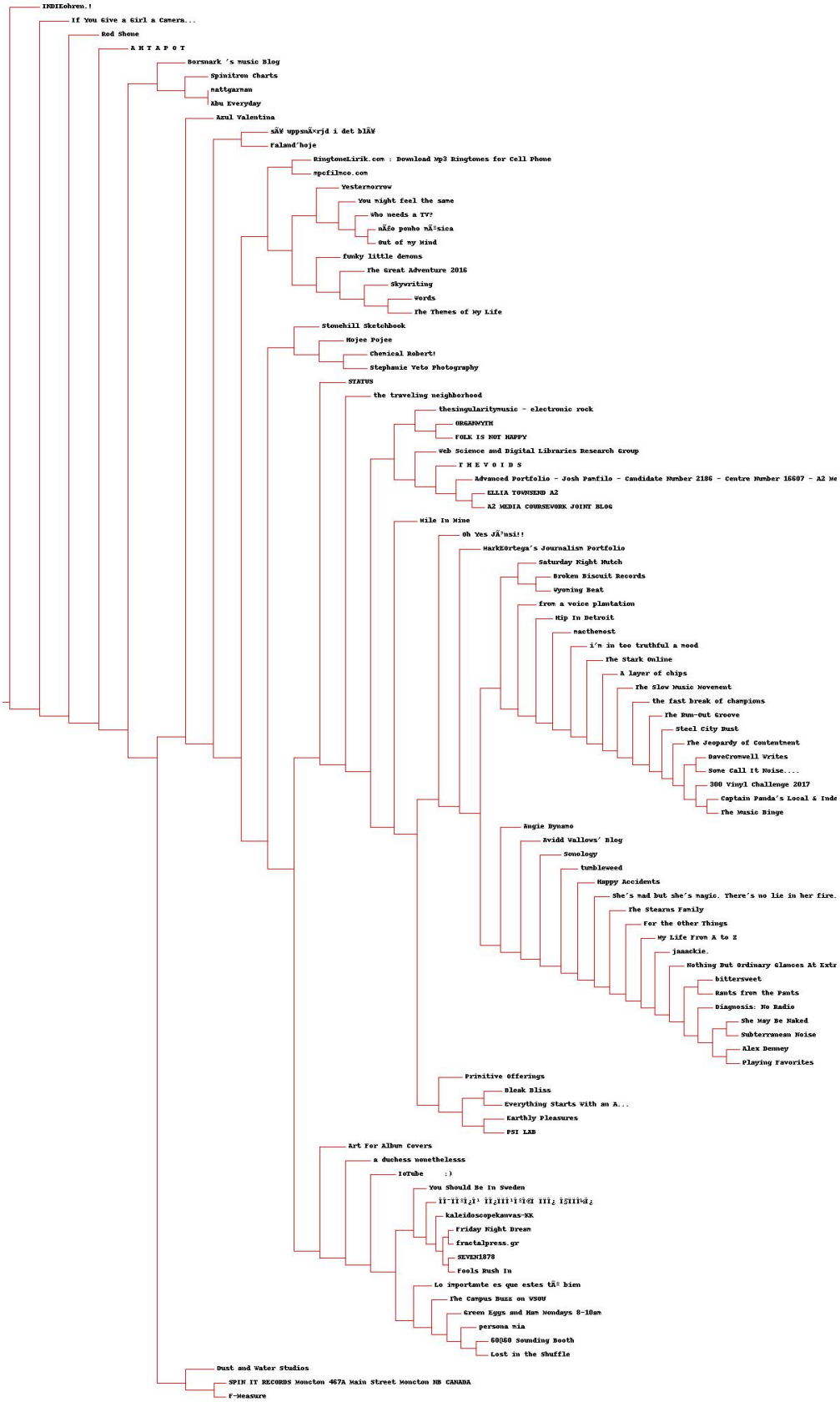

Figure 1: Dendogram

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 :

I have solved the problem as described in the below steps :

1. Wrote a python script **k5.py, k10.py** and **k20.py**, to call the methods from **clusters.py**
2. The script creates a ASCII text file **ASCII** and draws a dendogram **Dendrogram.jpg**

Listing 5: k5.py

```
import clusters

blogs,words,data=clusters.readfile('blogdata.txt')
kclust=clusters.kcluster(data,k=5)
print([blogs[item] for item in kclust[0]])
print([blogs[item] for item in kclust[1]])
print([blogs[item] for item in kclust[2]])
print([blogs[item] for item in kclust[3]])
print([blogs[item] for item in kclust[4]])
```

K5 required 8 iterations.

Listing 6: k10.py

```
import clusters

blogs,words,data=clusters.readfile('blogdata.txt')
kclust=clusters.kcluster(data,k=10)
print([blogs[item] for item in kclust[0]])
print([blogs[item] for item in kclust[1]])
print([blogs[item] for item in kclust[2]])
print([blogs[item] for item in kclust[3]])
print([blogs[item] for item in kclust[4]])
print([blogs[item] for item in kclust[5]])
print([blogs[item] for item in kclust[6]])
print([blogs[item] for item in kclust[7]])
print([blogs[item] for item in kclust[8]])
print([blogs[item] for item in kclust[9]])
```

k10 required 5 iterations

Listing 7: k20.py

```
import clusters

blog,words,data=clusters.readfile('blogdata.txt')
kclust=clusters.kcluster(data,k=20)
print([blog[item] for item in kclust[0]])
print([blog[item] for item in kclust[1]])
print([blog[item] for item in kclust[2]])
print([blog[item] for item in kclust[3]])
print([blog[item] for item in kclust[4]])
```

Problem 3 continued on next page...

```
print([blog[item] for item in kclust[5]])
print([blog[item] for item in kclust[6]])
print([blog[item] for item in kclust[7]])
print([blog[item] for item in kclust[8]])
print([blog[item] for item in kclust[9]])
print([blog[item] for item in kclust[10]])
print([blog[item] for item in kclust[11]])
print([blog[item] for item in kclust[12]])
print([blog[item] for item in kclust[13]])
print([blog[item] for item in kclust[14]])
print([blog[item] for item in kclust[15]])
print([blog[item] for item in kclust[16]])
print([blog[item] for item in kclust[17]])
print([blog[item] for item in kclust[18]])
print([blog[item] for item in kclust[19]])
```

K20 required 9 iterations
Results have been uploaded in to GitHub

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 **mdsJpeg.py**

Listing 8: k20.py

```
import clusters

blog,words,data=clusters.readfile('blogdata.txt')
coords=clusters.scaledown(data)
clusters.draw2d(coords,blog,jpeg='blogsMDS.jpg')
```

Result was generated with 44 iterations, output has been uploaded to github as text **mdsJPEG_Result** file. The below JPG diagram is generated

Figure 2: MDS

References

1. https://github.com/arthur-e/Programming-Collective-Intelligence/tree/master/chapter3
2. https://en.wikipedia.org/wiki/Dendrogram
3. https://en.wikipedia.org/wiki/K-means_clustering
4. https://en.wikipedia.org/wiki/Cluster_analysis