Web Science cs532-s16

Assignment 8 Report

Dr. Michael L. Nelson

By: Huan Huang
04/07/2016

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 entrytitle (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.

Answer

To solve this problem, I first wrote a shell script to click on the next-blog button for 397 times, then, the 397 final URIs I got are saved together with the 2 required links in a file.

```
391 http://semregrasluispink.blogspot.com/?expref=next-blog
392 http://partyfullofstrangers.blogspot.com/?expref=next-blog
393 http://www.sonology.com/?expref=next-blog
394 http://dinosaursarefun.blogspot.com/?expref=next-blog
395 http://blog.spinitron.com/?expref=next-blog
396 http://trembleunderboomlights.blogspot.com/?expref=next-blog
397 http://stonehillsketchbook.blogspot.com/?expref=next-blog
398 http://rosiegigga2media.blogspot.com/?expref=next-blog
399 http://floorshimezipperboots.blogspot.com/?expref=next-blog
400
```

Figure 1: Sample of part of blogList.txt

```
1 #!/bin/bash
2
3 echo "http://f-measure.blogspot.com/" > blogList.txt
4 echo "http://ws-dl.blogspot.com/" >> blogList.txt
5
6 for (( i = 3; i < 400; i++ )); do
7   #printf "blogPages$i.txt " >> blogList.txt
```

```
  curl -L -o uselesspage.txt -w %{url_effective} 'https://www.blogger.com/next-blog?navBar=true&blogID=3471633091411211117' >> blogList.txt
  printf "\n" >> blogList.txt
done
```

The reason I apply the next-blog link 397 times is because it often give me the same blog pages. Therefore, I use the command "sort -u filename" to remove the duplicate URIs and saved the result in the file unique.txt.

```
http://turnitupjack.blogspot.com/?expref=next-blog
http://ws-dl.blogspot.com/
http://www.chrisanne-grise.com/?expref=next-blog
http://www.radioshower.com/?expref=next-blog
http://www.samtasticreview.com/?expref=next-blog
http://www.sonology.com/?expref=next-blog
http://www.symmetrysymmetry.com/?expref=next-blog
http://www.thejeopardyofcontentment.com/?expref=next-blog

```

Figure 2: Sample of part of unique.txt

Once I have the unique links, I use another shell script to get the atom feed URIs of each of the unique links in the file unique.txt. Then, remove the extra links to make sure there are exactly 100 of them.

```
http://flowradio.blogspot.com/feeds/posts/default
http://f-measure.blogspot.com/feeds/posts/default
http://fortheotherthings.blogspot.com/feeds/posts/default
http://glipress.blogspot.com/feeds/posts/default
```

Figure 3: Sample of part of atom.txt

```
http://ws-dl.blogspot.com/feeds/posts/default
http://www.chrisanne-grise.com/feeds/posts/default
http://www.radioshower.com/feeds/posts/default
http://www.samtasticreview.com/feeds/posts/default
http://www.symmetrysymmetry.com/feeds/posts/default
http://www.thejeopardyofcontentment.com/feeds/posts/default

```

Figure 4: Another sample of part of atom.txt

```
#!/bin/bash

while read line;
do
    test=`curl $line | grep 'rel="alternate"' | grep atom | sed 's/.*href=//' | sed 's/\/>//' | sed 's/"//g' | sed 's/ //g'`
    echo $test >> atom.txt
done < unique.txt
```

Now, I use the generatefeedvector.py file I collected from the PCI(Programming Collective Intelligence) book to generate the word count file called blogdata.txt. I modified the generatefeedvector.py file slightly to count only 500 popular terms.

94	The Moon Topples	0	0	0	0	0	0
95	Encore	0	2	4	0	0	0
96	isyeli's	0	0	0	0	0	0
97	But She's Not Stupid	5	1	0	0	2	3
98	Haris Sfakianakis Photography	0	0	0	0	0	0
99	In the Frame Film Reviews	5	2	2	0	18	36
100	KISTE F.M.	0	2	0	0	0	0
101	I Hate The 90s	1	0	0	0	0	2
102							

Figure 5: Sample of part of blogdata.txt

```
import feedparser
import re

# Returns title and dictionary of word counts for an RSS feed
def getwordcounts(url):
  # 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={}
feedlist=[line for line in file('atom.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:
    print 'Failed to parse feed %s' % 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)>= 500:
      break

out=file('blogdata.txt','w')
out.write('Blog')
for word in wordlist: out.write('\t%s' % word)
out.write('\n')
for blog,wc in wordcounts.items():
  try:
    print blog
    out.write(blog)
  except:
    out.write(str(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')
```

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).

Answer

For this problem, I imported another file called cluster.py from PCI codes. Then, call the functions readfile() to process the data in blogdata.txt to get necessary information to feed to another function called hcluster(). The function hcluster() uses Person's method to decide which blogs are grouped together, then the clusters are plotted by another function called printclust() based on the result from hcluster(). Lastly, the image is drawn by the drawdendrogram() function.

```
from PIL import Image,ImageDraw

def readfile(filename):
  lines=[line for line in file(filename)]

  # First line is the column titles
  colnames=lines[0].strip().split('\t')[1:]
  rownames=[]
  data=[]
  for line in lines[1:]:
    p=line.strip().split('\t')
    # First column in each row is the rowname
    rownames.append(p[0])
    # The data for this row is the remainder of the row
    data.append([float(x) for x in p[1:]])
  return rownames,colnames,data


from math import sqrt

def pearson(v1,v2):
  # Simple sums
  sum1=sum(v1)
  sum2=sum(v2)

  # Sums of the squares
  sum1Sq=sum([pow(v,2) for v in v1])
  sum2Sq=sum([pow(v,2) for v in v2])

  # Sum of the products
  pSum=sum([v1[i]*v2[i] for i in range(len(v1))])

  # Calculate r (Pearson score)
  num=pSum-(sum1*sum2/len(v1))
  den=sqrt((sum1Sq-pow(sum1,2)/len(v1))*(sum2Sq-pow(sum2,2)/len(v1)))
  if den==0: return 0

  return 1.0-num/den

class bicluster:
  def __init__(self,vec,left=None,right=None,distance=0.0,id=None):
    self.left=left
    self.right=right
    self.vec=vec
    self.id=id
    self.distance=distance

def hcluster(rows,distance=pearson):
  distances={}
  currentclustid=-1

  # Clusters are initially just the rows
  clust=[bicluster(rows[i],id=i) for i in range(len(rows))]

```

```
  while len(clust)>1:
    lowestpair=(0,1)
    closest=distance(clust[0].vec,clust[1].vec)

    # loop through every pair looking for the smallest distance
    for i in range(len(clust)):
      for j in range(i+1,len(clust)):
        # distances is the cache of distance calculations
        if (clust[i].id,clust[j].id) not in distances:
          distances[(clust[i].id,clust[j].id)]=distance(clust[i].vec,clust[j].
    vec)

        d=distances[(clust[i].id,clust[j].id)]

        if d<closest:
          closest=d
          lowestpair=(i,j)

    # calculate the average of the two clusters
    mergevec=[
    (clust[lowestpair[0]].vec[i]+clust[lowestpair[1]].vec[i])/2.0
    for i in range(len(clust[0].vec))]

    # create the new cluster
    newcluster=bicluster(mergevec,left=clust[lowestpair[0]],
                         right=clust[lowestpair[1]],
                         distance=closest,id=currentclustid)

    # cluster ids that weren't in the original set are negative
    currentclustid-=1
    del clust[lowestpair[1]]
    del clust[lowestpair[0]]
    clust.append(newcluster)

  return clust[0]

def printclust(clust,labels=None,n=0):
  # indent to make a hierarchy layout
  for i in range(n): print ' ',
  if clust.id<0:
    # negative id means that this is branch
    print '-'
  else:
    # positive id means that this is an endpoint
    if labels==None: print clust.id
    else: print labels[clust.id]

  # now print the right and left branches
  if clust.left!=None: printclust(clust.left,labels=labels,n=n+1)
  if clust.right!=None: printclust(clust.right,labels=labels,n=n+1)

def getheight(clust):
  # Is this an endpoint? Then the height is just 1
  if clust.left==None and clust.right==None: return 1
```

```
108
109   # Otherwise the height is the same of the heights of
110   # each branch
111   return getheight(clust.left)+getheight(clust.right)
112
113 def getdepth(clust):
114   # The distance of an endpoint is 0.0
115   if clust.left==None and clust.right==None: return 0
116
117   # The distance of a branch is the greater of its two sides
118   # plus its own distance
119   return max(getdepth(clust.left),getdepth(clust.right))+clust.distance
120
121
122 def drawdendrogram(clust,labels,jpeg='clusters.jpg'):
123   # height and width
124   h=getheight(clust)*20
125   w=1200
126   depth=getdepth(clust)
127
128   # width is fixed, so scale distances accordingly
129   scaling=float(w-150)/depth
130
131   # Create a new image with a white background
132   img=Image.new('RGB',(w,h),(255,255,255))
133   draw=ImageDraw.Draw(img)
134
135   draw.line((0,h/2,10,h/2),fill=(255,0,0))
136
137   # Draw the first node
138   drawnode(draw,clust,10,(h/2),scaling,labels)
139   img.save(jpeg,'JPEG')
140
141 def drawnode(draw,clust,x,y,scaling,labels):
142   if clust.id<0:
143     h1=getheight(clust.left)*20
144     h2=getheight(clust.right)*20
145     top=y-(h1+h2)/2
146     bottom=y+(h1+h2)/2
147     # Line length
148     ll=clust.distance*scaling
149     # Vertical line from this cluster to children
150     draw.line((x,top+h1/2,x,bottom-h2/2),fill=(255,0,0))
151
152     # Horizontal line to left item
153     draw.line((x,top+h1/2,x+ll,top+h1/2),fill=(255,0,0))
154
155     # Horizontal line to right item
156     draw.line((x,bottom-h2/2,x+ll,bottom-h2/2),fill=(255,0,0))
157
158     # Call the function to draw the left and right nodes
159     drawnode(draw,clust.left,x+ll,top+h1/2,scaling,labels)
160     drawnode(draw,clust.right,x+ll,bottom-h2/2,scaling,labels)
161   else:
```

```
    # If this is an endpoint, draw the item label
    draw.text((x+5,y-7),labels[clust.id],(0,0,0))

def rotatematrix(data):
  newdata=[]
  for i in range(len(data[0])):
    newrow=[data[j][i] for j in range(len(data))]
    newdata.append(newrow)
  return newdata

import random

def kcluster(rows,distance=pearson,k=4):
  # Determine the minimum and maximum values for each point
  ranges=[(min([row[i] for row in rows]),max([row[i] for row in rows]))
  for i in range(len(rows[0]))]

  # Create k randomly placed centroids
  clusters=[[random.random()*(ranges[i][1]-ranges[i][0])+ranges[i][0]
  for i in range(len(rows[0]))] for j in range(k)]

  lastmatches=None
  for t in range(100):
    print 'Iteration %d' % t
    bestmatches=[[] for i in range(k)]

    # Find which centroid is the closest for each row
    for j in range(len(rows)):
      row=rows[j]
      bestmatch=0
      for i in range(k):
        d=distance(clusters[i],row)
        if d<distance(clusters[bestmatch],row): bestmatch=i
      bestmatches[bestmatch].append(j)

    # If the results are the same as last time, this is complete
    if bestmatches==lastmatches: break
    lastmatches=bestmatches

    # Move the centroids to the average of their members
    for i in range(k):
      avgs=[0.0]*len(rows[0])
      if len(bestmatches[i])>0:
        for rowid in bestmatches[i]:
          for m in range(len(rows[rowid])):
            avgs[m]+=rows[rowid][m]
        for j in range(len(avgs)):
          avgs[j]/=len(bestmatches[i])
        clusters[i]=avgs

  return bestmatches

def tanamoto(v1,v2):
  c1,c2,shr=0,0,0
```

```
216
217   for i in range(len(v1)):
218     if v1[i]!=0: c1+=1 # in v1
219     if v2[i]!=0: c2+=1 # in v2
220     if v1[i]!=0 and v2[i]!=0: shr+=1 # in both
221
222   return 1.0-(float(shr)/(c1+c2-shr))
223
224 def scaledown(data,distance=pearson,rate=0.01):
225   n=len(data)
226
227   # The real distances between every pair of items
228   realdist=[[distance(data[i],data[j]) for j in range(n)]
229             for i in range(0,n)]
230
231   # Randomly initialize the starting points of the locations in 2D
232   loc=[[random.random(),random.random()] for i in range(n)]
233   fakedist=[[0.0 for j in range(n)] for i in range(n)]
234
235   lasterror=None
236   for m in range(0,1000):
237     # Find projected distances
238     for i in range(n):
239       for j in range(n):
240         fakedist[i][j]=sqrt(sum([pow(loc[i][x]-loc[j][x],2)
241                                  for x in range(len(loc[i]))]))
242
243     # Move points
244     grad=[[0.0,0.0] for i in range(n)]
245
246     totalerror=0
247     for k in range(n):
248       for j in range(n):
249         if j==k: continue
250         # The error is percent difference between the distances
251         errorterm=(fakedist[j][k]-realdist[j][k])/realdist[j][k]
252
253         # Each point needs to be moved away from or towards the other
254         # point in proportion to how much error it has
255         grad[k][0]+=((loc[k][0]-loc[j][0])/fakedist[j][k])*errorterm
256         grad[k][1]+=((loc[k][1]-loc[j][1])/fakedist[j][k])*errorterm
257
258         # Keep track of the total error
259         totalerror+=abs(errorterm)
260     print totalerror
261
262     # If the answer got worse by moving the points, we are done
263     if lasterror and lasterror<totalerror: break
264     lasterror=totalerror
265
266     # Move each of the points by the learning rate times the gradient
267     for k in range(n):
268       loc[k][0]-=rate*grad[k][0]
269       loc[k][1]-=rate*grad[k][1]
```

```
270
271   return loc
272
273 def draw2d(data,labels,jpeg='mds2d.jpg'):
274   img=Image.new('RGB',(2000,2000),(255,255,255))
275   draw=ImageDraw.Draw(img)
276   for i in range(len(data)):
277     x=(data[i][0]+0.5)*1000
278     y=(data[i][1]+0.5)*1000
279     draw.text((x,y),labels[i],(0,0,0))
280   img.save(jpeg,'JPEG')
```

```
1 import clusters
2
3 (blognames, words, data) = clusters.readfile('blogdata.txt')
4 cluster = clusters.hcluster(data)
5
6 clusters.printclust(cluster, labels=blognames)
7
8 clusters.drawdendrogram(cluster, blognames, jpeg='blogclust.jpg')
```

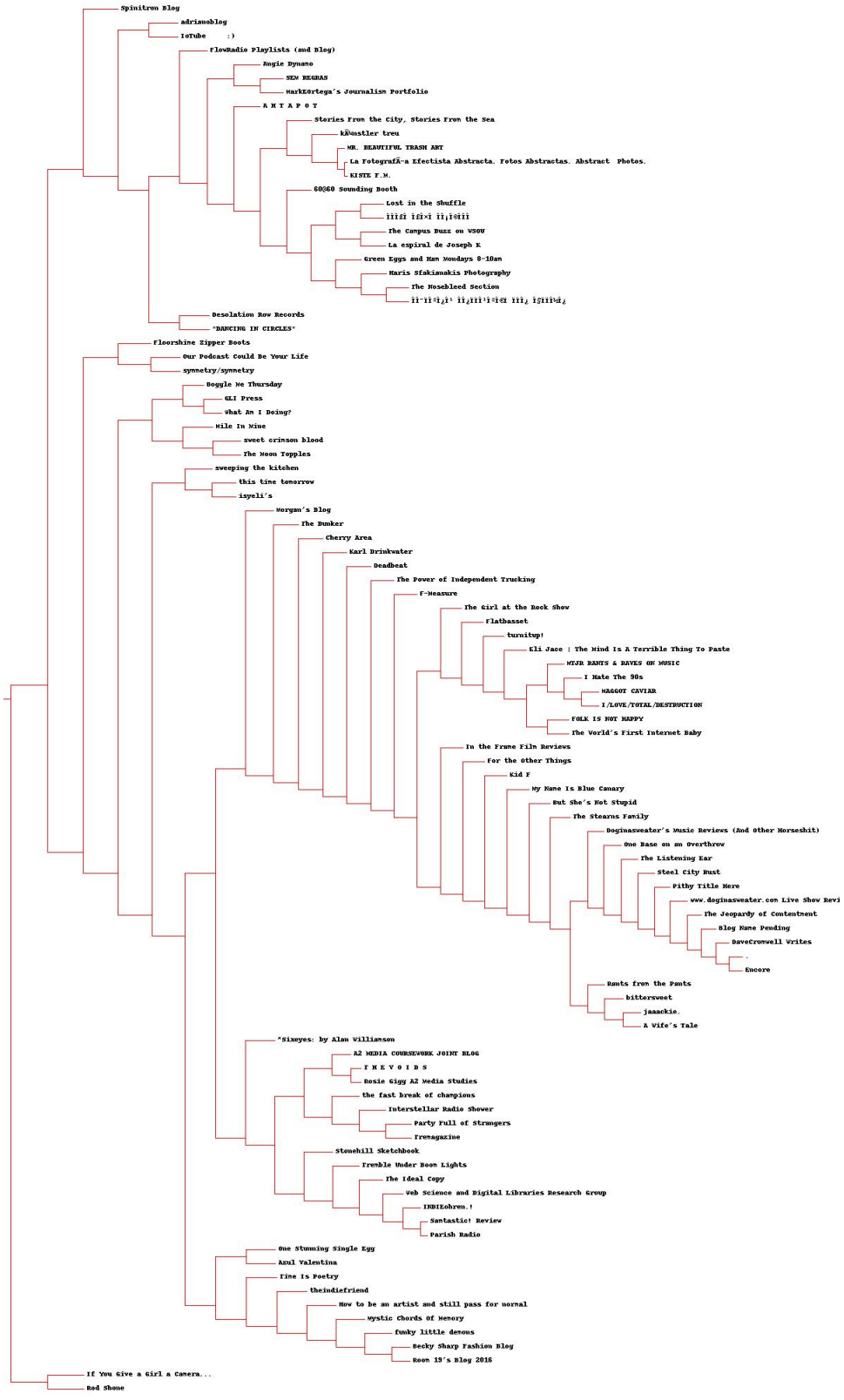

Figure 6: The dendrogram of clusters of 100 blog pages

Problem 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 this problem, I just wrote a short python code to call for the necessary functions in cluster.py and print out the information needed. Below is just a sample of part of the result, to see the full list please check the kcentroid.txt file in folder Problem3 on my github account.

```
Iteration 0
Iteration 1
Iteration 2
Iteration 3
Iteration 4
Iteration 5
Iteration for k=5 is: 5
Centroid 0: Web Science and Digital Libraries Research Group
Centroid 0: Stonehill Sketchbook
Centroid 0: Samtastic! Review
Centroid 0: INDIEohren.!
Centroid 0: Parish Radio
Centroid 0: Rod Shone
Centroid 1: Flatbasset
Centroid 1: Party Full of Strangers
```

Figure 7: Sample of part of kcentroid.txt

```
import clusters

blognames,words,data=clusters.readfile('blogdata.txt')
def k5():
  (kcluster, t) = clusters.kcluster(data, k=5)
  print "Iteration for k=5 is: " + str(t)

  k=0
  while k < 5:
    for r in kcluster[k]:
      print "Centroid " + str(k) + ": " + blognames[r]
    k+=1
  print "\n"

def k10():
  (kcluster, t) = clusters.kcluster(data, k=10)
  print "Iteration for k=10 is: " + str(t)

  k=0
  while k < 10:
    for r in kcluster[k]:
      print "Centroid " + str(k) + ": " + blognames[r]
    k+=1
  print "\n"

def k20():
  (kcluster, t)= clusters.kcluster(data, k=20)
  print "Iteration for k =20 is: " + str(t)

  k=0
  while k < 20:
    for r in kcluster[k]:
      print "Centroid " + str(k) + ": " + blognames[r]
    k+=1
  print "\n"

k5()
k10()
k20()
```

Problem 4

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

Answer

Again, I just called for the necessary functions in cluster.py to solve this problem. The functions need for this problem are:
readfile() - to process the data in blogdata.txt
scaledown() - to calculate the distance of the points and iterate
draw2d() - to draw the image based on the result from scaledown()

```
326 3059.12408306
327 3058.51664045
328 3058.05890442
329 3057.78307015
330 3057.56367522
331 3057.44346497
332 3057.44656707
333
```

Figure 8: Sample of part of scaledowniterations.txt

As you can see, the program stopped at iteration 332, because after iteration 331, the error value starts to increase.

```
1 #!/usr/local/bin/python
2
3 import clusters
4
5 (blognames, words, data)=clusters.readfile('blogdata.txt')
6
7 daata = clusters.scaledown(data)
8
9 clusters.draw2d(daata, blognames, jpeg='MDS.jpg')
```

Figure 9: Image of the 100 blog pages based on the Multidimensional Scaling