Srividya Majeti

Assignment 8

CS 532: Introduction to Web Science
Dr. Michael Nelson
Spring 2016

April 7, 2016

Contents

1 Question 1 1

2 Question 2 9

3 Question 3 13

4 Question 4 17

5 Extra-Credit Question-5 25

References 31

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) 500 terms, this is *after* the criteria on p. 32 (slide 7) has been satisfied.

Following are the steps I have taken to the solve the problem:

- First I grabbed 100 URIs using the method discussed in class. I stored all the URIs in a set data-structure which has a property of storing only unique values, this is illustrated in the function 'getUrl'. In function 'writeUrlToFile' I have written the 2 URIs mentioned in the question and all the grabbed URIs to a file 'get100URls' . This code is listed in Listing 1.1.

- Later I got the atoms for each blog URI by appending 'feeds/posts/default?max-results=500'. I wrote the atom URIs to a file 'getAtomsFor100Urls'. This function is listed in Listing 1.2.
- I downloaded the 'generateFeedVector.py' from PCI book code and ran it for the atom URIs to get the blog data with word count and words. But I realized that few blogs have more than one page.
- The python code listed in Listing 1.3 results a text file 'noOfPagesForEachBlog.txt' with number of pages and title of the blog.
- To solve the problem of multiple pages I made few modifications to 'generateFeedVector.py'. First I requested each URI and recursively checked if the blog has 'rel="next"' using the library BeautifulSoup and got the links for all the pages for each blog. I stored the resulted links for respective blogs in a list 'blogUrlList'. This is illustrated in function 'getBlogPagesURLs'.
- If a blog has more than 1 page then I ran the 'generateFeedVector' function for the first page and stored the word count in 'wc', then ran the function for the URIs in the 'blogUrlList' and stored the word count in 'nextwc'. Futhermore I consolidated both the dictionaries 'wc' and 'nextwc'. This code is listed in Listing 1.4
- Therefore I generated the output file 'blogdata.txt' which has a blog matrix with blog title as identifier for each blog. This text file is uploaded to github at `https://github.com/majetisiri/cs532-s16/blob/master/a8/q1-blogdata.txt`

Code Listing

```
def get100BlogUrl():
  link = ”http://www.blogger.com/next−blog?navBar=true&
      blogID=3471633091411211117”
  set = Set()

  while len(set) < 100:
    r = requests.get(link, allow_redirects=True)
    uri= r.url

    if len(uri) > 0:
        uri = uri.lower()

        parsedUrl = urlparse.urlparse(uri)
        parsedUrl = parsedUrl.scheme + ’://’ + parsedUrl.
            netloc + ’/’

        set.add(parsedUrl)
        print len(set)
        print parsedUrl
  return set

def writeUrlToFile(data):
  file= open(’get100Urls’,’w’)
  file.write(’http://f−measure.blogspot.com/’+’\n’)
  file.write(’http://ws−dl.blogspot.com/’+’\n’)

  for item in data:
    file.write(item+’\n’)
```

Listing 1.1. Function for getting 100 unique blog URIs

Code Listing

```
def getAtoms():
  file= open(’get100Urls’,’r’)
  f1= open(’getAtomsFor100Urls’,’w’)
  add= ”feeds/posts/default?max−results=500”
  for uri in file:
    uri= uri.strip()+add
    f1.write(uri+”\n”)
```

Listing 1.2. Function for getting atom URIs

Code Listing

```
import os
import sys
import urllib
import time
import feedparser

from bs4 import BeautifulSoup

def checkNextPage(url):

    f = urllib.urlopen(url)
    soup = BeautifulSoup(f.read(), from_encoding=f.info().
        getparam('charset'))

    try:
        link = soup.find('link', rel='next',href = True)['
            href']
    except TypeError:
        link = None
    return link


def getPages():
    feedlist    = open('getAtomsFor100Urls').readlines()

    for url in feedlist:
    d = feedparser.parse(url)
    title = d['feed']['title']
    count     = 1
    nextLink = checkNextPage( url )

    while nextLink:
      nextLink = checkNextPage( nextLink )
      count += 1

    print u'|'.join((str(count),title)).encode('utf-8').
        strip()

getPages()
```

Listing 1.3. Python code for getting number of pages for each blog

Code Listing

```python
def getBlogPagesURLs(url, blogUrlList=[]):
  req = requests.get(url)
  soup = BeautifulSoup(req.text)
  nextLink = soup.find('link', rel='next',href = True)
  if nextLink is not None:
    nextLink = nextLink['href']
    blogUrlList.append(nextLink)
    getBlogPagesURLs(nextLink, blogUrlList)
  return blogUrlList

def getwordcounts(url):
  d=feedparser.parse(url)
  wc={}

  for e in d.entries:
    if 'summary' in e:
      summary=e.summary
    else:
      summary=e.description

    words=getwords(e.title+' '+summary)
    for word in words:
      wc.setdefault(word,0)
      wc[word]+=1
  print d.feed.title
  return d.feed.title,wc

def getwords(html):
  txt=re.compile(r'<[^>]+>').sub('',html)
  words=re.compile(r'[^A-Z^a-z]+').split(txt)
  return [word.lower() for word in words if word!='']

def combineWC(wc, nextwc):
  if len(wc)>0 and len(nextwc)>0:
    for word, wordcount in nextwc.items():
      if word in wc:
        wc[word] = wc[word] + wordcount
      else:
        wc[word] = wordcount
    return wc
  else:
    return {}

def generateFeedVector():
  apcount={}
  wordcounts={}
  feedlist=[line for line in open('getAtomsFor100Urls')]
```

```
for feedurl in feedlist:
  try:
    blogUrlList = getBlogPagesURLs(feedurl)
    title,wc=getwordcounts(feedurl)
    for url in blogUrlList:
      title,nextwc=getwordcounts(feedurl)
      combineWC(wc,nextwc)
    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=[]
countFrequentWords=[]
for w,bc in apcount.items():
  frac=float(bc)/len(feedlist)
  if frac>0.1 and frac<0.5:
     countFrequentWords.append((w,bc))

countFrequentWords=sorted(countFrequentWords,key=lambda x: x[1], reverse = True)

for value in countFrequentWords:
  value1 = value[0]
  value2 = value[1]
  length = len(wordlist)
  if(length < 500):
    wordlist.append(value1)
  else:
    break

stop_words_list = [line.rstrip('\r\n') for line in open('stopWordList.txt')]

out=file('blogdata.txt','w')
out.write('Blog')
for word in wordlist:
  word1 = word.encode('UTF-8')
  out.write('\t%s' % word1)
out.write('\n')
for blog,wc in wordcounts.items():
  blogName = blog.encode('UTF-8')
  print blog
  print blogName
  out.write(blogName)
  for word in wordlist:
        if word not in stop_words_list:
        if word in wc:
          out.write('\t%d' % wc[word])
        else:
          out.write('\t0')
    out.write('\n')
```

Listing 1.4. Function for getting feed vector for each blog

2

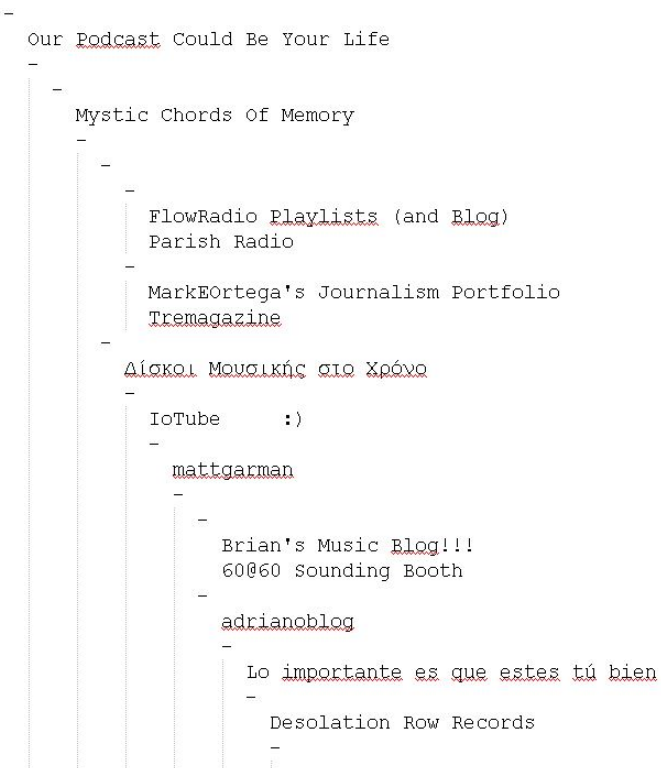
Question 2

Create an ASCII and JPEG dendrogram that clusters (i.e., HAC) the most similar blogs (see slides 12 and 13). Include the JPEG in your report and upload the ascii file to github (it will be too unwieldy for inclusion in the report).

Following are the steps I have taken to solve the problem:

- I downloaded the python code ‘clusters.py’ from the ‘Programming Collective Intelligence’ book by ‘Toby Segaran’. I used this for questions 2, 3 and 4.
- I imported the ‘clusters.py’ and used the code described in ‘presentation slide 12’ to create an ASCII that clusters the most similar blogs. This code is in Listing 2.1

- The output ascii file is uploaded to github at `https://github.com/majetisiri/cs532-s16/blob/master/a8/q2-AsciiOutput.txt`. The sample output is illustrated in Figure 2.1.



```
-
  Our Podcast Could Be Your Life
  -
    -
      Mystic Chords Of Memory
      -
        -
          -
            FlowRadio Playlists (and Blog)
            Parish Radio
          -
            MarkEOrtega's Journalism Portfolio
            Tremagazine
        -
          Δίσκοι Μουσικής σιο Χρόνο
          -
            IoTube     :)
            -
              mattgarman
              -
                -
                  Brian's Music Blog!!!
                  60@60 Sounding Booth
                -
                  adrianoblog
                  -
                    Lo importante es que estes tú bien
                    -
                      Desolation Row Records
                      -
```

Fig. 2.1. Sample ascii output

- Furthermore to get the JPEG dendogram I used ‘clusters.py’ and the code from ‘presentation slide 13’. This code is in Listing 2.2

- The output JPEG that clusters the most similar blogs is illustrated in the Figure 2.2

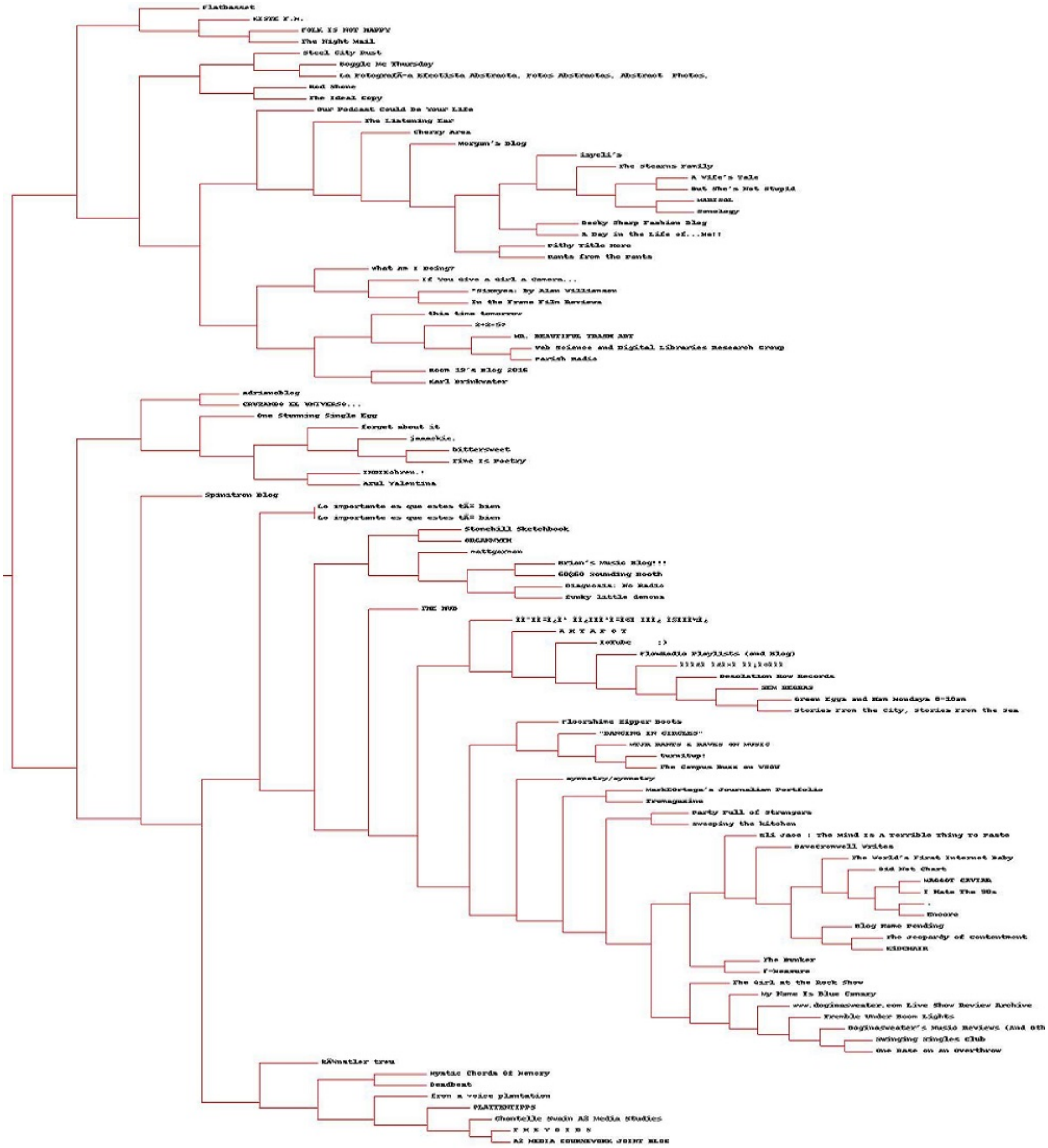


Fig. 2.2. JPEG dendogram

Code Listing

```
import clusters

def generateAscii():
    blognames,words,data=clusters.readfile('blogdata.txt')
    clust=clusters.hcluster(data)
    clusters.printclust(clust,labels=blognames)

generateAscii()
```

Listing 2.1. Python code for generating ASCII

Code Listing

```
import clusters

def drawDendogram():
  blognames,words,data=clusters.readfile('blogdata.txt')
  clust=clusters.hcluster(data)
  clusters.drawdendrogram(clust,blognames,jpeg='blogclust.jpg')

drawDendogram()
```

Listing 2.2. Python code for generating JPEG dendogram

3

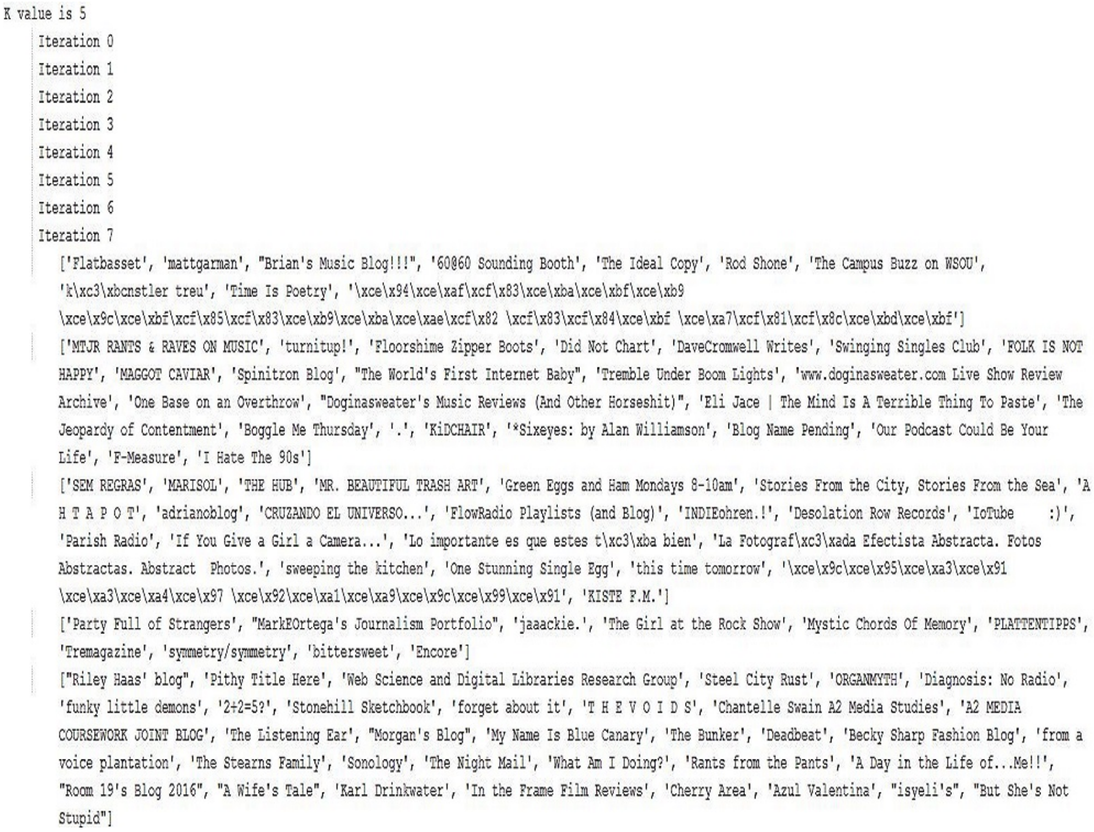
Question 3

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?

Following are the steps I have taken to solve the problem:

- I imported the 'clusters.py' mentioned in 'question 2' and used the code described in 'presentation slide 18' to cluster the blogs using K-Means, using k= 5, 10, 20. The output prints the values in each centroid, for each value of k and also the number of iterations required for each value of k. This code is in Listing 3.1

- The output file is uploaded to github at `https://github.com/majetisiri/cs532-s16/blob/master/a8/q3-numberOfIterationsAndValuesInEachCentroid.txt`. The sample output is illustrated in Figure 3.1.



```
K value is 5
    Iteration 0
    Iteration 1
    Iteration 2
    Iteration 3
    Iteration 4
    Iteration 5
    Iteration 6
    Iteration 7
      ['Flatbasset', 'mattgarman', "Brian's Music Blog!!!", '60@60 Sounding Booth', 'The Ideal Copy', 'Rod Shone', 'The Campus Buzz on WSOU',
      'k\xc3\xbcnstler treu', 'Time Is Poetry', '\xce\x94\xce\xaf\xcf\x83\xce\xba\xce\xbf\xce\xb9
      \xce\x9c\xce\xbf\xcf\x85\xcf\x83\xce\xb9\xce\xba\xce\xae\xcf\x82 \xcf\x83\xcf\x84\xce\xbf \xce\xa7\xcf\x81\xcf\x8c\xce\xbd\xce\xbf']
      ['MTJR RANTS & RAVES ON MUSIC', 'turnitup!', 'Floorshime Zipper Boots', 'Did Not Chart', 'DaveCromwell Writes', 'Swinging Singles Club', 'FOLK IS NOT
      HAPPY', 'MAGGOT CAVIAR', 'Spinitron Blog', "The World's First Internet Baby", 'Tremble Under Boom Lights', 'www.doginasweater.com Live Show Review
      Archive', 'One Base on an Overthrow', "Doginasweater's Music Reviews (And Other Horseshit)", 'Eli Jace | The Mind Is A Terrible Thing To Paste', 'The
      Jeopardy of Contentment', 'Boggle Me Thursday', '.', 'KiDCHAIR', '*Sixeyes: by Alan Williamson', 'Blog Name Pending', 'Our Podcast Could Be Your
      Life', 'F-Measure', 'I Hate The 90s']
      ['SEM REGRAS', 'MARISOL', 'THE HUB', 'MR. BEAUTIFUL TRASH ART', 'Green Eggs and Ham Mondays 8-10am', 'Stories From the City, Stories From the Sea', 'A
      H T A P O T', 'adrianoblog', 'CRUZANDO EL UNIVERSO...', 'FlowRadio Playlists (and Blog)', 'INDIEohren.!', 'Desolation Row Records', 'IoTube     :)',
      'Parish Radio', 'If You Give a Girl a Camera...', 'Lo importante es que estes t\xc3\xba bien', 'La Fotograf\xc3\xada Efectista Abstracta. Fotos
      Abstractas. Abstract  Photos.', 'sweeping the kitchen', 'One Stunning Single Egg', 'this time tomorrow', '\xce\x9c\xce\x95\xce\xa3\xce\x91
      \xce\xa3\xce\xa4\xce\x97 \xce\x92\xce\xa1\xce\xa9\xce\x9c\xce\x99\xce\x91', 'KISTE F.M.']
      ['Party Full of Strangers', "MarkEOrtega's Journalism Portfolio", 'jaaackie.', 'The Girl at the Rock Show', 'Mystic Chords Of Memory', 'PLATTENTIPPS',
      'Tremagazine', 'symmetry/symmetry', 'bittersweet', 'Encore']
      ["Riley Haas' blog", 'Pithy Title Here', 'Web Science and Digital Libraries Research Group', 'Steel City Rust', 'ORGANMYTH', 'Diagnosis: No Radio',
      'funky little demons', '2+2=5?', 'Stonehill Sketchbook', 'forget about it', 'T H E V O I D S', 'Chantelle Swain A2 Media Studies', 'A2 MEDIA
      COURSEWORK JOINT BLOG', 'The Listening Ear', "Morgan's Blog", 'My Name Is Blue Canary', 'The Bunker', 'Deadbeat', 'Becky Sharp Fashion Blog', 'from a
      voice plantation', 'The Stearns Family', 'Sonology', 'The Night Mail', 'What Am I Doing?', 'Rants from the Pants', 'A Day in the Life of...Me!!',
      "Room 19's Blog 2016", "A Wife's Tale", 'Karl Drinkwater', 'In the Frame Film Reviews', 'Cherry Area', 'Azul Valentina', "isyeli's", "But She's Not
      Stupid"]
```

Fig. 3.1. Sample output with number of iterations and values in each centroid for k=5

Code Listing

```
import clusters
def getKmeans():
  blognames,words,data=clusters.readfile(’blogdata.txt’)
  print ”K value is 5”
  kclust=clusters.kcluster(data,k=5)
  print ”\t\t”+str([blognames[r] for r in kclust[0]])
  print ”\t\t”+str([blognames[r] for r in kclust[1]])
  print ”\t\t”+str([blognames[r] for r in kclust[2]])
  print ”\t\t”+str([blognames[r] for r in kclust[3]])
  print ”\t\t”+str([blognames[r] for r in kclust[4]])
  print ”K value is 10”
  kclust=clusters.kcluster(data,k=10)
  print ”\t\t”+str([blognames[r] for r in kclust[0]])
  print ”\t\t”+str([blognames[r] for r in kclust[1]])
  print ”\t\t”+str([blognames[r] for r in kclust[2]])
  print ”\t\t”+str([blognames[r] for r in kclust[3]])
  print ”\t\t”+str([blognames[r] for r in kclust[4]])
  print ”\t\t”+str([blognames[r] for r in kclust[5]])
  print ”\t\t”+str([blognames[r] for r in kclust[6]])
  print ”\t\t”+str([blognames[r] for r in kclust[7]])
  print ”\t\t”+str([blognames[r] for r in kclust[8]])
  print ”\t\t”+str([blognames[r] for r in kclust[9]])
  print ”K value is 20”
  kclust=clusters.kcluster(data,k=20)
  print ”\t\t”+str([blognames[r] for r in kclust[0]])
  print ”\t\t”+str([blognames[r] for r in kclust[1]])
  print ”\t\t”+str([blognames[r] for r in kclust[2]])
  print ”\t\t”+str([blognames[r] for r in kclust[3]])
  print ”\t\t”+str([blognames[r] for r in kclust[4]])
  print ”\t\t”+str([blognames[r] for r in kclust[5]])
  print ”\t\t”+str([blognames[r] for r in kclust[6]])
  print ”\t\t”+str([blognames[r] for r in kclust[7]])
  print ”\t\t”+str([blognames[r] for r in kclust[8]])
  print ”\t\t”+str([blognames[r] for r in kclust[9]])
  print ”\t\t”+str([blognames[r] for r in kclust[10]])
  print ”\t\t”+str([blognames[r] for r in kclust[11]])
  print ”\t\t”+str([blognames[r] for r in kclust[12]])
  print ”\t\t”+str([blognames[r] for r in kclust[13]])
  print ”\t\t”+str([blognames[r] for r in kclust[14]])
  print ”\t\t”+str([blognames[r] for r in kclust[15]])
  print ”\t\t”+str([blognames[r] for r in kclust[16]])
  print ”\t\t”+str([blognames[r] for r in kclust[17]])
  print ”\t\t”+str([blognames[r] for r in kclust[18]])
  print ”\t\t”+str([blognames[r] for r in kclust[19]])
getKmeans()
```

Listing 3.1. Python code for generating ASCII

4

Question 4

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

Following are the steps I have taken to solve the problem:

- I imported the ‘clusters.py’ mentioned in ‘question 2’ and used the code described in ‘presentation slide 29’ to create a JPEG of the most similar blogs using MDS. This code is in Listing 4.1

- The output JPEG file is illustrated in 4.1.

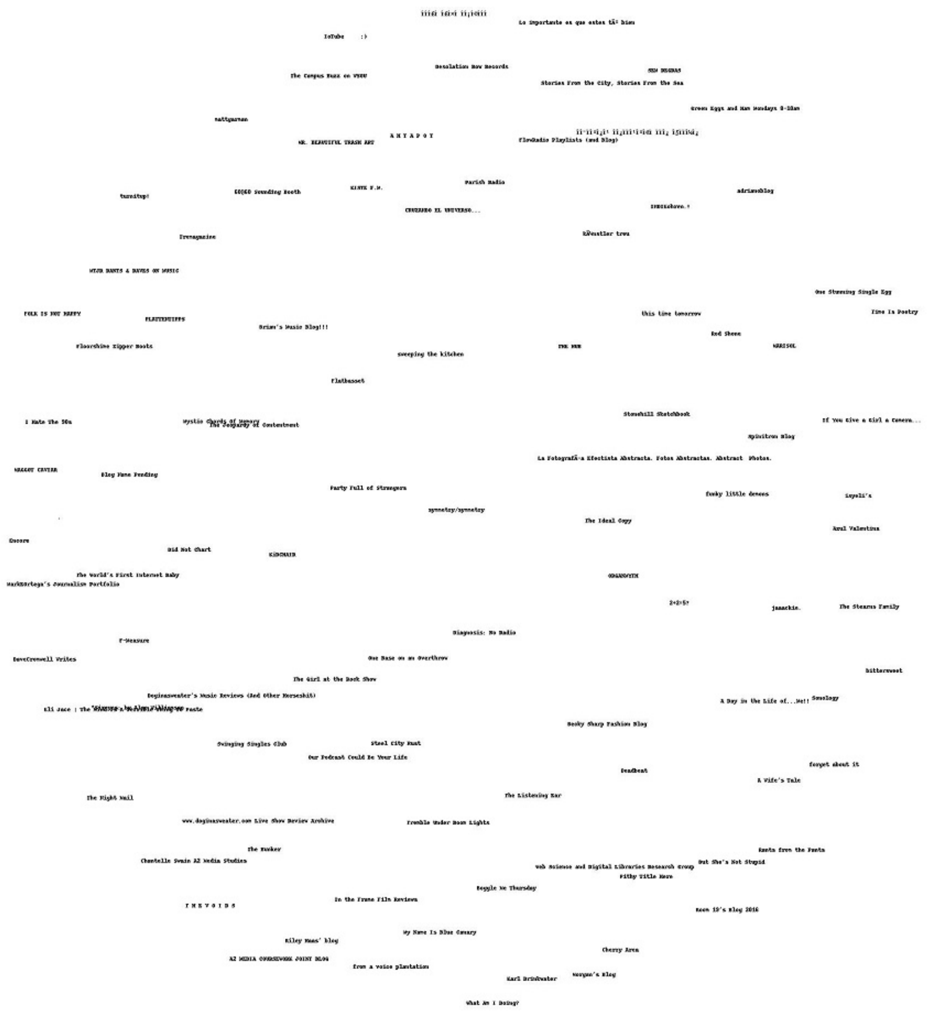


Fig. 4.1. JPEG of blogs using MDS

- To get the number of iterations I have written a print statement in function ‘scaledown(data,distance=pearson,rate=0.01)’ of ‘clusters.py’. ‘304’ iterations were required for creating the JPEG using MDS. The file ‘numberOfIterations.txt’ has ‘total error’ and ‘iteration count’.
- The python code ‘clusters.py’ that I downloaded from the PCI book is illustrated in Listing 4.2

Code Listing

```
import clusters

def createMDS():

    blognames,words,data=clusters.readfile('blogdata.txt')
    coords=clusters.scaledown(data)
    clusters.draw2d(coords,blognames,jpeg='blogs2d.jpg')

createMDS()
```

Listing 4.1. Python code for creating MDS

Code Listing

```
from PIL import Image,ImageDraw

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

  colnames=lines[0].strip().split('\t')[1:]
  rownames=[]
  data=[]
  for line in lines[1:]:
    p=line.strip().split('\t')
    rownames.append(p[0])
    data.append([float(x) for x in p[1:]])
  return rownames,colnames,data


from math import sqrt

def pearson(v1,v2):
  sum1=sum(v1)
  sum2=sum(v2)
  sum1Sq=sum([pow(v,2) for v in v1])
  sum2Sq=sum([pow(v,2) for v in v2])
  pSum=sum([v1[i]*v2[i] for i in range(len(v1))])
  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
  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)

    for i in range(len(clust)):
      for j in range(i+1,len(clust)):
        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)

    mergevec=[
    (clust[lowestpair[0]].vec[i]+clust[lowestpair[1]].vec[i])/2.0
    for i in range(len(clust[0].vec))]

    newcluster=bicluster(mergevec,left=clust[lowestpair[0]],
                         right=clust[lowestpair[1]],
                         distance=closest,id=currentclustid)

    currentclustid-=1
    del clust[lowestpair[1]]
    del clust[lowestpair[0]]
    clust.append(newcluster)

  return clust[0]

```

```
def printclust(clust,labels=None,n=0):
  for i in range(n): print ' ',
  if clust.id<0:
    print '-'
  else:
    if labels==None: print clust.id
    else: print labels[clust.id]

  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):
  if clust.left==None and clust.right==None: return 1
  return getheight(clust.left)+getheight(clust.right)

def getdepth(clust):
  if clust.left==None and clust.right==None: return 0
  return max(getdepth(clust.left),getdepth(clust.right))+clust.distance


def drawdendrogram(clust,labels,jpeg='clusters.jpg'):
  h=getheight(clust)*20
  w=1200
  depth=getdepth(clust)

  scaling=float(w-150)/depth

  img=Image.new('RGB',(w,h),(255,255,255))
  draw=ImageDraw.Draw(img)

  draw.line((0,h/2,10,h/2),fill=(255,0,0))

  drawnode(draw,clust,10,(h/2),scaling,labels)
  img.save(jpeg,'JPEG')

def drawnode(draw,clust,x,y,scaling,labels):
  if clust.id<0:
    h1=getheight(clust.left)*20
    h2=getheight(clust.right)*20
    top=y-(h1+h2)/2
    bottom=y+(h1+h2)/2
    ll=clust.distance*scaling
    draw.line((x,top+h1/2,x,bottom-h2/2),fill=(255,0,0))
    draw.line((x,top+h1/2,x+ll,top+h1/2),fill=(255,0,0))
    draw.line((x,bottom-h2/2,x+ll,bottom-h2/2),fill=(255,0,0))
    drawnode(draw,clust.left,x+ll,top+h1/2,scaling,labels)
    drawnode(draw,clust.right,x+ll,bottom-h2/2,scaling,labels)
  else:
    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):
  ranges=[(min([row[i] for row in rows]),max([row[i] for row in rows]))
  for i in range(len(rows[0]))]

  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)]
    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 bestmatches==lastmatches: break
    lastmatches=bestmatches

    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

  for i in range(len(v1)):
    if v1[i]!=0: c1+=1
    if v2[i]!=0: c2+=1
    if v1[i]!=0 and v2[i]!=0: shr+=1

  return 1.0-(float(shr)/(c1+c2-shr))

def scaledown(data,distance=pearson,rate=0.01):
  n=len(data)

  realdist=[[distance(data[i],data[j]) for j in range(n)]
             for i in range(0,n)]

  loc=[[random.random(),random.random()] for i in range(n)]
  fakedist=[[0.0 for j in range(n)] for i in range(n)]

  lasterror=None
  for m in range(0,1000):
    for i in range(n):
      for j in range(n):
        fakedist[i][j]=sqrt(sum([pow(loc[i][x]-loc[j][x],2)
                                 for x in range(len(loc[i]))
                                   ]))

    grad=[[0.0,0.0] for i in range(n)]

    totalerror=0
    for k in range(n):
      for j in range(n):
        if j==k: continue
        errorterm=(fakedist[j][k]-realdist[j][k])/realdist[j
            ][k]
        grad[k][0]+=((loc[k][0] - loc[j][0])/fakedist[j][k])*
            errorterm
        grad[k][1]+=((loc[k][1] - loc[j][1])/fakedist[j][k])*
            errorterm
        totalerror+=abs(errorterm)
    print totalerror

    if lasterror and lasterror<totalerror: break
    lasterror=totalerror

    for k in range(n):
      loc[k][0]-=rate*grad[k][0]
      loc[k][1]-=rate*grad[k][1]

  return loc

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

Listing 4.2. Python code ‘clusters.py’ from PCI

5

Extra-Credit 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 500 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 500 terms and instead come up with TFIDF scores for all the terms and then choose the top 500 from that list, but I'm trying to limit the amount of work necessary.

Following are the steps I have taken to solve the problem:

- I calculated the TFIDF values similar to assignment 3.
- I got the JPEG dendogram using the same python code as question 2 5.1

- The output JPEG that clusters the most similar blogs based on TFIDF values is illustrated in the Figure 5.1

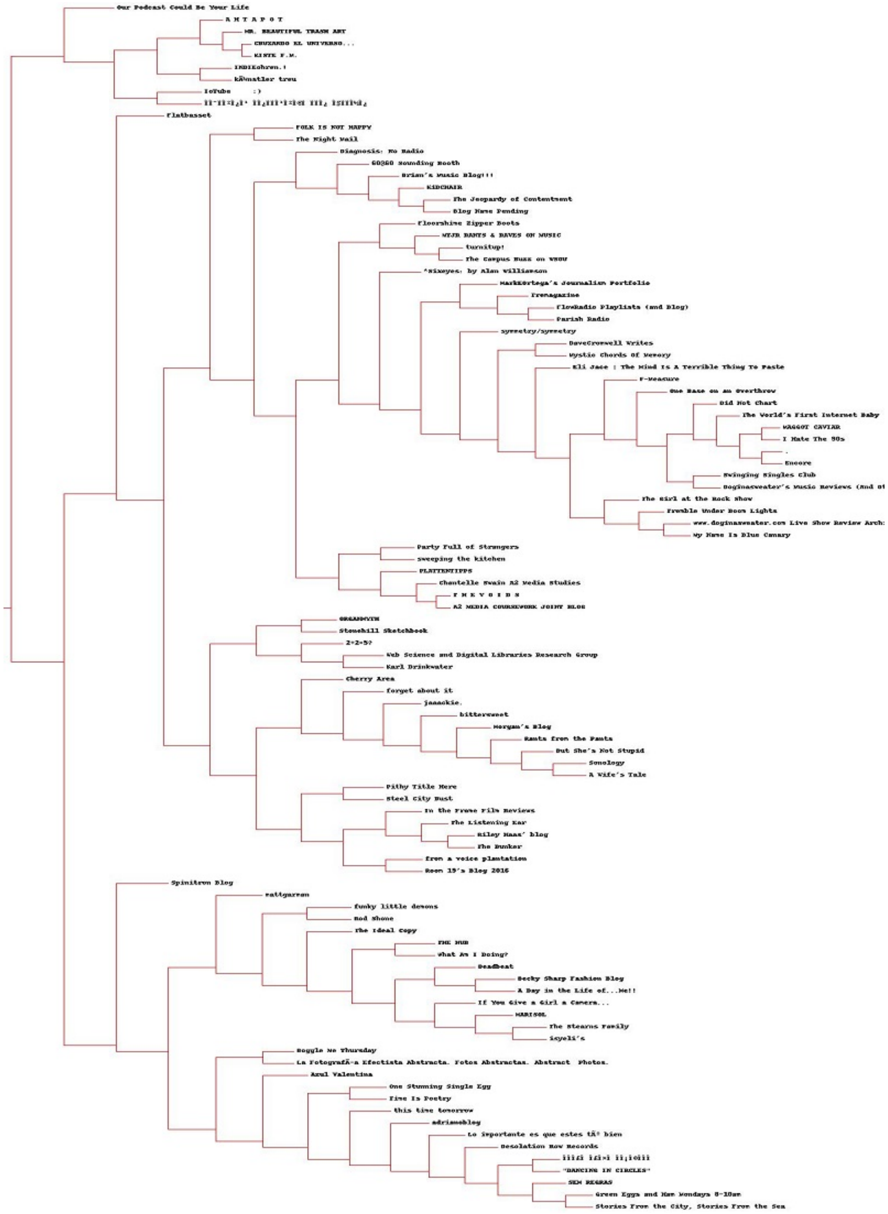


Fig. 5.1. JPEG dendogram

- Therefore I generated the output file 'blogdataWithTFIDF.txt' which has a blog matrix with blog title as identifier for each blog. This text file is uploaded to github at `https://github.com/majetisiri/cs532-s16/blob/master/a8/q5-blogdataWithTFIDF.txt`
- By comparing and contrasting we can say that the blog dendogram based on TFIDF has more clusters than the blog dendogram with frequency count in question 1. By looking at the dendograms it seems there is a much difference, only the hierarchy of clusters are changing but the clusters are around the same blog with different hierarchy. This is illustrated in Figure 5.2.

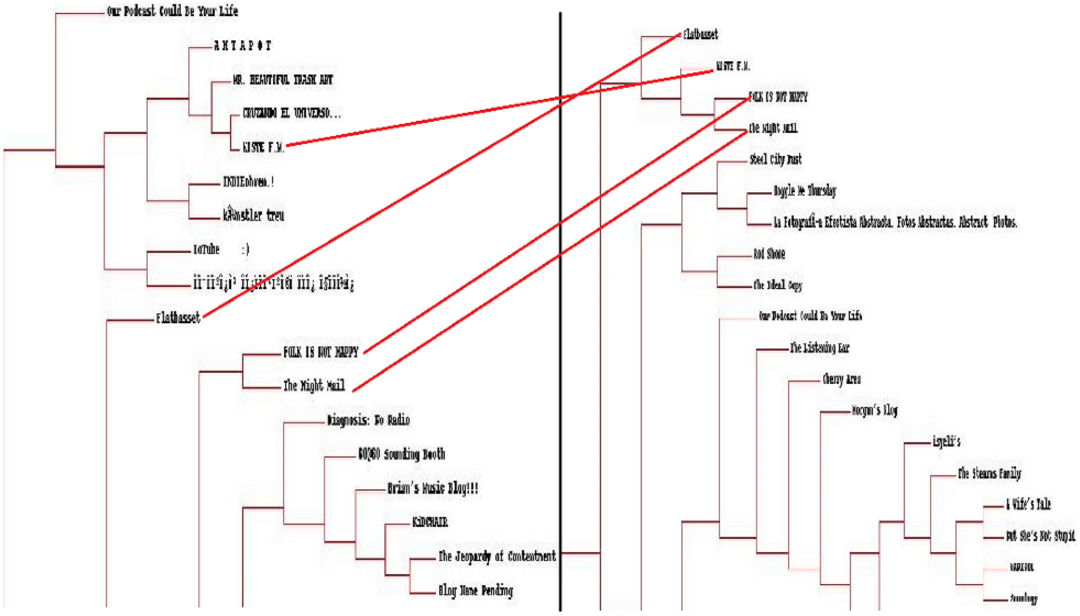


Fig. 5.2. comparing and contrasting dendograms from question 1 and question 5

Code Listing

```
import feedparser
import re
import sys
import math

def getwordcounts(url):
  d=feedparser.parse(url)
  wc={}

  for e in d.entries:
    if 'summary' in e:
      summary=e.summary
    else:
      summary=e.description

    words=getwords(e.title+' '+summary)
    for word in words:
      wc.setdefault(word,0)
      wc[word]+=1
  print d.feed.title
  return d.feed.title,wc

def getwords(html):
  txt=re.compile(r'<[^>]+>').sub('',html)
  words=re.compile(r'[^A-Z^a-z]+').split(txt)
  return [word.lower() for word in words if word!='']

def generateFeedVector():
  apcount={}
  wordcounts={}
  iteration = 1
  feedlist=[line for line in file('getAtomsFor100Urls')]
  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
    iteration+=1

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

  countFrequentWords=sorted(countFrequentWords,key=lambda x: x[1], reverse = True)

  for value in countFrequentWords:
    value1 = value[0]
    value2 = value[1]
    length = len(wordlist)
    if(length < 500):
      wordlist.append(value1)
    else:
      break

  out=file('blogdata.txt','w')
  out.write('Blog')
  for word in wordlist:
    word1 = word.encode('UTF-8')
    out.write('\t%s' % word1)
  out.write('\n')
  for blog,wc in wordcounts.items():
    blogName = blog.encode('UTF-8')
    print blog
    print blogName
    out.write(blogName)
    for word in wordlist:
      if word in wc:
      # EDITED CODE
        termFrequency = wc[word]/float(len(wc))
        inverseDocumentFrequency = logBase2(iteration/float(apcount[word]))
        tfIdf = termFrequency*inverseDocumentFrequency
        out.write('\t%f' % tfIdf)
      else:
        out.write('\t0')
    out.write('\n')

def logBase2(number):
  return math.log(number) / math.log(2)

generateFeedVector()
```

Listing 5.1. Python code for calculating TFIDF

References

1. get requests for URI: http://docs.python-requests.org/en/master/user/quickstart/, 2016, A Kenneth Reitz Project
2. Python code for generating feed vector from PCI book. Igraph Tutorial. https://github.com/cataska/programming-collective-intelligence-code/blob/master/chapter3/generatefeedvector.py, 2007, Toby Segaran
3. Python code for clusters from PCI book. Download GraphML for Karate Club. https://github.com/cataska/programming-collective-intelligence-code/blob/master/chapter3/clusters.py, 2007, Toby Segaran