INTRO. TO WEB SCIENCE: CS 532: A8

Due on Thursday, April 13, 2017

Dr. Nelson

Udochukwu Nweke

Contents

Problem 1

Listing 1: Grab 100 Unique Blog Code

```
#http://blogtimenow.com/blogging/find-blogger-blog-id-post-id-unique-id-number/
import os, sys
import requests
import time

from mod_generatefeedvector import generateFeedVector

def errorMsg():
    exc_type, exc_obj, exc_tb = sys.exc_info()
    fname = os.path.split(exc_tb.tb_frame.f_code.co_filename)[1]
    print(fname, exc_tb.tb_lineno, sys.exc_info() )

def getTerminalBlog(blogURL):

    try:
        r = requests.head(blogURL, allow_redirects=True)
        return r.url.replace('?expref=next-blog', '')
    except:
        errorMsg()

    return ''


def getBlogs(countOfBlogs):

    if( countOfBlogs < 1 ):
        print('Invalid input for count of blogs')
        return

    try:
        output = open('unique100Blogs.txt', 'w')
        output.write('http://f-measure.blogspot.com/\n')
        output.write('http://ws-dl.blogspot.com/\n')
    except:
        errorMsg()

    blogDict = {}
    while len(blogDict) < countOfBlogs:

        blogUrl = 'http://www.blogger.com/next-blog?navBar=true&blogID
        =5885297259923277298'
        terminalBlogURL = getTerminalBlog(blogUrl)

        if( len(terminalBlogURL) != 0 ):
            print('count:', len(blogDict) )
            print('blog:', terminalBlogURL)
            blogDict[terminalBlogURL] = False

        print('\tsleeping')
        print()
```

Problem 1 continued on next page. . .

```
          time.sleep(3)


     for url in blogDict:
          output.write(url + '\n')

     output.close()

#1a
#getBlogs(120)

#1b
#create blog matrix, e.g: https://github.com/ahangchen/PCInotes/blob/master/
#chapter3/blogdata.txt
generateFeedVector(20)#generate blogMatrix.txt
```

Listing 2: Generate Feed Vector Code

```
import os, sys
import feedparser
import re
import requests
from bs4 import BeautifulSoup


def errorMsg():
     exc_type, exc_obj, exc_tb = sys.exc_info()
     fname = os.path.split(exc_tb.tb_frame.f_code.co_filename)[1]
     print(fname, exc_tb.tb_lineno, sys.exc_info() )

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 mergeDicts(dictA, dictB):

     for term, TF in dictB.items():

          if( term in dictA ):
               dictA[term] = dictA[term] + TF
          else:
               dictA[term] = TF

     return dictA


def getPagesForBlog_main(blogUrl, pages=[]):
    try:
        html = requests.get(blogUrl)
        soup = BeautifulSoup(html.text, 'html.parser')
        nextLink = soup.find('link', { 'rel' : 'next' })

        if nextLink is not None:
            nextLink = nextLink['href']
            pages.append(nextLink)
            getPagesForBlog_main(nextLink, pages)

    except:
        errorMsg()

    return pages

def getPagesForBlog_pre(blogUrl):

    blogUrl = blogUrl.strip()
    pages = []

    if( blogUrl[-1:] != '/' ):
        blogUrl = blogUrl + '/feeds/posts/default?max-results=500'
    else:
        blogUrl = blogUrl + 'feeds/posts/default?max-results=500'

    pages = getPagesForBlog_main(blogUrl, pages)

    return pages

def getwordcounts(url):
    '''
    Returns title and dictionary of word counts for an RSS feed
    '''
    # Parse the feed


    #url: http://blogName.blogspot.com/
    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)

#modified to look at all pages of the blog
def generateFeedVector(blogCount=10):

    if( blogCount < 1 ):
        return

    apcount = {}
    wordcounts = {}

    infile = open('./unique100Blogs.txt', 'r')
    feedlist = infile.readlines()
    infile.close()

    counter = 1
    for feedurl in feedlist:

        print('counter: ', counter)

        feedurl = feedurl.strip()

        try:
            #before: (title, wc) = getwordcounts(feedurl + 'feeds/posts/default/')
            #after:
            (title, wc) = getwordcounts(feedurl +
            'feeds/posts/default?max-results=500')

            #get wc for other pages - start
            otherPages = getPagesForBlog_pre(feedurl)
            for page in otherPages:
                page = page.strip()
                (sameTitle, nextPageWordCount) = getwordcounts(page)
                mergeDicts(wc, nextPageWordCount)
            #get wc for other pages - end

            #wc is union
            wordcounts[title] = wc
            for (word, count) in wc.items():
                apcount.setdefault(word, 0)
                if count > 1:
                    apcount[word] += 1
        except:
            print('Failed parsing for feed %s' % feedurl)
            errorMsg()

        if( blogCount == counter ):
            break

        counter += 1


    wordlist = []
    TermTermFrequencyTuplesList = []
    for (term, termFrequency) in apcount.items():

         frac = float(termFrequency) / len(feedlist)

         if frac > 0.1 and frac < 0.5:
              termTermFrequencyTuple = (term, termFrequency)
              TermTermFrequencyTuplesList.append(termTermFrequencyTuple)


    #Limit the number of terms to the most "popular" (i.e., frequent) 1000 terms
    TermTermFrequencyTuplesList = sorted(TermTermFrequencyTuplesList, key=lambda tup:
   tup[1], reverse=True)
    for termFrequencyTuple in  TermTermFrequencyTuplesList:

         #get 1000 most popular terms
         if( len(wordlist) <= 1000 ):
              wordlist.append( termFrequencyTuple[0] )
         else:
              break


    out = open('blogMatrix.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
         out.write(blog)
         for word in wordlist:

              if word in wc:
                   out.write('\t%d' % wc[word])
              else:
                   out.write('\t0')
         out.write('\n')

    out.close()
```

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.
Solution 1:

1. In order to grab 98 more unique blogs, I continuously dereferenced `http://blogtimenow.com/blogging/find-blogger-blog-id-post-id-unique-id-number/` to grab a new blog each time. I used `http://blogtimenow.com/blogging/find-blogger-blog-id-post-id-unique-id-number/` to discover how to find the blog IDs. This is seen in *getBlogs()* in Listing 1. The 100 unique blog url is displayed in Table 1

2. I used *generateFeedVector()* in (PCI code,listing 2). I extracted all the pages for each blog through *getPagesForBlog_pre()* (Listing 2) I also included line 148-163 in listing 2 in order to limit the number of terms to the most “popular” 1000 terms.

Problem 2

Listing 3: Code To Create Dendogram

```
import clusters#from:https://raw.githubusercontent.com/nico/
#collectiveintelligence-book/master/clusters.py
import drawclust

def generateASCIIDendogram(filename):
     blognames,words,data=clusters.readfile(filename)
     clust=clusters.hcluster(data)
     clusters.printclust(clust,labels=blognames)

def generateJPegDendogram(filename):
     blognames,words,data=clusters.readfile(filename)
     clust=clusters.hcluster(data)
     drawclust.drawdendogram(clust,blognames,filename + '.jpg')



filename = './blogMatrix.txt'
#P2.a
#generateASCIIDendogram(filename)

#P2.a
#generateJPegDendogram(filename)
```

Listing 4: ASCII Dendogram Snippet

```
 -
  -
   If You Give a Girl a Camera...
   Riley Haas' blog
  -
   -
    -
```

```
     -
      Avidd Wallows' Blog
      -
       Myopiamuse
       -
        Punk Rock Teaching
        -
         juanbook
         -
          Pithy Title Here
          -
           Morgan's Blog
           -
            Cherry Area
            -
             -
              Kid F
              -
               STANLEY SAYS
               -
                nonsense a la mode
                -
                 -
                  A Day in the Life of...Me!!
                  Room 19's Blog 2016
                 -
                  What A Wonderful World
                  The Stearns Family
             -
              She May Be Naked
              -
               bittersweet
               The Perfect Vent
     -
      Sonology
      -
```

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 2:

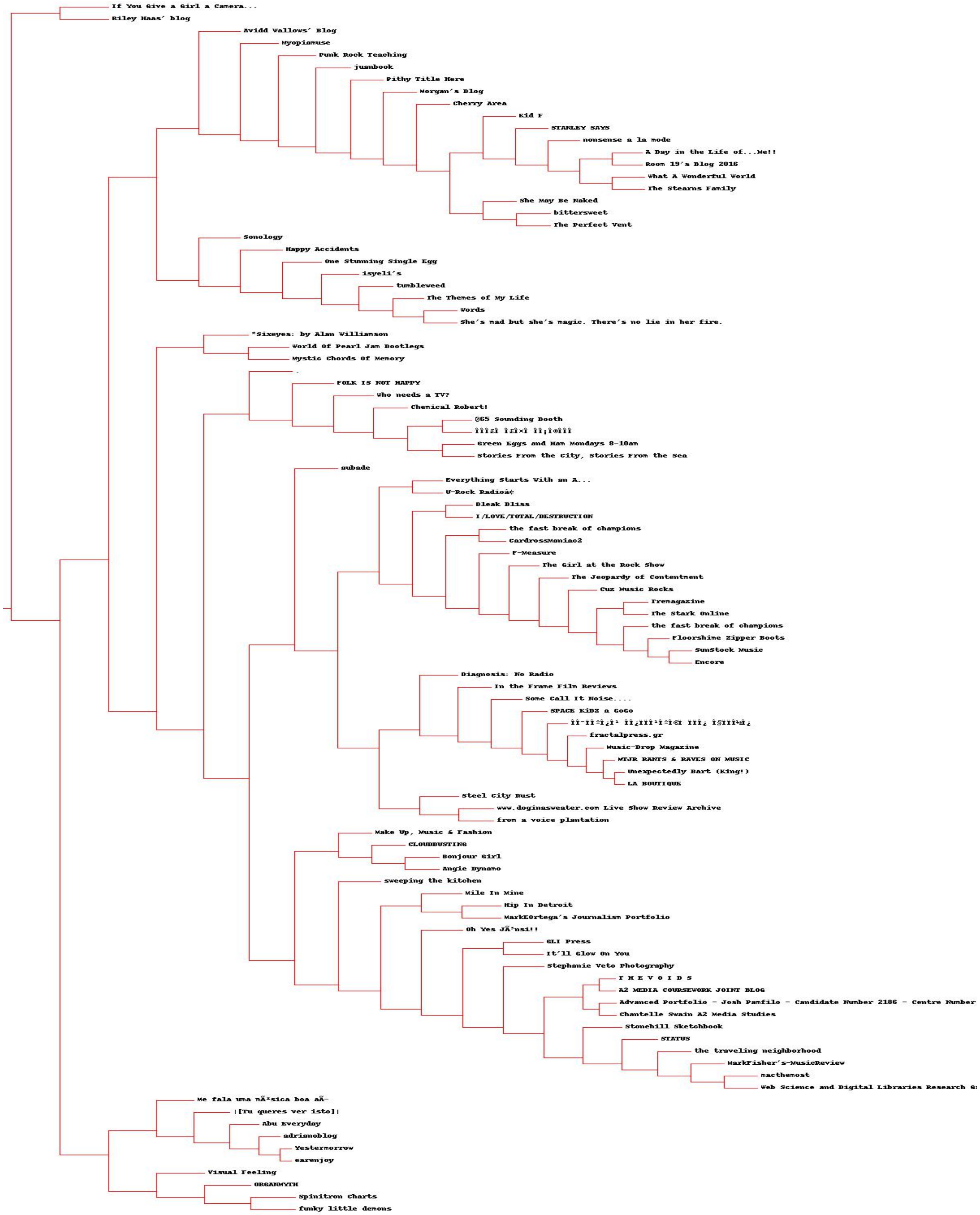
I used *generateASCIIDendogram()* and *generateJPegDendogram()* in listing 3 to create an ASCII and JPEG dendograms respectively. Listing 4 contains a snippet of the ASCII dedogram. *generateASCIIDendogram()* utilizes PCI *printclust()* to draw an ASCII dendogram. *generateJPegDendogram()* utilizes the PCI code *drawdendogram()* to draw a JPEG dendogram. Blogs JPEG dendogram is seen in Figure 1.

Problem 3

Listing 5: K-Means Cluster Solution

Problem 3 continued on next page. . .

Figure 1: Blogs Dendogram



```
import clusters

def KMeans(k):

     if( k<1 ):
          return

     blognames,words,data=clusters.readfile(’blogMatrix.txt’)
     kclust=clusters.kcluster(data,k=k)
     clusterCount = 0

     print ’’
     for cluster in kclust:
          if( len(cluster) > 0 ):
               print ’cluster’, clusterCount
               for item in cluster:
                    print ’\t’,blognames[item]
               clusterCount += 1

k=20
KMeans(k)
```

Listing 6: K-Means Cluster

```
from math import sqrt
import random



def readfile(filename):
  return do_readfile(open(filename).readlines())


def do_readfile(lines):
  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


def pearson(v1, v2):
  """Returns the similarity between v1 and v2.

  1.0 means very similar and 0.0 means no correlation. -1.0 means
  anticorrelation.  v1 and v2 must have the same number of elements."""

  assert len(v1) == len(v2)

  n = len(v1)
  if n == 0: return 0
  sum1 = sum(v1)
  sum2 = sum(v2)

  sqSum1 = sum([pow(v, 2) for v in v1])
  sqSum2 = sum([pow(v, 2) for v in v2])

  pSum = sum([v1[i] * v2[i] for i in range(n)])

  num = pSum - (sum1*sum2/n)
  den = sqrt((sqSum1 - pow(sum1, 2)/n) * (sqSum2 - pow(sum2, 2)/n))
  if den == 0:
    # It's not clear what to do here. It can happen when all components are
    # equal (which means "very similar"), or if one of the vectors contains
    # only zeroes, or if the two vectors contain only one element. In these
    # cases, this function can't figure out how to "scale" its result. Cop
    # out and simply return 0 for those cases.
    return 0

  return num/den


def pearson_dist(v1, v2):
  """0.0 means "near", 1.0 means "far"."""
  return 1 - pearson(v1, v2)


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

  def __eq__(self, b):
    return (self.vec == b.vec
        and self.left == b.left
        and self.right == b.right
        and self.distance == b.distance
        and self.id == b.id)

  # If we have __eq__, we better have __ne__ too
  # so that `not (a == b) == a != b'
  def __ne__(self, b):
    return not (self == b)

  # If we have __eq__, we better have __hash__ too
  # so that `a == b => hash(a) == has(b)'. Since we don't need bicluster objects
  # as dict keys, it's ok if this function fails loudly (instead of silently
  # returning a wrong value, which is the defaul)
  def __hash__(self):
    raise NotImplementedError
  def __str__(self):
    return '%s %f %d (%s %s)' % (str(self.vec), self.distance, self.id,
        self.left, self.right)


def mergevecs(a, b):
  return [(a[i] + b[i])/2.0 for i in range(len(a))]


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

  # Clusters start off as just rows
  clust = [bicluster(rows[i], id=i) for i in range(len(rows))]

  # O(n^3), yuck! Effectively, only the distance() calls are expensive,
  # and we cache them, so this is really O(n^2)
  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)):
        # cache distances. Makes this much faster.
        # (can't use the cache() function because we cache on ids, not
        # function arguments. as clust shrinks, we can't just cache on indices
        # either)
        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

    # Merge closest pair into a single vector
    mergevec = mergevecs(clust[lowestpair[0]].vec, clust[lowestpair[1]].vec)

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

    # Update
    currentclustid -= 1
    del clust[lowestpair[1]]  # Need to del() bigger index first!
    del clust[lowestpair[0]]
    clust.append(newcluster)

  return clust[0]
```

```
def printclust(clust, labels=None, n=0):
  print ’ ’ * n,
  if clust.id < 0:  # branch
    print ’-’
  else:
    print labels[clust.id] if labels else clust.id

  if clust.left: printclust(clust.left, labels=labels, n=n+1)
  if clust.right: printclust(clust.right, labels=labels, n=n+1)


def transpose(data):
  return map(list, zip(*data))


def rowbb(rows):
  """Returns the bounding box of the row vectors of the matrix ‘rows‘
  as list of min/max pairs for each dimension."""
  return zip(map(min, transpose(rows)), map(max, transpose(rows)))


def getnearest(v, points, distance):
  """Returns the index of the point in ‘points‘ closest to ‘v‘."""
  bestmatch = 0
  for i in range(len(points)):
    d = distance(points[i], v)
    if d < distance(points[bestmatch], v): bestmatch = i
  return bestmatch


def average(indices, rows):
  """Returns the average of all rows indexed by ‘indices‘. All rows have to
  have the same number of elements."""
  avg = [0.0] * len(rows[0])
  if len(indices) > 0:
    for rowid in indices:
      for m in range(len(rows[0])):
        avg[m] += rows[rowid][m]
    for j in range(len(avg)):
      avg[j] /= len(indices)
  return avg


def kcluster(rows, distance=pearson_dist, k=4):
  """Returns a list of ‘k‘ lists, each containing all indices of a cluster."""

  ranges = rowbb(rows)
  clusters = [[random.uniform(r[0], r[1]) for r in ranges] for j in range(k)]

  lastmatches = None
  for t in range(100):
    print ’Iteration’, t
    bestmatches = [[] for i in range(k)]

    # find best centroid for each row
    for j in range(len(rows)):
      bestmatches[getnearest(rows[j], clusters, distance)].append(j)

    # if the results didn’t change in this iteration, we are done
    if bestmatches == lastmatches: break
    lastmatches = bestmatches

    # move centroids to the averages of their elements
    for i in range(k):
      clusters[i] = average(bestmatches[i], rows)

  return bestmatches


def tanimoto_dist(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 hypot(v):
  return sqrt(sum([x*x for x in v]))


def euclid_dist(v1, v2):
  return hypot([v[0] - v[1] for v in zip(v1, v2)])


def scaledown(data, distance=pearson_dist, rate=0.01):
  iterCount = 1
  n = len(data)

  realdist = [[distance(data[i], data[j]) for j in range(n)] for i in range(n)]
  outersum = 0.0

  # random start positions
  loc = [[random.random(), random.random()] for i in range(n)]

  lasterror = None
  for m in range(0, 1000):
    # find projected distance
    fakedist = [[euclid_dist(loc[i], loc[j])
      for j in range(n)] for i in range(n)]

    # 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

        # error is percent difference between distances
        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
    iterCount += 1


    # if we got worse by moving the points, quit
    if lasterror and lasterror < totalerror: break

    # also break if the improvement is only very small
    if lasterror and lasterror - totalerror < 1e-15: break

    lasterror = totalerror

    # move points by learning rate times gradient
    if k in range(n):
      loc[k][0] -= rate * grad[k][0]
      loc[k][1] -= rate * grad[k][1]

  return loc, iterCount


if __name__ == '__main__':
  # stupid demo
  import drawclust
  blognames, words, data = readfile('blogdata.txt')
  c = hcluster(data)
  #printclust(c, labels=blognames)
  drawclust.drawdendogram(c, blognames, 'dendo.png')
  print 'Wrote dendo.png'

  ## this is _much_ slower, as hcluster computes O(rows^2) many distances,
  ## and there are many more words than blognames in out data.
  #c = hcluster(transpose(data))
  #drawclust.drawdendogram(c, words, 'dendo_words.png')
  #print 'Wrote dendo_words.png'

  kclust = kcluster(data, k=10)
  for i in range(len(kclust)):
    print 'k-cluster %d:' % i, [blognames[r] for r in kclust[i]]
    print

  # another demo
  coords = scaledown(data)
  drawclust.draw2d(coords, blognames, filename='blogs2d.png')
  print 'Wrote blogs2d.png'

  # and yet another demo
  wants, people, data = readfile('official_zebo.txt')
  cl = hcluster(data, distance=tanimoto_dist)
  drawclust.drawdendogram(cl, wants, 'wants.png')
  print 'Wrote wants.png'
```

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 3:

Clustering the blogs using K-Means is achieved by using *KMeans()* in listing 5. I modified *scaledown()* in listing 6 in order to return the number of iterations required for each value of k. Table 2 shows the k-values and number of iterations. K5.txt, K10.txt, and K20.txt files contains K clustering counts and iterations respectively.

Problem 4

Listing 7: MDS Code language

```
import clusters
import drawclust

def MDS():
     iterations = 0
     blognames,words,data=clusters.readfile('blogMatrix.txt')
     coords, iterations=clusters.scaledown(data)
     drawclust.draw2d(coords,blognames,'blogMatrix.mds.jpg')

     print 'iterations', iterations

MDS()
```

Use MDS to create a JPEG of the blogs similar to slide 29 of the week 12 lecture. How many iterations were required?

Solution 4:

In order to use MDS to create a JPEG of similar blogs, I used *MDS()* in listing 6. The MDS JPEG of blogs is seen in Figure 2.

Figure 2: MDS Diagram, Required iteration = 5

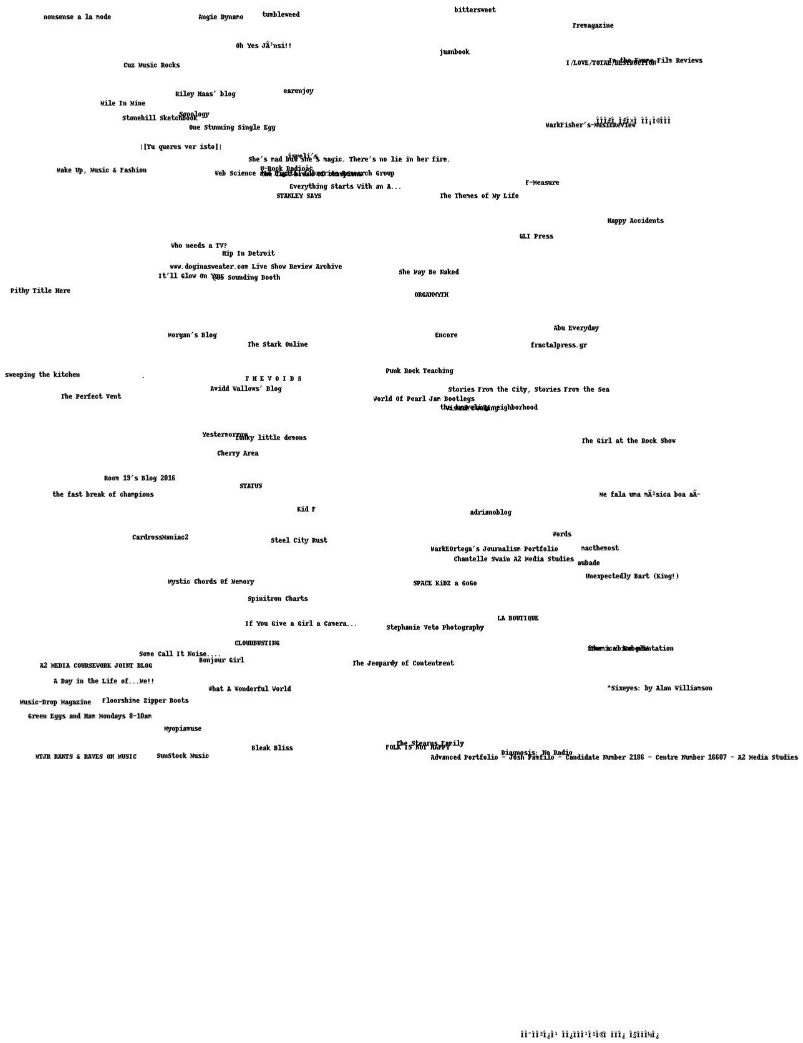


Table 1: 100 Unique Blogs

S/N	URL
1	`http://f-measure.blogspot.com/`
2	`http://ws-dl.blogspot.com/`
3	`http://antonellagiugliano.blogspot.com/`
4	`http://markeortega.blogspot.com/`
5	`http://myopiamuse.blogspot.com/`
6	`http://floorshimezipperboots.blogspot.com/`
7	`http://ps-music.blogspot.com/`
8	`https://chemical-robert.blogspot.com/`
9	`http://onestunningsingleegg.blogspot.com/`
10	`http://www.thestarkonline.com/`
11	`http://macthemost.blogspot.com/`
12	`http://psychfolkmusic.blogspot.com/`
13	`http://adrianomarquesblog.blogspot.com/`
14	`http://www.punkrockteaching.org/`
15	`http://cherryarea.blogspot.com/`
16	`http://doyouneedatv.blogspot.com/`
17	`http://ohyesjonsi.blogspot.com/`
18	`http://mts-dailythemes.blogspot.com/`
19	`http://www.thejeopardyofcontentment.com/`
20	`http://www.sonology.com/`
21	`http://jamiemclelland.blogspot.com/`
22	`http://organmyth.blogspot.com/`
23	`http://jbreitling.blogspot.com/`
24	`http://nonsensealamode.blogspot.com/`
25	`http://mondaywakeup.blogspot.com/`
26	`http://chantellesmedia2.blogspot.com/`
27	`http://doginasweatershowreviews.blogspot.com/`
28	`http://jojobethkatiehannahlcm1516.blogspot.com/`
29	`http://angie-dynamo.blogspot.com/`
30	`http://mandolinnn.blogspot.com/`
31	`http://ngaio1619.blogspot.com/`
32	`http://isyelili.blogspot.com/`
33	`http://earenjoy.blogspot.com/`
34	`http://tuqueresveristo.blogspot.com/`
35	`http://mediastudiesa2advanced.blogspot.com/`
36	`https://norecordshopsleft.blogspot.com/`
37	`http://bonjourgirl.blogspot.com/`
38	`http://intheframefilmreviews.blogspot.com/`
39	`http://sixeyes.blogspot.com/`
40	`http://johnandmaureensanto.blogs`
41	`http://thefastbreakofchampions.blogspot.com/`
42	`http://paradoxical-era.blogspot.com/`
43	`http://markfishers-musicreview.blogspot.com/`
44	`http://smalltumbleweed.blogspot.com/`

S/N	URL
45	http://ilovetotaldestruction.blogspot.com/
46	http://somecallitnoise.blogspot.com/
47	http://steel-city-rust.blogspot.com/
48	http://fractalpress.blogspot.com/
49	http://noradiorecs.blogspot.com/
50	http://bartkings.blogspot.com/
51	http://ablazingflame.blogspot.com/
52	http://stanleysaystanley.blogspot.com/
53	http://sixtyat60.blogspot.com/
54	http://mtjrrantsravesonmusic.blogspot.com/
55	http://encorenorthernireland.blogspot.com/
56	http://londynsky.blogspot.com/
57	http://mcomv2.blogspot.com/
58	http://skinnyshoes.blogspot.com/
59	http://worldofpearljambootlegs.blogspot.com/
60	http://mileinmine.blogspot.com/
61	http://globalgoon.blogspot.com/
62	http://glipress.blogspot.com/
63	http://avidsblog.blogspot.com/
64	http://stephanieveto.blogspot.com/
65	http://storiesfromthecityradiovalencia.blogspot.com/
66	http://www.juanbook.com/
67	http://thetremagazine.blogspot.com/
68	http://momslilprincess.blogspot.com/
69	http://travelingneighborhood.blogspot.com/
70	http://spicyseatdolphin.blogspot.com/
71	http://abueveryday.blogspot.com/
72	http://justwordsnomeaning.blogspot.com/
73	http://out-of-the-swamp.blogspot.com/
74	http://aubade1.blogspot.com/
75	http://www.chrisanne-grise.com/
76	http://musicneedshelp.blogspot.com/
77	http://laboutiquemusic.blogspot.com/
78	http://stonehillsketchbook.blogspot.com/
79	http://pithytitlehere.blogspot.com/
80	http://marshwiggle.blogspot.com/
81	http://www.hipindetroit.com/
82	http://www.sunstockmusic.com/
83	http://dana9morgan.blogspot.com/
84	http://blog.spinitron.com/
85	http://itll-glow-on-you.blogspot.com/
86	http://jlmdlhlcm1516.blogspot.com/
87	http://klavierspielerman.blogspot.com/
88	http://dinosaursarefun.blogspot.com/
89	http://cloudbusting87.blogspot.com/

S/N	URL
90	`http://bleakbliss.blogspot.com/`
91	`http://mefalaumamusicaboaai.blogspot.com/`
92	`http://theonionfield.blogspot.com/`
93	`http://cardrossmaniac2.blogspot.com/`
94	`http://dcresider.blogspot.com/`
95	`http://ourstatus.blogspot.com/`
96	`http://hani-bittersweet.blogspot.com/`
97	`http://cuzmusicrocks.blogspot.com/`
98	`https://urockradio.blogspot.com/`
99	`http://makeupmusicandfashion.blogspot.com/`
100	`http://superchicken46.blogspot.com/`

Table 2: K-Values/Iterations

Item	K-value	Iterations
1	5	6
2	10	7
3	20	4

References

[1] Blog Time Now. http://blogtimenow.com/blogging/find-blogger-blog-id-post-id-unique-id-number/. Accessed: 2017-10-04.

[2] Toby Segaran. Programming Collective Intelligence, 2007.