INTRODUCTION TO WEB SCIENCES:
Assignment 9

Babitha Bokka

06 December 2014

Contents

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.

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.

Create a histogram of how many pages each blog has (e.g., 30
blogs with just one page, 27 with two pages, 29 with 3 pages and
so on).
```

1.1 Approach

1. In doing this assignment I took help from Mallika and Victor. They explained me about the packages to be installed and what to do? and what not to do?.
2. To solve this questions I wrote shell script to grab the other 98 links.
3. I wrote a short python script to append the “feeds/posts/default?max-results=500”.
4. And to get the top 500 words I modified feedparser.py.

1.2 Source Code

1.2.1 extractLinks.sh

```
#!/bin/bash

if  [ $# -ne 1 ]
then
    echo "usage <extractLinks.sh> <number of links> "
    echo "e.g.,extractLinks.sh 98 "
    exit
fi

numberOfLinks=$1

link1="http://f-measure.blogspot.com/"
link2="http://ws-dl.blogspot.com/"

echo "$link1" > blogLinksLists.txt
echo "$link2" >> blogLinksLists.txt

for (( count=1; count<=$numberOfLinks; count++ ))
do
   curl -I -L 'http://www.blogger.com/next-blog?navBar=true&blogID
   =3471633091411211117' | grep Location | tail -n 1 | cut -d" " -f2 | cut -d"?" -f 1
    >> blogLinksLists.txt
done
```

1.2.2 makingAtoms.py

```
#!/usr/bin/env python

import sys

def main():
    saveFile = open('blogLinksLists.txt', 'r')
    writeFile = open('blogLinksAtom.txt', 'w')
    add = "feeds/posts/default?max-results=500"
    for line in saveFile:
        writeFile.write(line+add)
        writeFile.write("\n")
    saveFile.close()
    writeFile.close()

if __name__ == "__main__":
    try:
        main()
    except KeyboardInterrupt:
        sys.exit(1)
```

1.2.3 generatefeedvector.py

```
#!/usr/bin/env python

import re
import sys
import feedparser

# 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!='']

def main():

  apcount={}
  wordcounts={}
  feedlist=[line for line in file('blogLinksAtom.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=[]
  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:
    # word
    value1 = value[0]
    #count
    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
    out.write(blogName)
    for word in wordlist:
      if word in wc: out.write('\t%d' % wc[word])
      else: out.write('\t0')
    out.write('\n')

if __name__ == "__main__":
  try:
      main()
  except KeyboardInterrupt:
      sys.exit(1)
```

1.2.4 pagesEachBlog.py

Could not finish to generate the Histogram.

```
#!/usr/bin/env python

import sys
import time
import socket
import urllib2
import unicodedata
from bs4 import BeautifulSoup




def main():
    saveFile = open('blogLinksLists.txt', 'r')
    writeFile = open('pagesBlog.txt', 'w')
    #add = "feeds/posts/default"
```

```
    for url in saveFile.readlines():
        try :
            response    = urllib2.urlopen(url,timeout=30)
            html_content= response.read()
            # get the html content using beautiful soup
            soup        = BeautifulSoup(html_content)

            try:
                links       = soup.find('link',rel='next',href = True)['href']
                print links
            except TypeError:
                pass
            except KeyError:
                pass

        except urllib2.HTTPError :
            pass
        except urllib2.URLError :
            pass
        except socket.timeout :
            pass




    saveFile.close()
    writeFile.close()

if __name__ == "__main__":
    try:
        main()
    except KeyboardInterrupt:
        sys.exit(1)
```

1.3 Output Files

1. The output for this question is a matix which is uploaded in Github blogdata.txt.

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).
```

2.1 Approach

1. To answer this question I used clusters.py.
2. Used the lines of code provided in the presentation.
3. Python program generateAscii.py produces ASCII results which are piped to an output file ascii.txt.
4. Also produces dendrogram blogclust.jpg.

2.2 Source Code

2.2.1 generateAscii.py

```
#!/usr/bin/env python

import sys
import clusters

def main():

    # returns blog titles, words in blog (10%-50% boundaries), list of frequency info
    blognames,words,data=clusters.readfile('blogdata.txt')

    # returns a tree of foo.id, foo.left, foo.right
    clust=clusters.hcluster(data)

    # walks tree and prints ascii approximation of a dendogram; distance measure is
    Pearson's r
    clusters.printclust(clust,labels=blognames)

if __name__ == "__main__":
    try:
        main()
    except KeyboardInterrupt:
        sys.exit(1)
```

2.2.2 clusters.py

This program is imported into question 2, 3 and 4.

```
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

  # Otherwise the height is the same of the heights of
  # each branch
  return getheight(clust.left)+getheight(clust.right)

def getdepth(clust):
  # The distance of an endpoint is 0.0
  if clust.left==None and clust.right==None: return 0

  # The distance of a branch is the greater of its two sides
  # plus its own distance
  return max(getdepth(clust.left),getdepth(clust.right))+clust.distance


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

  # width is fixed, so scale distances accordingly
  scaling=float(w-150)/depth

  # Create a new image with a white background
  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))

  # Draw the first node
  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
    # Line length
    ll=clust.distance*scaling
    # Vertical line from this cluster to children
    draw.line((x,top+h1/2,x,bottom-h2/2),fill=(255,0,0))

    # Horizontal line to left item
    draw.line((x,top+h1/2,x+ll,top+h1/2),fill=(255,0,0))

    # Horizontal line to right item
    draw.line((x,bottom-h2/2,x+ll,bottom-h2/2),fill=(255,0,0))

    # Call the function to draw the left and right nodes
    drawnode(draw,clust.left,x+ll,top+h1/2,scaling,labels)
    drawnode(draw,clust.right,x+ll,bottom-h2/2,scaling,labels)
  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

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

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

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
```

```
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')
```

2.3 Output Files

2.3.1 blogclust.jpg

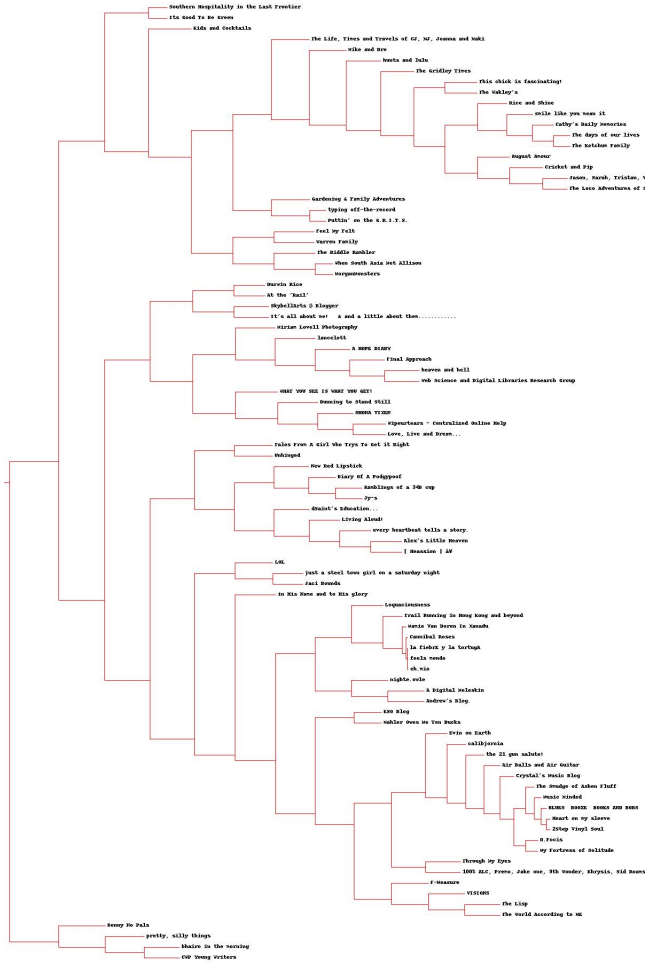


Figure 1: Dendrogram

3 Question 3:

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

3.1 Approach

1. To solve this question I used lines of code provided in the presentation (calculateK-mean.py) .

3.2 Source Code

3.2.1 calculateK-mean.py

```
#!/usr/bin/env python

import clusters

def main():

    blognames,words,data=clusters.readfile('blogdata.txt')
    print "K value is 5"
    kclust=clusters.kcluster(data,k=5)
    print "K value is 10"
    kclust=clusters.kcluster(data,k=10)
    print "K value is 20"
    kclust=clusters.kcluster(data,k=20)


if __name__ == "__main__":
    try:
        main()
    except KeyboardInterrupt:
        sys.exit(1)
```

3.3 Output Files

3.3.1 output.png

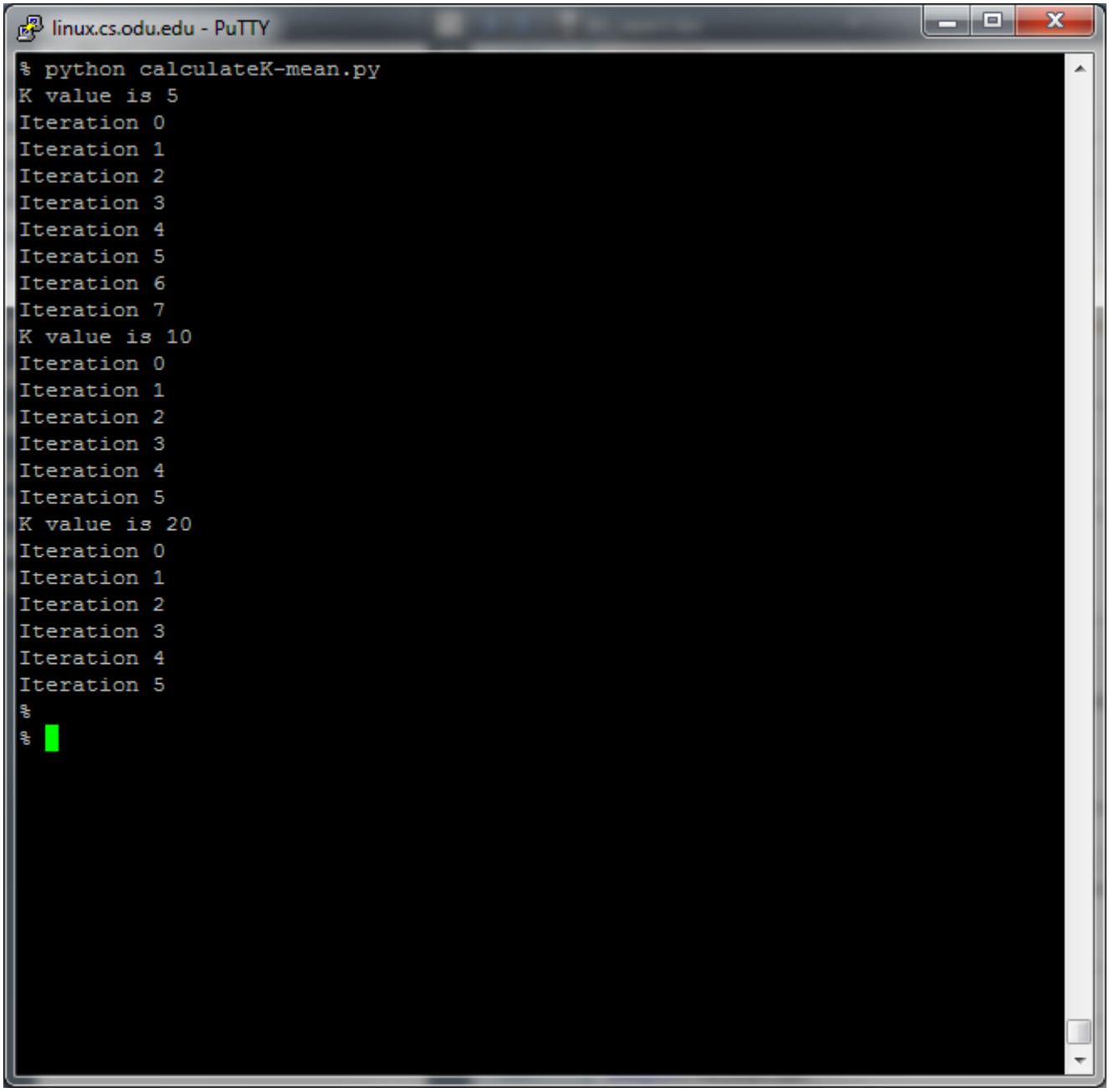


Figure 2: Output of iterations for different k values

4 Question 4:

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

4.1 Approach

1. To solve this questions I used the few lines of code provided in the presentation (createMDS.py).

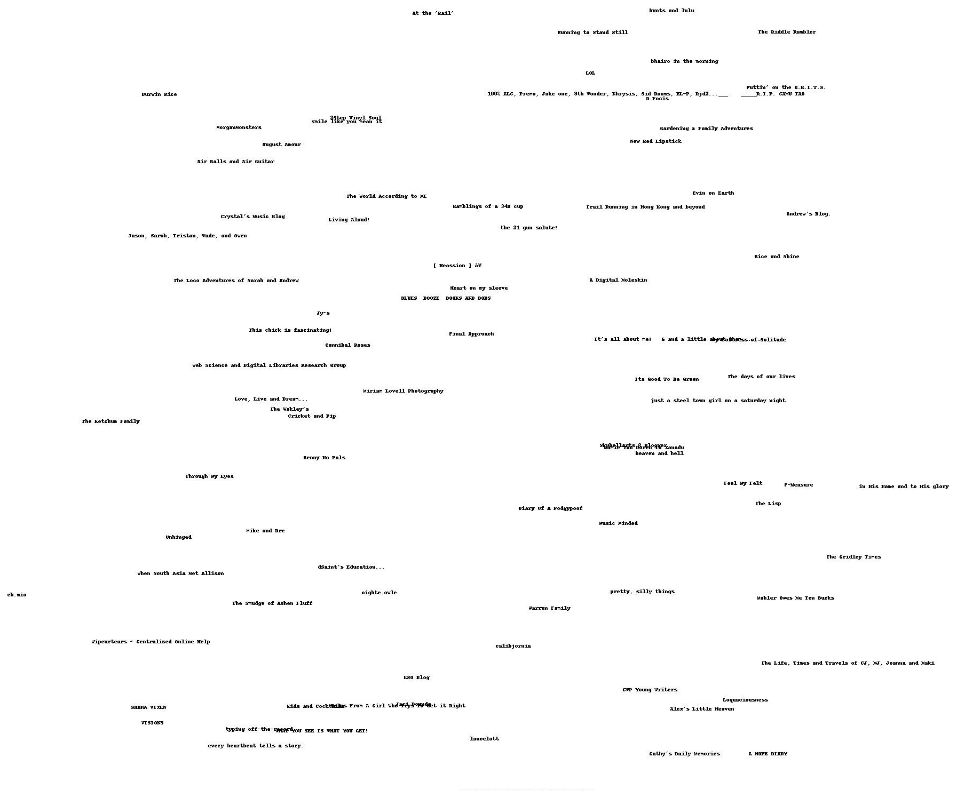
4.2 Source Code

4.2.1 createMDS.py

```
#!/usr/bin/env python

import clusters

def main():

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


if __name__=="__main__":
    try:
        main()
    except KeyboardInterrupt:
        sys.exit(1)
```

4.3 Output Files

```
% python createMDS.py
4515.42807078
5524.56099361
% python createMDS.py
4299.13362421
5370.72157029
% python createMDS.py
4868.240652
7215.65051908
```


At the 'Rail'
hunts and lulu
Running to Stand Still
The Riddle Rambler
bhairo in the morning
LOL
Durwin Rice
100% ALC, Premo, Jake one, 9th Wonder, Khrysis, Sid Roams, EL-P, Rjd2...___
Puttin' on the G.R.I.T.S.
_____R.I.P. CAMU TAO
D.Focis
2Step Vinyl Soul
smile like you mean it
MorganMonsters
Gardening & Family Adventures
August Amour
New Red Lipstick
Air Balls and Air Guitar
The World According to ME
Evin on Earth
Ramblings of a 34B cup
Trail Running in Hong Kong and beyond
Crystal's Music Blog
Living Aloud!
Andrew's Blog.
the 21 gun salute!
Jason, Sarah, Tristan, Wade, and Owen
Rice and Shine
[Meassion] â¥
The Loco Adventures of Sarah and Andrew
A Digital Moleskin
Heart on my sleeve
BLUES BOOZE BOOKS AND BOBS
Jy~s
This chick is fascinating!
Final Approach
It's all about me!
Cannibal Roses
Web Science and Digital Libraries Research Group
Its Good To Be Green
The days of our lives
Miriam Lovell Photography
Love, Live and Dream...
just a steel town girl on a saturday night
The Wakley's
Cricket and Pip
The Ketchum Family
heaven and hell
Benny No Pals
Through My Eyes
Feel My Felt
F-Measure
in His Name and to His glory
The Lisp
Diary Of A Podgypoof
Music Minded
Mike and Dre
Unhinged
The Gridley Times
dSaint's Education...
When South Asia Met Allison
eh.mio
nighte.owle
pretty, silly things
The Smudge of Ashen Fluff
Mahler Owes Me Ten Bucks
Warren Family
Wipeurtears ~ Centralized Online Help
calibjornia
The Life, Times and Travels of CJ, MJ, Joanna and Maki
ESO Blog
CWP Young Writers
Loquaciousness
SHONA VIXEN
Alex's Little Heaven
VISIONS
lancelott
every heartbeat tells a story.
Cathy's Daily Memories
A HOPE DIARY
Southern Hospitality in the Last Frontier

References

[1] feedparser.py program. http://www.cs.odu.edu/ mln/teaching/cs595-f13/chapter3/code/generatefeedvector.python.

[2] Python for beginners. http://www.pythonforbeginners.com/feedparser/using-feedparser-in-python, October 2009.

[3] eumiro. Sorted function in python. http://stackoverflow.com/questions/14218933/sorted-function-in-python, January 2013.

[4] Toby Segaran. *Programming Collective Intelligence*. O'Reilly Media, August 2007.

[]