Bryan Carey
CS532 Assignment 8

F

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.

To solve this problem, I created getBlog.py, which is listed below.

Bryan Carey
CS532 Assignment 8

```
import requests
import generatefeedvector.py

def get100blogurls():
    f = open('100blogurls.txt', 'w')
    url_set = set()
    while (len(url_set) < 100):
        url = "http://www.blogger.com/next-blog?navBar=true&blogID=3471633091411211117"
        generate_url = requests.get(url)
        final_url = generate_url.url.strip('?expref=next-blog/')
        url_set.add(final_url)
    for element in url_set:
        print element
        f.write(element + '\n')
    f.write('http://f-measure.blogspot.com' + '\n')
    f.write('http://ws-dl.blogspot.com')


def completeURL():
    input = open('100blogurls.txt', 'r')
    output = open('100completeurls.txt', 'w')
    for element in input:
        add = "/feeds/posts/default?alt=rss"
        item = element.strip() + add
        output.write(item + '\n')
        print item

#get100blogurls()
completeURL()
```

I experimented with a few data structures in which to store the URLs, however a set gave me the least amount of trouble. I used the PowerPoints help in creating the file, as it gave several of the needed pieces, such as the blogger.com site I stored as my url variable, and a few others. This was also very similar to our earlier assignment in which we worked with URLs so I referenced that one as well in regards to handling them. I ran the two functions in succession, which took a few times because the first function would sometimes crash before it ran to completion. Once I was able to gather all 100 blog URLs, and completed them with the necessary suffix, '/feeds/post/default?alt=rss', I ran the newly generated file, 100completeurls.txt through generatefeedvector.py. I found this online at https://github.com/arthur-e/Programming-Collective-Intelligence/blob/master/chapter3/generatefeedvector.py while searching for the PCI text. The code for generatevectorfee.py is found below.

Bryan Carey
CS532 Assignment 8

```python
#!/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 != '']

```

```
42  apcount = {}
43  wordcounts = {}
44  feedlist = [line for line in file('feedlist.txt')]
45  for feedurl in feedlist:
46      try:
47          (title, wc) = getwordcounts(feedurl)
48          wordcounts[title] = wc
49          for (word, count) in wc.items():
50              apcount.setdefault(word, 0)
51              if count > 1:
52                  apcount[word] += 1
53      except:
54          print 'Failed to parse feed %s' % feedurl
55
56  wordlist = []
57  for (w, bc) in apcount.items():
58      frac = float(bc) / len(feedlist)
59      if frac > 0.1 and frac < 0.5:
60          wordlist.append(w)
61
62  out = file('blogdata1.txt', 'w')
63  out.write('Blog')
64  for word in wordlist:
65      out.write('\t%s' % word)
66  out.write('\n')
67  for (blog, wc) in wordcounts.items():
68      print blog
69      out.write(blog)
70      for word in wordlist:
71          if word in wc:
72              out.write('\t%d' % wc[word])
73          else:
74              out.write('\t0')
75      out.write('\n')
```

The output of this generatefeedvector.py was spat out into a .txt file entitled blogdata1.txt. A sample of the output is as follows, as the entire file is too large to display in one shot.

Bryan Carey
CS532 Assignment 8

Blog	woods	kids	golden	catchy	absolute	wrong	fit	songwriter	service	needed	master	feeling	singer
SPIN IT RECORDS Moncton 467A Main Street Moncton NB CANADA						3	4	8	0	0	0	0	0
Riley Haas' blog	0	0	0	0	0	0	0	0	0	0	0	0	0
Coyote Doc Music Co-op	0	0	2	6	1	5	1	2	0	1	3	6	5
U-Rock Radio™	0	1	0	0	1	0	1	1	3	0	0	1	1	0
(Insert World Problem Here) Sucks.			0	0	0	0	0	0	0	0	0	0	1
Friday Night Dream	0	0	0	0	0	1	0	0	0	0	0	0	0
On Warmer Music 2	7	6	8	1	3	2	3	2	4	0	15	10	3
New Music Matters	3	1	1	3	1	0	0	12	0	3	0	1	23
She May Be Naked	2	3	0	0	0	2	2	0	0	8	1	9	0
Pithy Title Here	7	9	0	3	1	11	8	1	12	7	0	12	9
Spinitron Charts	0	0	0	0	0	0	0	0	1	0	0	0	0
THE HUB 0	2	1	0	0	0	0	0	1	0	0	0	0	0	0
Anthems and Atleticos	0	7	0	0	2	0	1	0	5	2	3	7	1
Primitive Offerings	0	1	0	0	0	0	1	0	0	0	0	0	0
Web Science and Digital Libraries Research Group						0	2	0	0	1	5	2	0	33
Words	0	0	0	0	0	2	0	0	0	2	0	1	0	2	0
Steel City Rust 0	2	0	0	0	0	2	0	1	0	0	4	3	1
Fran Brighton	0	0	0	0	0	0	0	0	0	0	0	1	0	0
Stereo Pills	0	0	0	7	0	0	0	2	0	1	0	0	2	0
MarkEOrtega's Journalism Portfolio			0	0	0	1	0	1	0	1	0	0	0
Oh Yes Jónsi!!	0	0	0	0	0	0	0	0	0	0	0	0	0	0
aubade	4	2	0	1	0	3	3	0	2	1	0	2	1	1	0
turnitup!	0	1	0	0	0	3	0	0	2	1	1	1	1	1
Stories From the City, Stories From the Sea						5	1	2	0	0	0	0	0	0
Chemical Robert!	0	1	0	0	0	0	0	0	1	0	0	0	1
adrianoblog	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Stephanie Veto Photography			0	0	0	0	0	0	0	0	0	0	0	0
A H T A P O T	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Diagnosis: No Radio	0	9	0	0	0	3	5	1	0	2	2	11	1
Floorshime Zipper Boots 0	0	0	1	3	0	0	1	0	0	0	0	1
Did Not Chart	0	0	0	0	0	0	0	0	3	0	0	2	0	0
Cast Iron Songs 0	0	0	0	0	0	0	0	0	0	2	0	0	0	0
|[Tu queres ver isto]|	0	0	0	0	0	0	0	0	0	0	0	0	0	0
The Stearns Family	0	12	0	0	0	4	0	0	0	2	0	1	0
IoTube :)	0	0	0	0	0	0	0	0	0	0	0	0	0	0
DaveCromwell Writes	0	3	5	4	2	5	8	17	1	5	4	16	27
jaaackie.	0	0	0	0	0	2	0	0	0	2	0	5	0	0
A2 MEDIA COURSEWORK JOINT BLOG	0	0	0	0	0	0	0	1	0	1	2	0	0
nonsense a la mode	0	9	1	0	0	1	0	0	4	2	1	0	0
Happy Accidents 0	0	1	0	1	1	0	0	2	0	1	2	0	0

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).

My colleague Michelle talked me through how to construct the dendogram. I created dendogramCreator.py. that calls from clusters.py, https://github.com/arthur-e/Programming-Collective-Intelligence/blob/master/chapter3/clusters.py

Bryan Carey
CS532 Assignment 8

```
1   import clusters
2   import sys
3
4   blognames, words, data = clusters.readfile('blogdata1.txt')
5
6   clust = clusters.hcluster(data)
7
8   clusters.printclust(clust, labels=blognames)
9   sys.stdout = open('ascii.text', 'w+')
10
11  clusters.drawdendrogram(clust, blognames, jpeg='dendogram.jpg')
```

The functions that were called are also listed below.

```
62  def hcluster(rows, distance=pearson):
63      distances = {}
64      currentclustid = -1
65
66    # Clusters are initially just the rows
67      clust = [bicluster(rows[i], id=i) for i in range(len(rows))]
68
69      while len(clust) > 1:
70          lowestpair = (0, 1)
71          closest = distance(clust[0].vec, clust[1].vec)
72
73      # loop through every pair looking for the smallest distance
74          for i in range(len(clust)):
75              for j in range(i + 1, len(clust)):
76          # distances is the cache of distance calculations
77                  if (clust[i].id, clust[j].id) not in distances:
78                      distances[(clust[i].id, clust[j].id)] = \
79                          distance(clust[i].vec, clust[j].vec)
80
81                  d = distances[(clust[i].id, clust[j].id)]
82
83                  if d < closest:
84                      closest = d
85                      lowestpair = (i, j)
86
87      # calculate the average of the two clusters
88          mergevec = [(clust[lowestpair[0]].vec[i] + clust[lowestpair[1]].vec[i])
89                      / 2.0 for i in range(len(clust[0].vec))]
90
91      # create the new cluster
92          newcluster = bicluster(mergevec, left=clust[lowestpair[0]],
93                                 right=clust[lowestpair[1]], distance=closest,
94                                 id=currentclustid)
95
96      # cluster ids that weren't in the original set are negative
97          currentclustid -= 1
98          del clust[lowestpair[1]]
99          del clust[lowestpair[0]]
100         clust.append(newcluster)
101
102     return clust[0]
```

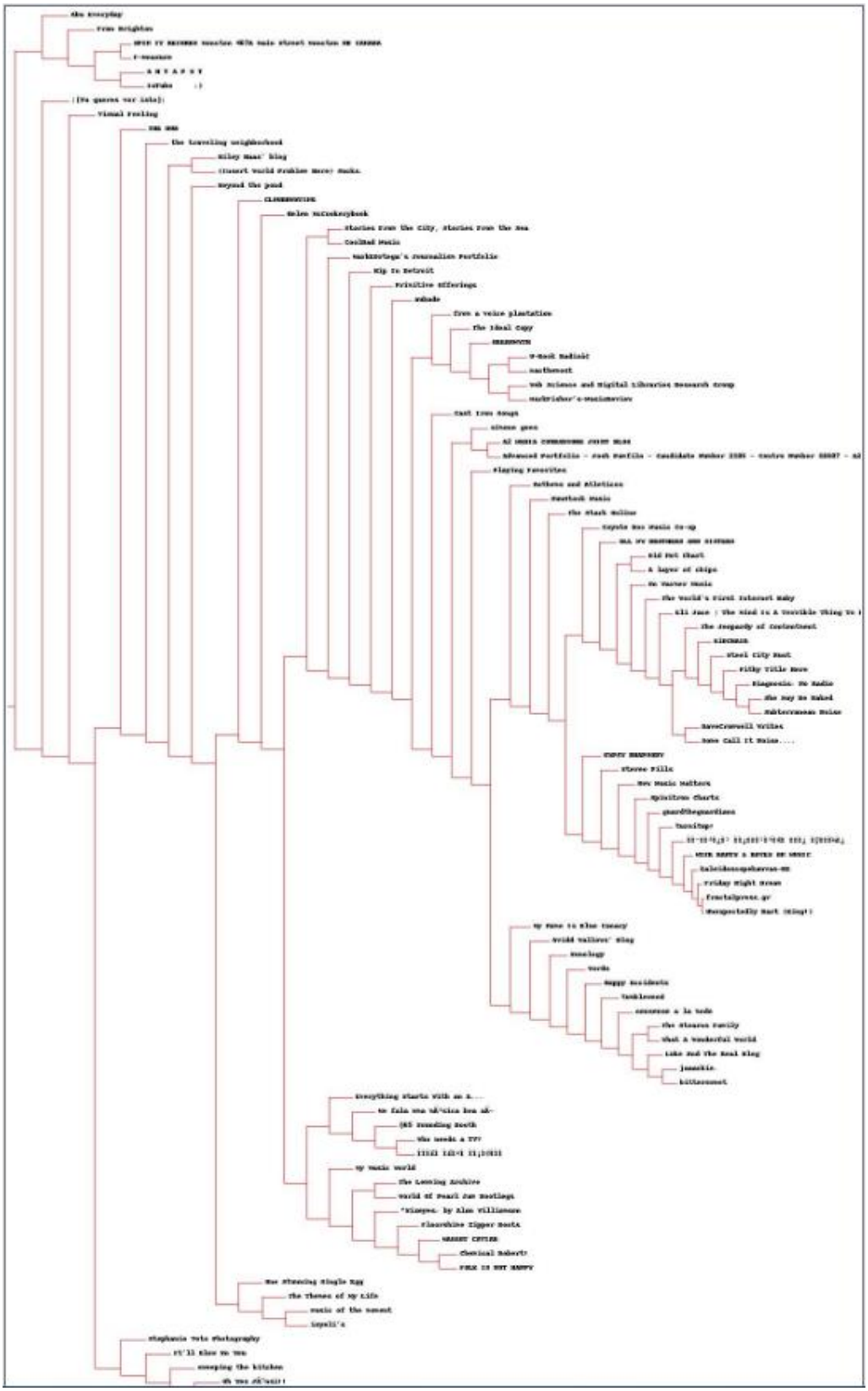
Bryan Carey
CS532 Assignment 8

```
105  def printclust(clust, labels=None, n=0):
106    # indent to make a hierarchy layout
107      for i in range(n):
108          print ' ',
109      if clust.id < 0:
110      # negative id means that this is branch
111          print '-'
112      else:
113      # positive id means that this is an endpoint
114          if labels == None:
115              print clust.id
116          else:
117              print labels[clust.id]
118
119    # now print the right and left branches
120      if clust.left != None:
121          printclust(clust.left, labels=labels, n=n + 1)
122      if clust.right != None:
123          printclust(clust.right, labels=labels, n=n + 1)
124

146  def drawdendrogram(clust, labels, jpeg='clusters.jpg'):
147    # height and width
148      h = getheight(clust) * 20
149      w = 1200
150      depth = getdepth(clust)
151
152    # width is fixed, so scale distances accordingly
153      scaling = float(w - 150) / depth
154
155    # Create a new image with a white background
156      img = Image.new('RGB', (w, h), (255, 255, 255))
157      draw = ImageDraw.Draw(img)
158
159      draw.line((0, h / 2, 10, h / 2), fill=(255, 0, 0))
160
161    # Draw the first node
162      drawnode(
163          draw,
164          clust,
165          10,
166          h / 2,
167          scaling,
168          labels,
169          )
170      img.save(jpeg, 'JPEG')
171
```

Bryan Carey
CS532 Assignment 8

The output of my file is as follows, and the ascii.txt file is available on GitHub as requested.



Bryan Carey
CS532 Assignment 8

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?

To solve this problem I used another piece of functionality from clusters, to construct my k_from_clusters.py. My colleague also aided in the creation of this, I was instructed to create three identical calls to the kcluster function with the three specified increments of k.

```
1  import clusters
2
3
4
5  print('k = 5')
6  k = clusters.kcluster(data, k=5)
7  print str(k) + '\n'
8
9  print('k = 10')
10 k = clusters.kcluster(data, k=10)
11 print str(k) + '\n'
12
13 print('k =20')
14 k = clusters.kcluster(data, k=20)
15 print str(k) + '\n'
16
```

Kcluster's declaration is as follows from clusters.py.

Bryan Carey
CS532 Assignment 8

```
228  def kcluster(rows, distance=pearson, k=4):
229    # Determine the minimum and maximum values for each point
230      ranges = [(min([row[i] for row in rows]), max([row[i] for row in rows]))
231                for i in range(len(rows[0]))]
232
233    # Create k randomly placed centroids
234      clusters = [[random.random() * (ranges[i][1] - ranges[i][0]) + ranges[i][0]
235                  for i in range(len(rows[0]))] for j in range(k)]
236
237      lastmatches = None
238      for t in range(100):
239          print 'Iteration %d' % t
240          bestmatches = [[] for i in range(k)]
241
242      # Find which centroid is the closest for each row
243          for j in range(len(rows)):
244              row = rows[j]
245              bestmatch = 0
246              for i in range(k):
247                  d = distance(clusters[i], row)
248                  if d < distance(clusters[bestmatch], row):
249                      bestmatch = i
250              bestmatches[bestmatch].append(j)
251
252      # If the results are the same as last time, this is complete
253          if bestmatches == lastmatches:
254              break
255          lastmatches = bestmatches
256
257      # Move the centroids to the average of their members
258          for i in range(k):
259              avgs = [0.0] * len(rows[0])
260              if len(bestmatches[i]) > 0:
261                  for rowid in bestmatches[i]:
262                      for m in range(len(rows[rowid])):
263                          avgs[m] += rows[rowid][m]
264                  for j in range(len(avgs)):
265                      avgs[j] /= len(bestmatches[i])
266                  clusters[i] = avgs
267
268      return bestmatches
```

Bryan Carey
CS532 Assignment 8

The output is as follows.

```
C:\Python27\python.exe Z:/CS432/Assignment8/Q3/k_from_clusters.py
k = 5
Iteration 0
Iteration 1
Iteration 2
Iteration 3
Iteration 4
Iteration 5
Iteration 6
[[1, 2, 3, 11, 12, 14, 18, 19, 20, 23, 24, 29, 30, 31, 35, 37, 42, 44, 45,

k = 10
Iteration 0
Iteration 1
Iteration 2
Iteration 3
Iteration 4
Iteration 5
Iteration 6
[[5, 10, 22, 32, 43, 46, 59, 63, 88, 97], [7, 18, 30, 35, 44, 49, 52, 56, 5

k =20
Iteration 0
Iteration 1
Iteration 2
Iteration 3
Iteration 4
Iteration 5
Iteration 6
Iteration 7
[[], [], [26], [87], [2, 12, 19, 28, 30, 51, 53, 61, 64, 72, 73, 74, 77, 90


Process finished with exit code 0
```

6, 6 & 7 iterations were required respectively for 5, 10, & 20 as values of k.

Bryan Carey
CS532 Assignment 8

4. Use MDS to create a JPEG of the blogs similar to slide 29 of the

week 12 lecture. How many iterations were required?

My MDS.py file was very simple and called from two functions form the clusters.py file- scaledown & draw2d. My file is as follows,

```
1  import clusters
2
3  blognames, words, data = clusters.readfile('blogdata1.txt')
4
5  c = clusters.scaledown(data)
6
7  clusters.draw2d(c, blognames, jpeg='mds.jpg')
```

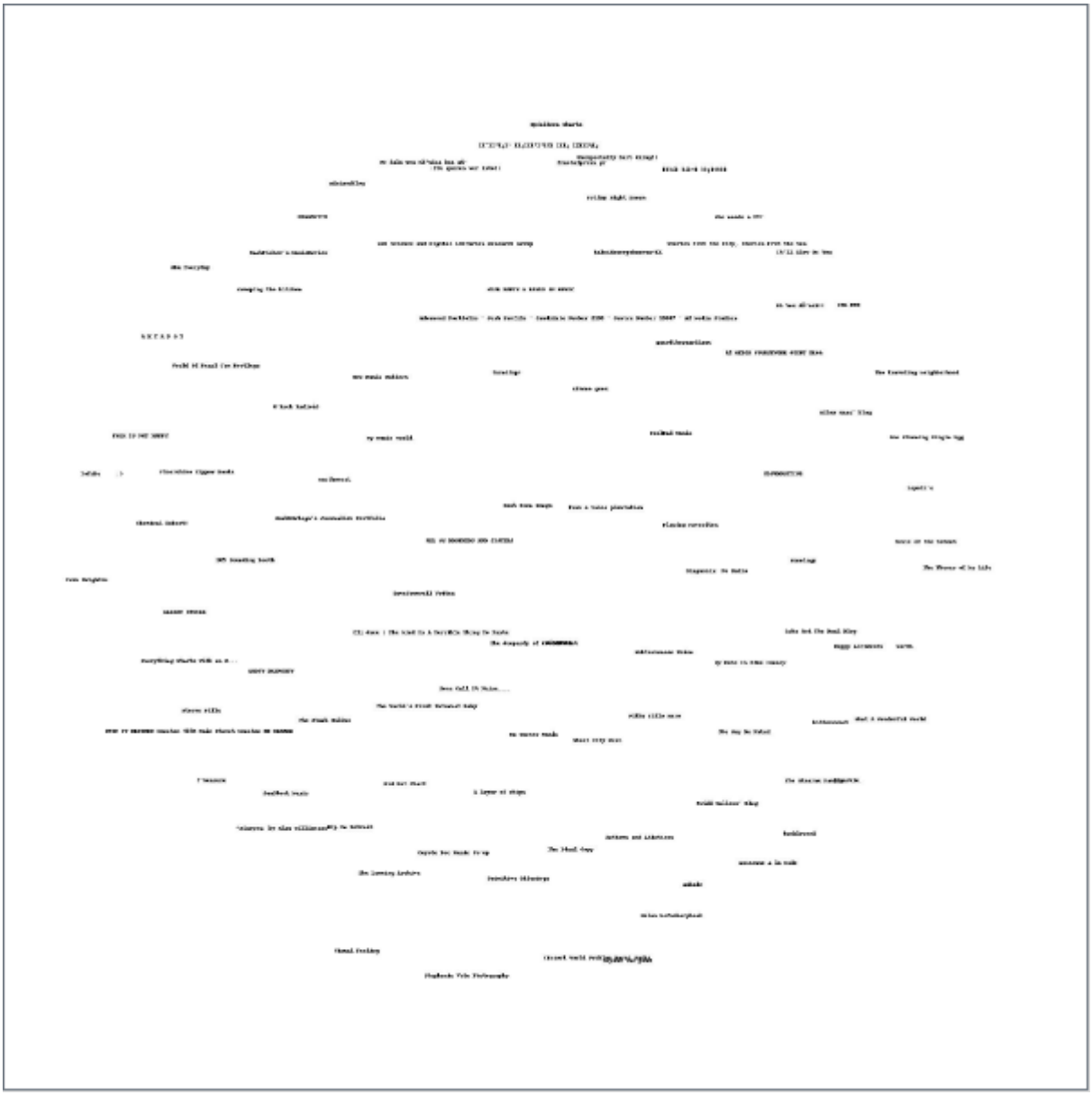
And the borrowed functionality are

```
285  def scaledown(data, distance=pearson, rate=0.01):
286      n = len(data)
287
288    # The real distances between every pair of items
289      realdist = [[distance(data[i], data[j]) for j in range(n)] for i in
290                  range(0, n)]
```

```
339  def draw2d(data, labels, jpeg='mds2d.jpg'):
340      img = Image.new('RGB', (2000, 2000), (255, 255, 255))
341      draw = ImageDraw.Draw(img)
342      for i in range(len(data)):
343          x = (data[i][0] + 0.5) * 1000
344          y = (data[i][1] + 0.5) * 1000
345          draw.text((x, y), labels[i], (0, 0, 0))
346      img.save(jpeg, 'JPEG')
347
```

There was more to the scaledown function, however it would not fit in one screenshot.

Bryan Carey
CS532 Assignment 8



The MDS.jpeg I generated. It took a total of 29 interations.

Bryan Carey
CS532 Assignment 8

```
C:\Python27\python.exe Z:/CS432/Assignment8/Q4/MDS.py
4233.62320823
3381.37477765
3318.06532197
3283.64398847
3254.14246173
3228.77840884
3207.27087737
3188.93674699
3174.08189037
3161.24926018
3148.95380844
3137.88980421
3127.71465533
3119.15387249
3112.55859396
3106.80427737
3100.98067888
3095.92650838
3091.64319906
3088.48428465
3085.87100794
3083.34331957
3081.21708818
3079.74051901
3078.73751234
3078.03575557
3077.77262293
3077.6126892
3077.63891296

Process finished with exit code 0
```

==========

========The questions below is for 3 points extra credit==========

==========

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.

Bryan Carey
CS532 Assignment 8

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.

==========

========The questions below is for 5 points extra credit==========

==========

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.