Assignment 8

CS 532: Introduction to Web Science

Spring 2018

Chandrasekhar Reddy Muthyala

1

Question

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 7) has been satisfied. Remember that blogs are paginated.

Answer

To solve this problem I decided to write a code in python language called **getBlogs.py** to get the 100 unique blogs.

Libraries used:

- import requests

I wrote a **getUniqueBlogs** function to retrieve 100 unique blogs. Inside that function I am creating a **100BlogUrls.txt** text file to store 100 blogs. To collect unique element in python we have one data type called **set** and I used sets to collect my 100 unique blogs. For getting blog I am hitting this URL http://www.blogger.com/next-blog?navBar=trueblogID=3471633091411211117 using **requests** library and this is looping 100 time through while and storing in **uniqueBlogs** variable. And after collecting 100 blogs, I added two URL given in the question to **uniqueBlogs** variable. The reason for collecting 100 blogs instead of 98 is sometimes few blogs will be having with no type called application/atom+xml. For safety reasons I am collecting 100 blogs.

After collecting 100 unique blogs in **uniqueBlogs** variable. I am getting raw HTML for each and every blog in uniqueBlogs . All raw HTML are storing in **blogs** folder. I saved blog id and the URI found inside a file called **100BlogUrls.txt**.

Run on the command line

```
python getBlogs.py
```

```
1  import requests
2
3
4  def getUniqueBlogs():
5          file = open('100BlogUrls.txt','w')
6          uniqueBlogs = set()
7          while (len(uniqueBlogs) < 100) :
8
9                  try:
10                         url="http://www.blogger.com/next-blog?
                               navBar=true&blogID
                               =3471633091411211117"
11                         request = requests.get(url)
12                         final_url = request.url.strip('?expref=
                               next-blog/')
13                         uniqueBlogs.add(final_url)
```

```
                        # print final_url
                        print len(uniqueBlogs)
                except:
                        pass
        uniqueBlogs.add('http://f-measure.blogspot.com')
        uniqueBlogs.add('http://ws-dl.blogspot.com')
        uniqueBlogsList = list(uniqueBlogs)
        for i in range(0,len(uniqueBlogsList)):
                request = requests.get(uniqueBlogsList[i])
                rawhtml = request.content
                print uniqueBlogsList[i]
                #storing raw html content of 120 blogs in blogs
                    in a sequence wise like 1.html, 2.html....
                rawf = open('blogs/%s.html' % str(i+1), 'w')
                rawf.write(rawhtml)
                # writing squence number of a blog and url of
                    the blog in 100BlogUrls.txt file
                # For Example: 1.html<\t><blog_url_name>
                string = str(i+1)+".html        "+
                    uniqueBlogsList[i]+"\n"
                file.write(string)

if __name__ == '__main__':
        getUniqueBlogs()
```

Listing 1: Python script to get 100 unique blogs

Once this completed I had 100 unique blogs with raw HTML. I then wrote a script in python called **getFeed.py** as shown in Listing **??** which parsed each html document saved using the library Beautiful soup which allowed me to search the document by an HTML 'link' element to find the atom+xml feed of the blog [2]. I saved these feeds to a file called **feedList.txt** which was later cleaned for any blogs that did not have atom+xml feeds. Usually these were blogs that no longer existed.

Libraries used:

- import requests
- from bs4 import BeautifulSoup
- import os

```
python getFeed.py
```

```

import requests
from bs4 import BeautifulSoup
```

```
4  import os
5
6
7  def getFeed(filename):
8      # print("FILENAME:", filename)
9      with open("blogs/" + filename) as f:
10         f.seek(0)
11         html = f.read()
12         soup = BeautifulSoup(html, 'html.parser')
13         feed = soup.find_all('link', attrs={'type': 'application
               /atom+xml'})
14         if(feed):
15             return feed[0]['href']
16         return None
17
18
19 def deleteNoFeedFiles():
20     noneLines = []
21     f = open("feedList.txt", "r+")
22     d = f.readlines()
23     f.seek(0)
24     for i, line in enumerate(d):
25         if 'None' in line:
26             noneLines.append(i)
27         else:
28             f.write(line)
29     f.truncate()
30     f.close()
31
32     print("noneLine:",noneLines)
33     f = open("100BlogUrls.txt", "r+")
34     d = f.readlines()
35     f.seek(0)
36     for i, line in enumerate(d):
37         if i in noneLines:
38             # delete file
39             fileToDelete = line.split()[0]
40             print("Deleting:", fileToDelete)
41             os.remove("blogs/" + fileToDelete)
42         else:
43             f.write(line)
44     f.truncate()
45     f.close()
46
47
48 if __name__ == "__main__":
49
50     with open("100BlogUrls.txt") as f, open("feedList.txt", 'w')
           as out:
```

```
51          for line in f:
52              filename = line.split()[0]
53              feed = getFeed(filename)
54              print(feed, file=out)
55
56      deleteNoFeedFiles()
```

Listing 2: Python script to get 100 atom feed blogs

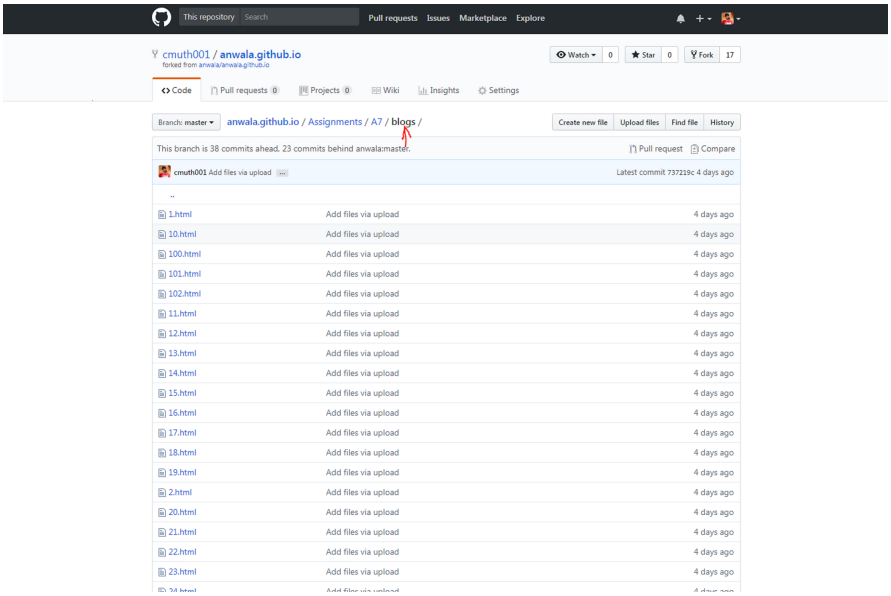

Figure 1: Blogs folder structure

Finally I proceeded to to write another python script called **generateFeedVector.py** shown in Listing 4 which utilized the code provide by the Programming Collective Intelligence (PCI) book [3]. This script was slightly modified to be usable for python but also adding a limit to the amount of words the blog-term matrix allowed to a maximum of 1000 . I also didn't check for stop words when I retrieved the atom feeds, but I did check if words were stop words before creating the the matrix by using the**from nltk.corpus import stopwords**. If a word was found to be a stop word according to their corpus it would not be added to the matrix. When this was completed it was saved to a file called **blogdata.txt**.

Libraries used:

- import feedparser

- import re
- from nltk.corpus import stopwords

```
python generateFeedVector.py
```

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
import feedparser
import re
from nltk.corpus import stopwords

stops = stopwords.words("english")
stopWordsCount = 0
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:
            if word not in stops:
                wc.setdefault(word, 0)
                wc[word] += 1
            else:
                stopWordsCount += 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)
```

```
43
44      # Convert to lowercase
45      return [word.lower() for word in words if word != '']
46
47
48  apcount = {}
49  wordcounts = {}
50  nofeed = 0
51  feedlist = [line for line in open('feedList.txt')]
52  for feedurl in feedlist:
53      try:
54          (title, wc) = getwordcounts(feedurl)
55          wordcounts[title] = wc
56          for (word, count) in wc.items():
57              apcount.setdefault(word, 0)
58              if count > 1:
59                  apcount[word] += 1
60      except:
61          nofeed+=1
62          print('Failed to parse feed', feedurl)
63
64  wordlist = []
65
66  for (w, bc) in apcount.items():
67      frac = float(bc) / len(feedlist)
68      if frac > 0.1 and frac < 0.5:
69          wordlist.append(w)
70      if len(wordlist) >= 1000:
71          break
72
73  out = open('blogdata.txt', 'w')
74  out.write('Blog')
75  for word in wordlist:
76      out.write('\t%s' % word)
77  out.write('\n')
78  for (blog, wc) in wordcounts.items():
79      print("Title: ",blog)
80      out.write(blog)
81      for word in wordlist:
82          if word in wc:
83              out.write('\t%d' % wc[word])
84          else:
85              out.write('\t0')
86      out.write('\n')
87  print("nofeed:",nofeed)
88  print("stopWordsCount:", stopWordsCount)
```

Listing 3: Python script to generate blog-term matrix

2

Question

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).

Answer

To solve the above question I used the code provided by the Programming Collective Intelligence book to write a script in python called **ASCIIAndJPEGDendrogram.py** as shown in Listing **??** [3]. This script has a method called *createDendrogram* which utilizes the clusters.py file provided by the PCI book load the blog-term matrix created in question 1 to create a Hierarchical Clustering tree image, the dendrogram, as shown in Figure 2. I also created an ASCII file named **ASCII.txt** to represent this tree structure in text which is available on my Github page [1].

Libraries used:

- import clusters
- import sys

```
python ASCIIAndJPEGDendrogram.py
```

```
1
2  import clusters
3  import sys
4
5
6  def createDendrogram():
7      blogs, colnames, data = clusters.readfile('blogdata.txt')
8      cluster = clusters.hcluster(data)
9      clusters.drawdendrogram(cluster, blogs, jpeg='Dendrogram.jpg
           ')
10     f = open("ASCII.txt", 'w')
11     sys.stdout = f
12     clusters.printclust(cluster, labels=blogs)
13     f.close()
14     sys.stderr.close()
15
16
```

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

Listing 4: Python script to create Dendrogram image and tree structure

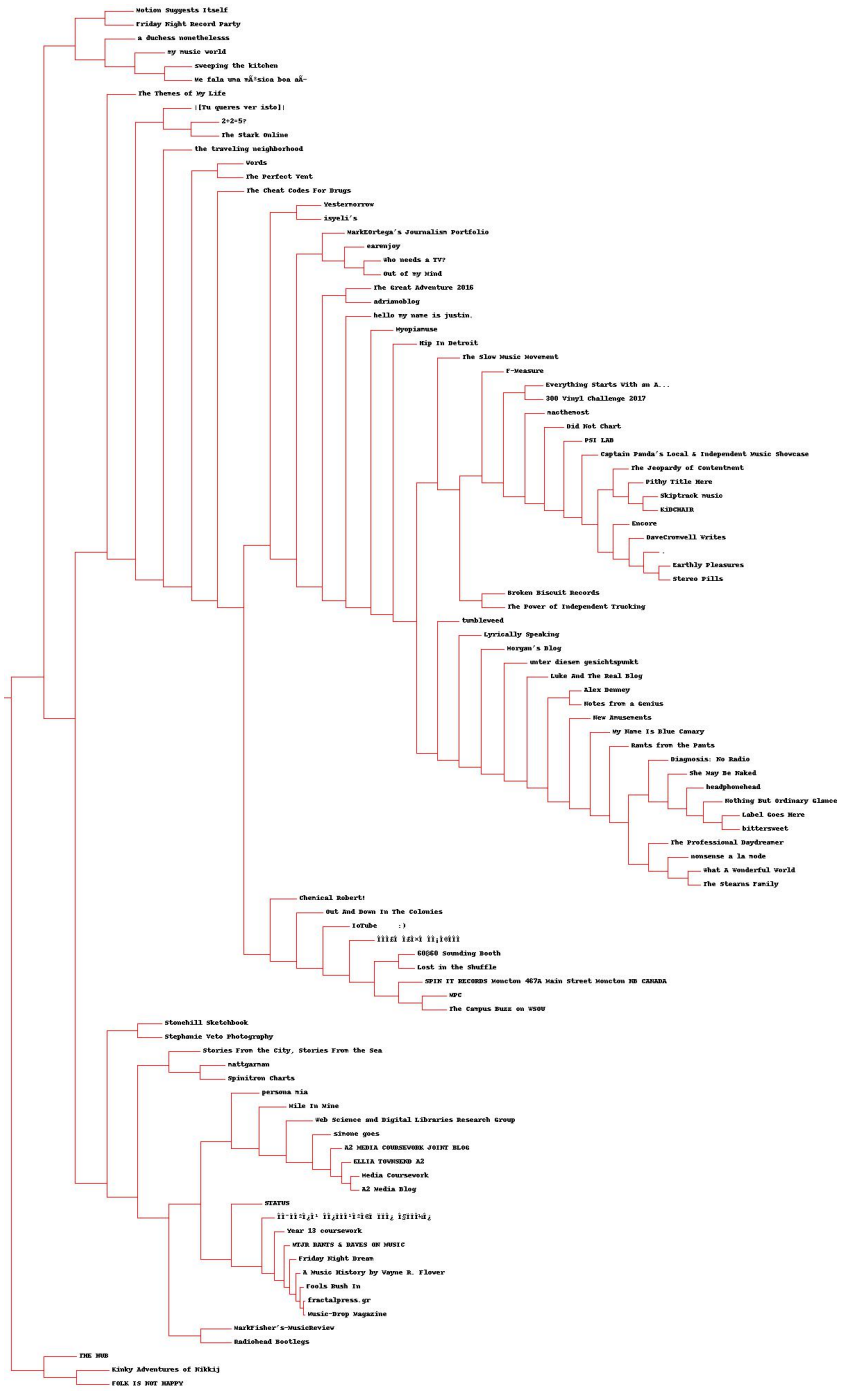

Figure 2: Dendrogram for blogs collected

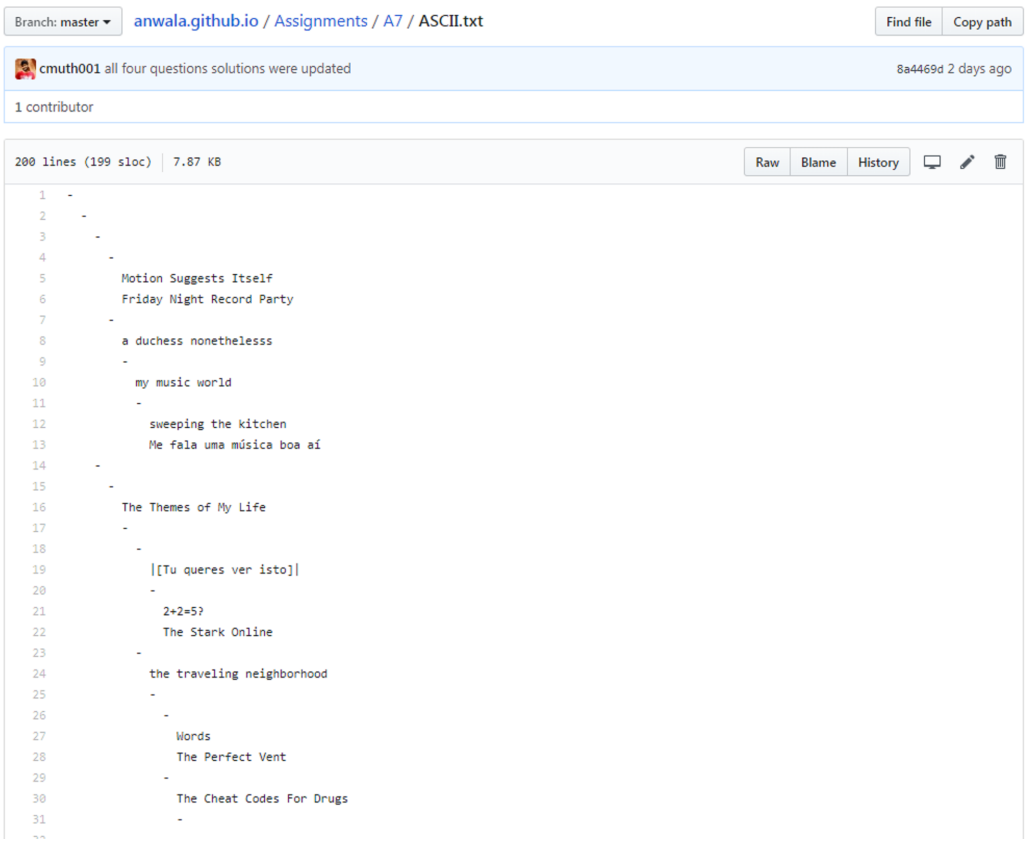

Figure 3: Ascii Tree Structure for collected blogs

3

Question

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

Answer

To solve the above question I used the code provided by the Programming Collective Intelligence book as shown in below to write a script in python called **kMean.py** [3]. The **kMean** method in **kMean.py** reads my blog-term matrix and using the method *kcluster* for the clusters.py library. I modified the *kcluster* method to also return the iteration count so it could be saved along with the blog title. For each value of k I created a separate file named **kclut_n.txt**and cluster group are listed in that file, where n is the value of k and those k values are stored in kMeanValues variable . For a k value of 5 it took 7 iterations with the values of each centroids shown in Listing 8. For a k value of 10 it took 4 iterations with the values of each centroids shown in Listing 7. For a k value of 20 it took 3 iterations with the values of each centroids shown in below

```
python kMean.py
```

```
1  import clusters
2  def kMean():
3      kMeanValues = [5, 10, 20]
4      blogs, colnames, data = clusters.readfile('blogdata.txt')
5      for i in kMeanValues:
6
7          kclust, itercount = clusters.kcluster(data, k=i)
8          print(kclust)
9          f = open("kclust_%d.txt" % i, 'w')
10         f.write("Total Number Of Iterations: %d \n" % itercount)
11         print(len(kclust))
12         clusterCount = 1
13         for cluster in kclust:
14             i=1
15             f.write("---\n")
16             f.write("Cluster %d \n" % clusterCount)
17             for blogid in cluster:
18                 f.write(str(i)+".\t"+blogs[blogid] + "\n")
19                 i+=1
20             f.write("\n")
```

```
                clusterCount+=1
if __name__ == "__main__":
    kMean()
```

Listing 5: K-means clustering with a value of 5

when k = 5, the below text file is generated

```
Total Number Of Iterations: 7
---
Cluster 1
1.      Skiptrack music
2.      mattgarman
3.      Diagnosis: No Radio
4.      persona mia
5.      Chemical Robert!
6.      MarkFisher's-MusicReview
7.      DaveCromwell Writes
8.      F-Measure
9.      Hip In Detroit
10.     Earthly Pleasures
11.     .
12.     2+2=5?
13.     Did Not Chart
14.     PSI LAB
15.     macthemost
16.     Broken Biscuit Records
17.     Stonehill Sketchbook
18.     Stereo Pills
19.     Stories From the City, Stories From the Sea
20.     Captain Panda's Local & Independent Music Showcase
21.     Stephanie Veto Photography
22.     KiDCHAIR
23.     The Power of Independent Trucking
24.     Encore
25.     my music world
26.     The Jeopardy of Contentment
27.     Everything Starts With an A...
28.     The Stark Online
29.     Myopiamuse
30.     300 Vinyl Challenge 2017
31.     The Slow Music Movement
32.     New Amusements

---
Cluster 2
1.      Media Coursework
2.      tumbleweed
3.      Label Goes Here
4.      The Professional Daydreamer
```

```
5.      nonsense a la mode
6.      Yestermorrow
7.      Pithy Title Here
8.      The Themes of My Life
9.      Kinky Adventures of Nikkij
10.     headphonehead
11.     a duchess nonethelesss
12.     Words
13.     What A Wonderful World
14.     My Name Is Blue Canary
15.     simone goes
16.     60@60 Sounding Booth
17.     Nothing But Ordinary Glances At Extraordinary Things
18.     Lost in the Shuffle
19.     She May Be Naked
20.     SPIN IT RECORDS Moncton 467A Main Street Moncton NB
    CANADA
21.     Rants from the Pants
22.     Lyrically Speaking
23.     THE HUB
24.     bittersweet
25.     MPC
26.     The Campus Buzz on WSOU
27.     ELLIA TOWNSEND A2
28.     A2 Media Blog
29.     The Cheat Codes For Drugs
30.     A2 MEDIA COURSEWORK JOINT BLOG
31.     hello my name is justin.
32.     Spinitron Charts
33.     The Perfect Vent
34.     Luke And The Real Blog
35.     isyeli's
36.     The Stearns Family
37.     adrianoblog

---
Cluster 3
1.      IoTube      :)
2.      Alex Denney
3.      |[Tu queres ver isto]|
4.      unter diesem gesichtspunkt
5.      Motion Suggests Itself
6.      Who needs a TV?
7.      Morgan's Blog
8.      earenjoy
9.      Mile In Mine
10.     Web Science and Digital Libraries Research Group
11.
12.     Out of my Mind
```

```
13.      The Great Adventure 2016
14.      MarkEOrtega's Journalism Portfolio
15.      Notes from a Genius
16.      Radiohead Bootlegs
17.      sweeping the kitchen
18.      Me fala uma m sica boa a
19.      the traveling neighborhood
20.      FOLK IS NOT HAPPY

---
Cluster 4
1.       Out And Down In The Colonies
2.       Friday Night Dream
3.       MTJR RANTS & RAVES ON MUSIC
4.       STATUS
5.       Year 13 coursework
6.       fractalpress.gr
7.       Fools Rush In
8.
9.       Music-Drop Magazine

---
Cluster 5
1.       A Music History by Wayne R. Flower
2.       Friday Night Record Party
```

Listing 6: K-means clustering with a value of 5

when k = 10, the below text file is generated

```
Total Number Of Iterations: 4
---
Cluster 1
1.       nonsense a la mode
2.       Kinky Adventures of Nikkij
3.       Stonehill Sketchbook
4.       What A Wonderful World
5.       60@60 Sounding Booth
6.       Lost in the Shuffle
7.       Stephanie Veto Photography
8.       The Stearns Family
9.       adrianoblog

---
Cluster 2
1.       Skiptrack music
2.       unter diesem gesichtspunkt
3.       Friday Night Record Party
4.       Yestermorrow
5.       PSI LAB
```

6. Stereo Pills
7. THE HUB
8. Everything Starts With an A...
9. 300 Vinyl Challenge 2017

Cluster 3
1. Media Coursework
2. simone goes
3. ELLIA TOWNSEND A2
4. A2 Media Blog
5. A2 MEDIA COURSEWORK JOINT BLOG

Cluster 4
1. Alex Denney
2. tumbleweed
3. mattgarman
4. Motion Suggests Itself
5. Label Goes Here
6. The Professional Daydreamer
7. Morgan's Blog
8. DaveCromwell Writes
9. F–Measure
10. Hip In Detroit
11. Earthly Pleasures
12. .
13. 2+2=5?
14. Did Not Chart
15. Pithy Title Here
16. The Themes of My Life
17. headphonehead
18. macthemost
19. a duchess nonethelessss
20. Broken Biscuit Records
21. Words
22. My Name Is Blue Canary
23. Nothing But Ordinary Glances At Extraordinary Things
24. She May Be Naked
25. Captain Panda's Local & Independent Music Showcase
26. SPIN IT RECORDS Moncton 467A Main Street Moncton NB CANADA
27. Rants from the Pants
28. KiDCHAIR
29. Lyrically Speaking
30. Encore
31. my music world
32. The Jeopardy of Contentment
33. bittersweet

```
34.     Notes from a Genius
35.     MPC
36.     The Campus Buzz on WSOU
37.     The Stark Online
38.     The Cheat Codes For Drugs
39.     Myopiamuse
40.     The Slow Music Movement
41.     hello my name is justin.
42.     Spinitron Charts
43.     The Perfect Vent
44.     Luke And The Real Blog
45.     isyeli's
46.     FOLK IS NOT HAPPY
47.     New Amusements

---
Cluster 5
1.      IoTube      :)
2.      Out And Down In The Colonies
3.      |[Tu queres ver isto]|
4.      Chemical Robert!
5.      Who needs a TV?
6.      earenjoy
7.
8.      Out of my Mind
9.      Stories From the City, Stories From the Sea
10.     The Great Adventure 2016
11.     MarkEOrtega's Journalism Portfolio
12.     Radiohead Bootlegs
13.     sweeping the kitchen
14.     Me fala uma m sica boa  a

---
Cluster 6
1.      persona mia
2.      MarkFisher's-MusicReview
3.      Mile In Mine
4.      Web Science and Digital Libraries Research Group
5.      the traveling neighborhood

---
Cluster 7
1.      Diagnosis: No Radio
2.      STATUS
3.      The Power of Independent Trucking

---
Cluster 8
1.      A Music History by Wayne R. Flower
```

```
118
119 ---
120 Cluster 9
121 1.      Friday Night Dream
122 2.      MTJR RANTS & RAVES ON MUSIC
123 3.      Year 13 coursework
124 4.      fractalpress.gr
125 5.      Fools Rush In
126 6.
127 7.      Music-Drop Magazine
128
129 ---
130 Cluster 10
```

Listing 7: K-means clustering with a value of 10

when k = 20, the below text file is generated

4

Question

4. Use MDS to create a JPEG of the blogs similar to slide 29 of the week 12 lecture. How many iterations were required?

Answer

To solve the above question I again used the code provided by the Programming Collective Intelligence book as shownb elow to write a script called *mds.py* [3]. The *mds* method reads my blog-term matrix and this time uses the method *scaledown* in the clusters.py library. I modified the *scaledown* method to also return the iteration count so it could be saved along with the blog title. The *mds* method then utilizes the data it takes from my blog-term matrix and utilizes multidimensional scaling (MDS) to create a visual representation of the distance matrix in two dimensions created from the *scaledown* method. The MDS visualization is shown in Figure 4. The iteration count was 345 and I took the scrren shot of it and saved as **mdsScreenShot.png** which is available on my Github page [1].

```
python mds.py
```

```
1  import clusters
2
3  def mds():
4      blognames, words, data = clusters.readfile('blogdata.txt')
5      coords, itercount = clusters.scaledown(data)
6      clusters.draw2d(coords, labels=blognames, jpeg='mds.jpg')
7      print ('Iteration count: %d' % itercount)
8
9  if __name__ == "__main__":
10     mds()
```

Listing 8: Python script for MDS visualization

Figure 4: Two dimensional representation of blogs

5

Question

5. Re-run question 2, but this time with proper TFIDF calculations instead of the hack discussed on slide 7 (p. 32). Use the same 1000 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 1000 terms and instead come up with TFIDF scores for all the terms and then choose the top 1000 from that list, but I'm trying to limit the amount of work necessary.

Answer

NOT ATTEMPTED

6

Question

6. Re-run questions 1-4, but this time instead of using the 98 "random" blogs, use 98 blogs that should be "similar" to:

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

Choose approximately equal numbers for both blog sets (it doesn't have to be a perfect 49-49 split, but it should be close). Explain in detail your strategy for locating these blogs.

Compare and contrast the results from the 98 "random" blogs and the 98 "targeted" blogs.

Answer

NOT ATTEMPTED

References

[1] https://github.com/cmuth001/anwala.github.io/tree/master/Assignments/A7.

[2] https://docs.python.org/2/library/sets.html.

[3] Richardson, Leonard. "Beautiful Soup Documentation." Beautiful Soup Documentation - Beautiful Soup 4.4.0 Documentation. N.p., n.d. Web. 24 Jan. 2017. https://www.crummy.com/software/BeautifulSoup/bs4/doc/.

[4] Segaran, Toby. "Programming Collective Intelligence". O' Reilly, 2007. Web. 6 April 2017. https://github.com/arthur-e/Programming-Collective-Intelligence.