Web Science cs532: Assignment #8

Due on Thursday, April 8, 2016

Dr.Michael.L.Nelson 4:20pm

Zetan Li

Contents

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

SOLUTION

Grab the blogs

To parse the entries' title, we have first get feed uri from raw blog page. According to slides, we can use curl to grab random blog page from uri
https://www.blogger.com/next-blog?navBar=true&blogID=3471633091411211117

The option -L enable curl to auto-tracking the redirection, and -w option enable curl to print out the final effective uri of the blog page.
Ideally, we can get 100 different blogs when we visit this uri 100 times. However, in fact, we get bunch of duplicated uris from it after running query for many times. So we have to make the uris unique. The -u option in sort can fulfill this task.
(The raw blog page is not uploaded, only uri list is uploaded to github. In the sub-directory rawBlogs)

Listing 1: Shell script to download 100 different blog page from internet

```
#!/bin/bash
curl --silent "http://f-measure.blogspot.com/" > rawBlogs/blog001
curl --silent "http://ws-dl.blogspot.com/" > rawBlogs/blog002
rm -f rawBlogs/uriList
touch rawBlogs/uriList
echo "blog001 http://f-measure.blogspot.com/" >> rawBlogs/uriList
echo "blog002 http://ws-dl.blogspot.com/" >> rawBlogs/uriList
for (( i = 3; i <= 350; i++ )); do
    num=`seq -f%03g $i $i`
    uri=`curl -Ls -o  rawBlogs/blog$num -w %{url_effective} "https://www.blogger.
        com/next-blog?navBar=true&blogID=3471633091411211117"`

    echo "blog$num $uri" >> rawBlogs/uriList
done
#remove duplicate uri and page files
sort -u -k2 rawBlogs/uriList > rawBlogs/uriListTmp
sort -k1 rawBlogs/uriListTmp> rawBlogs/uriList
rm rawBlogs/uriListTmp
for file in `cat rawBlogs/uriList | cut -d' ' -f1`; do
    mv rawBlogs/$file rawBlogs/pages/
```

Problem 1 continued on next page...

```
done
```

Get feed uri

Now with 100 blog pages, we can retrieve rss feed uri from them by BeautifulSoup. According to slides, the rss link is in the link label with type "application/rss+xml"

Listing 2: Python code to retrieve the feed uri list

```
import requests
import sys
from bs4 import BeautifulSoup

def get_rss(fname):
     page=open('rawBlogs/pages/'+fname)
     text=page.read()
     page.close()
     soup=BeautifulSoup(text)
     links=soup.find_all('link',{'type':'application/rss+xml'})
     if links:
          return str(links[0]['href'])
     return None

outfile=open('feeders.txt','w')
pathfile=open('rawBlogs/uriList')
paths=pathfile.readlines()
for line in paths:
     filename=line.split(' ')[0]
     uri=get_rss(filename)
     if uri :
          outfile.write(uri+'\n')
pathfile.close()
outfile.close()
```

Generate matrix

The code on PCI book [1] can generate the matrix for us, the only modification on the code is to add a limit to the column we pick.(Notice the code on line 52)

Listing 3: Python code to generate matrix

```
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('feeders.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():
  #print blog
  try:
    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')
```

For some blog title that contains non-ascii code, we have to encode them in utf-8(code line 60)
The matrix is stored in file "blogdata.txt".

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).

SOLUTION

According to slides, we can pick hcluster function from PCI to finish this task.

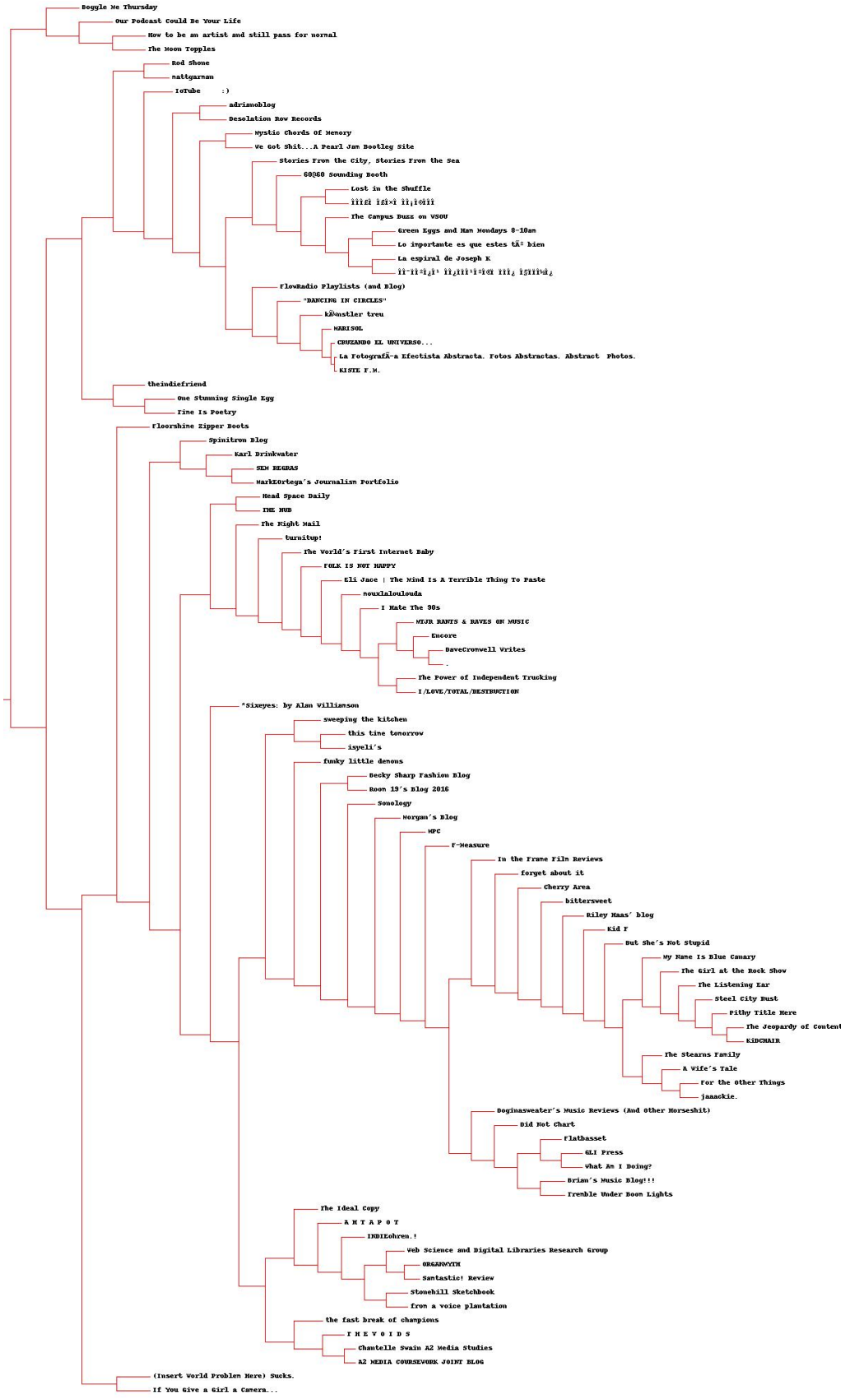
Listing 4: Python code to plot dendrogram

```
blogs,colnames,data=readfile('blogdata.txt')
cluster=hcluster(data)
drawdendrogram(cluster,blogs,jpeg='p2_dendrogram.jpg')
printclust(cluster,labels=blogs)
```

We running this code in the way like
pythonp2.py>p2_ascii.txt
so we can get ascii output into a file instead of printing them on console.
The complete code is in p2.py .
The ascii version of dendrogram is included in p2_ascii.txt

Problem 2 continued on next page...

Figure 1: Dendrogram of blogs(large image, check p2_dendrogram on github)



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?

SOLUTION

The function kcluster in the PCI book can do this work. The only modification we have to do is to return the iteration time in the function then print it out.

Listing 5: Python code to get k clusters

```
from PIL import Image,ImageDraw
import random

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

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]))]
```

```
45
    # 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)]

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

55    # 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):
60        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
65    if bestmatches==lastmatches: break
      lastmatches=bestmatches

      # Move the centroids to the average of their members
      for i in range(k):
70      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]
75        for j in range(len(avgs)):
            avgs[j]/=len(bestmatches[i])
          clusters[i]=avgs

    return bestmatches,t
80


blogs,colnames,data=readfile('blogdata.txt')
for i in [5,10,20] :
  outfile=open('p3_k'+str(i)+'.txt','w')
85 kclust,iternum=kcluster(data,k=i)
  outfile.write('Iterations = %d\n'%iternum)

  for cluster in kclust:
    outfile.write('[')
90  for blogidx in cluster:
      outfile.write(blogs[blogidx]+', ')
    outfile.write(']\n')

  outfile.close()
```

For output, check p3_k5.txt for k=5, p3_k10.txt for k=10 and p3_k20.txt for k=20 in the github.

Problem 4

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

SOLUTION

The function scaledown in the PCI book can finish this task.
Here, we let the function return the iteration time at the same time.

Listing 6: Python code to get MDS

```
from PIL import Image,ImageDraw
import random

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

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

  # The real distances between every pair of items
  realdist=[[distance(data[i],data[j]) for j in range(n)]
```

```
            for i in range(0,n)]

  # Randomly initialize the starting points of the locations in 2D
  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):
    # Find projected distances
    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]))]))

    # Move points
    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
        # The error is percent difference between the distances
        errorterm=(fakedist[j][k]-realdist[j][k])/realdist[j][k]

        # Each point needs to be moved away from or towards the other
        # point in proportion to how much error it has
        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

        # Keep track of the total error
        totalerror+=abs(errorterm)
    print totalerror

    # If the answer got worse by moving the points, we are done
    if lasterror and lasterror<totalerror: break
    lasterror=totalerror

    # Move each of the points by the learning rate times the gradient
    for k in range(n):
      loc[k][0]-=rate*grad[k][0]
      loc[k][1]-=rate*grad[k][1]

  return loc,m

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')
```

```
blogs,colnames,data=readfile('blogdata.txt')
coord,it=scaledown(data)
draw2d(coord,labels=blogs,jpeg='p4_MDS.jpg')
print ('Total iteration: %d'%it)
```

Figure 2: Processing output

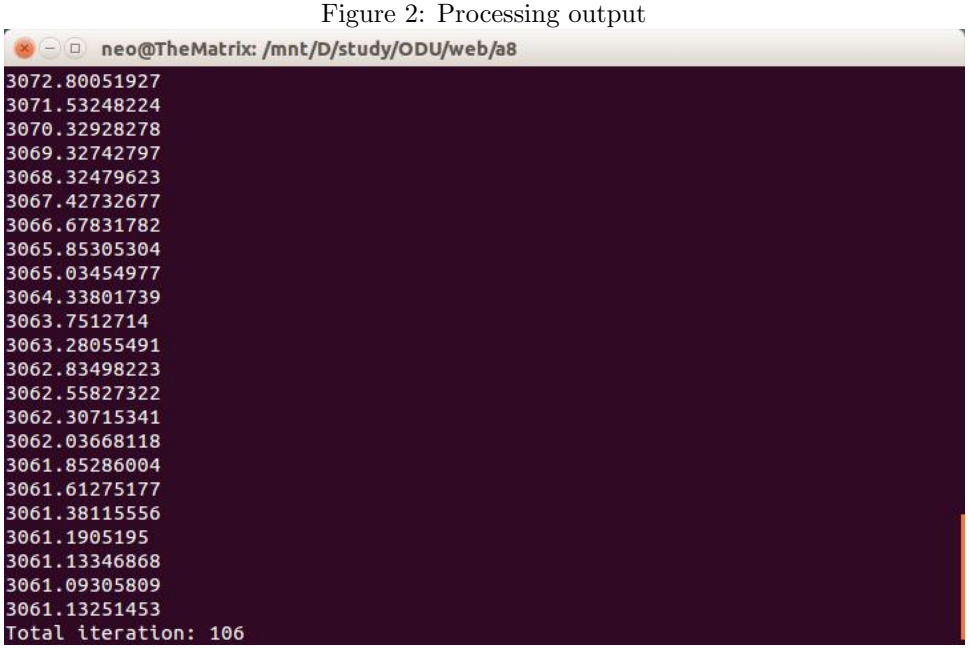


Figure 3: MDS(large image, check p4_MDS.jpg on github)

References

[1] cataska. *programming-collective-intelligence-code*, 2016 (accessed April 7, 2016).