Ryan Condotta

Final Report Assignment 8

Question 1:

In order to create a blog-term matrix, I first needed to create a python script to extract 98 unique links since 2 links were already provided for us. So, I created the python script A8Q1.py which constantly get a new blog url and outputs the blog name to Blogs.txt in Figure 1. Then, I store the html in a variable and find the RSS link in the html and output that RSS link into the file RSS.txt in Figure 2. The process is shown in Figure 3 of the extraction of the urls to get the RSS url.

http://pithytitlehere.blogspot.com/
http://ihatethe90s.blogspot.com/
http://mondaywakeup.blogspot.com/
http://karldrinkwater.blogspot.com/

Figure 1

http://pithytitlehere.blogspot.com/feeds/posts/default?alt=rss
http://ihatethe90s.blogspot.com/feeds/posts/default?alt=rss
http://mondaywakeup.blogspot.com/feeds/posts/default?alt=rss
http://karldrinkwater.blogspot.com/feeds/posts/default?alt=rss
http://markeortega.blogspot.com/feeds/posts/default?alt=rss
http://ps-music.blogspot.com/feeds/posts/default?alt=rss

Figure 2

```
count = 1
link = r"http://www.blogger.com/next-blog?navBar=true&blogID=3471633091411211117"
while count < 100:
	print(count)
	response = urllib.request.urlopen(link)
	print(response.geturl())
	soup = BeautifulSoup(response, 'html.parser')
	#print(tag)
	filename1 = r"C:\Users\Ryan\Documents\WebScience\Assignment8\Blogs.txt"
	outfile = open(filename1, 'a')
	temp = response.geturl()
	temp1 = temp[0:len(temp)-17]
	outfile.write(str(temp1))
	outfile.write("\n")
	outfile.close()

	for tag in soup.findAll('link'):
		filename2 = r"C:\Users\Ryan\Documents\WebScience\Assignment8\Response.txt"
		outfile = open(filename2, 'w')
		outfile.write(str(tag.encode('utf-8')))
		outfile.write("\n")
		outfile.close()

		for line in open(filename2):
```

```
                    if "application/rss+xml" in line:
                        filename3 =
r"C:\Users\Ryan\Documents\WebScience\Assignment8\RSS.txt"
                        outfile = open(filename3, 'a')
                        outfile.write(str(temp1)+"feeds/posts/default?alt=rss")
                        outfile.write("\n")
                        outfile.close()

    count+= 1
```

Figure 3

Next, since we were able to get the RSS urls, we have the information to make the blog-term matrix which is written in the python script A8Q1.1.py. To simplify this process, the Programming Collective Intelligence had several functions that I used and updated to python 3.3 to accomplish the task. The first two function I used was getwordcounts() and getwords, which passes this summary to getwords, which strips out all of the HTML and splits the words by nonalphabetical characters, returning them as a list as shown Figure 4.

```
# 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!='']
```

Figure 4

The code that loops over every line RSS.txt and generates the word counts for each blog, as well as the number of blogs each word appeared in (apcount) as shown in Figure 5. After, we need to generate the list of words that will actually be used in the counts for each blog which is limited to 500 words for the assignment. We did not want every word so we limited the range to get rid of the common words as we as the rare words which describes the range of 10 percent to 50 percent which is shown in the process of Figure 6.

```
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)
```

Figure 5

```
#get 500 words in the range
for w,bc in apcount.items():
  frac=float(bc)/len(feedlist)
  if len(wordlist) <= 500:
    if frac>0.1 and frac<0.5:
      wordlist.append(w)
print(len(wordlist))
```

Figure 6

The final step was to output the blog-term matrix to a file which I named blogdata.txt. I would output the words used from each blog as the top row and the first column represents the blog title. The file creation was successful, however, there were some abnormalities in the printing of the blog names where the encoding would not transition for some of the titles which was peculiar as I could not find a solution for that, and the tabs are not perfectly lined up with the numbers and the words that they were correlated since length was an issue and messes up the tabs.

Question 2:

In order to accomplish the task of creating a dendogram in ASCII and JPEG format from the blog-term matrix, I created the python script of A8Q2.py. In that script, I use again the functions described in Programming Collective Intelligence to help me create these dendograms. The first function I used was readfile which basically stores the words from top row, stores the blog names from the first column, and the data from blogdata.txt as shown in Figure 7.

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

```
    lines.append(line)

  # 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
```

Figure 7

Since, we were able to read the blog-term matrix and store the values into variables in the python script, we then created the functions pearson and hcluster to help create these hierarchical relationships. Pearson works by comparing relationships between blogs based of the number of words as shown in Figure 8, since some blogs contain more words than others. Hcluster, as shown in Figure 9, works by creating a group of clusters that are just the original items. The main loop searchs for the best correlation between a pair which is then merged into a single cluster. The new cluster is the average of the pair of data. This process is repeated until only one cluster remains.

```
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
```

Figure 8

```
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]
```

Figure 9

Lastly, we needed a way to output to dendogram in ASCII and JPEG format. I created the two functions of printclust, Figure 10, for ASCII and drawdendrogram, Figure 11, for JPEG. Printclust traverses the cluster recursively and prints the tree to a txt file. Drawdendogram function also works along the same lines but in a fancier way to make the dendogram more appealing by drawing nodes with height, width, and depths. Figure 12 shows the result of the drawdendogram.

```
def printclust(clust,labels=None,n=0):

  filename2 = r"C:\Users\Ryan\Documents\WebScience\Assignment8\blogAsciiDendogram.txt"
  out=open(filename2,'a')
```

```
  # indent to make a hierarchy layout
  for i in range(n):
    print(' '),
    out.write(' '),
  if clust.id<0:
    # negative id means that this is branch
    print('-')
    out.write('-\t')
  else:
    # positive id means that this is an endpoint
    if labels==None:
      print(clust.id)
      out.write(clust.id)
    else:
      print(labels[clust.id])
      out.write(labels[clust.id])
  out.write('\n')
  # 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)

  out.close()
```

Figure 10

```
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')
```

Figure 11

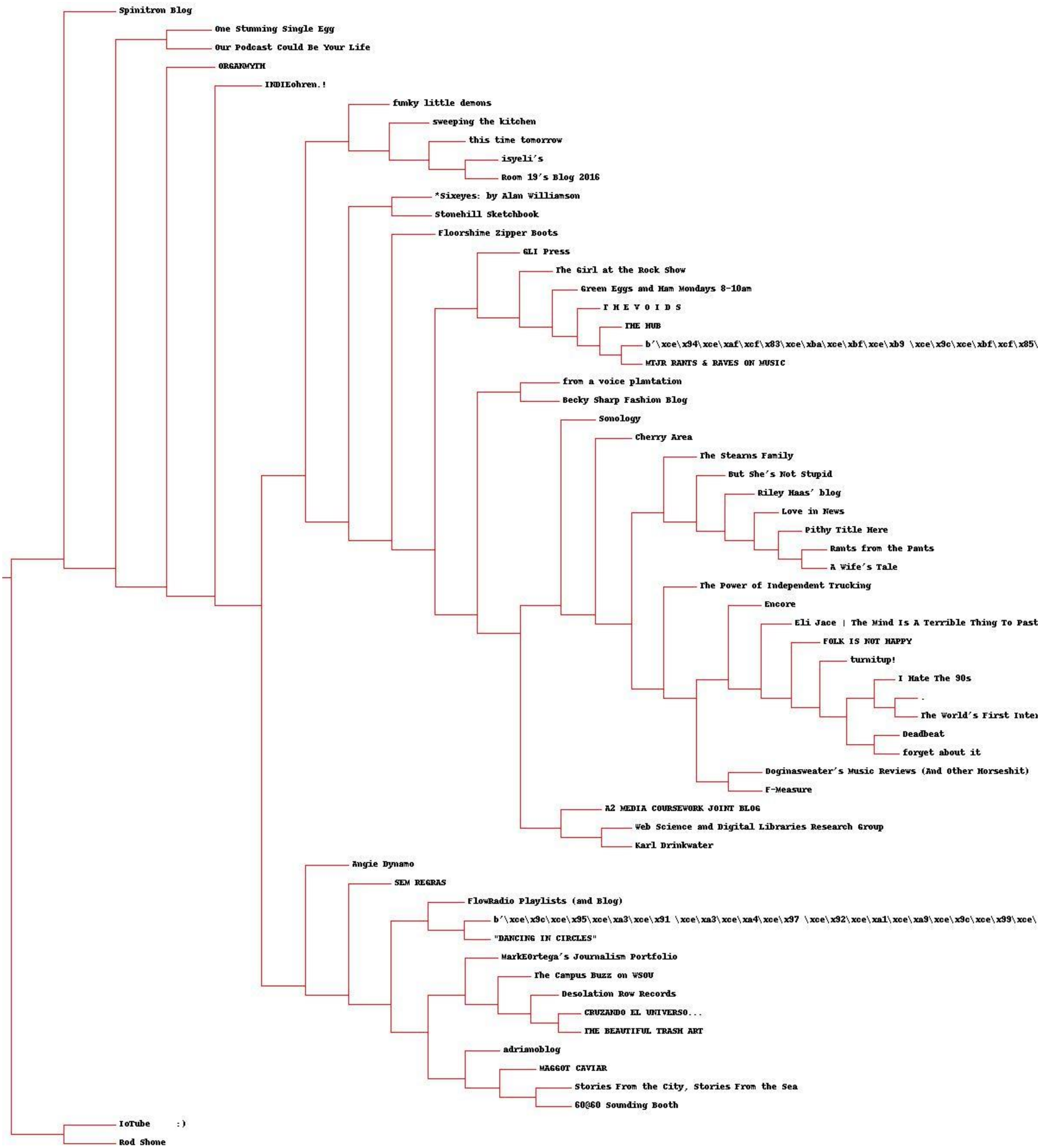
Spinitron Blog
One Stunning Single Egg
Our Podcast Could Be Your Life
ORGANMYTH
INDIEohren.!
funky little demons
sweeping the kitchen
this time tomorrow
isyeli's
Room 19's Blog 2016
*Sixeyes: by Alan Williamson
Stonehill Sketchbook
Floorshime Zipper Boots
GLI Press
The Girl at the Rock Show
Green Eggs and Ham Mondays 8-10am
T H E V O I D S
THE HUB
b'\xce\x94\xce\xaf\xcf\x83\xce\xba\xce\xbf\xce\xb9 \xce\x9c\xce\xbf\xcf\x85\
MTJR RANTS & RAVES ON MUSIC
from a voice plantation
Becky Sharp Fashion Blog
Sonology
Cherry Area
The Stearns Family
But She's Not Stupid
Riley Haas' blog
Love in News
Pithy Title Here
Rants from the Pants
A Wife's Tale
The Power of Independent Trucking
Encore
Eli Jace | The Mind Is A Terrible Thing To Past
FOLK IS NOT HAPPY
turnitup!
I Hate The 90s
.
The World's First Inte
Deadbeat
forget about it
Doginasweater's Music Reviews (And Other Horseshit)
F-Measure
A2 MEDIA COURSEWORK JOINT BLOG
Web Science and Digital Libraries Research Group
Karl Drinkwater
Angie Dynamo
SEM REGRAS
FlowRadio Playlists (and Blog)
b'\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\
"DANCING IN CIRCLES"
MarkEOrtega's Journalism Portfolio
The Campus Buzz on WSOU
Desolation Row Records
CRUZANDO EL UNIVERSO...
THE BEAUTIFUL TRASH ART
adrianoblog
MAGGOT CAVIAR
Stories From the City, Stories From the Sea
60@60 Sounding Booth
IoTube :)
Rod Shone

Figure 12

Question 3:

In order to accomplish the task of clustering the blogs using K-means, I created the file of A8Q3.py. I used the PCI function of kcluster(). This function takes the same data rows as input as does the hierarchical clustering algorithm, along with the number of clusters (k) that the caller would like returned as shown in Figure 13.

```
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, t
```

Figure 13

Since we are able to define the number of clusters, I had to create a way of printing out the centroid values. So, I created a new file which the name would be the value for k and have the number of iterations in it, as well as, blognames for each centroid. This format is shown in Figure 15.

```
filename2 = r"C:\Users\Ryan\Documents\WebScience\Assignment8\k_5.txt"
out=open(filename2,'a')
out.write("Iterations: "+ str(t)+"\n")
k = 0
while k < 5:
  out.write("[")
  for r in kclust[k]:
    out.write(blognames[r]+ ", ")
  out.write("]\n")
  k+=1
out.close()
```

Figure 14

Iterations: 4
[., Deadbeat, Eli Jace | The Mind Is A Terrible Thing To Paste, from a voice plantation, GLI Press, Doginasweater's Music Reviews (And Other Horseshit), MAGGOT CAVIAR, forget about it, The World's First Internet Baby, Encore, Floorshime Zipper Boots, Cherry Area, Rod Shone, Sonology, turnitup!, Riley Haas' blog, I Hate The 90s, Angie Dynamo, FOLK IS NOT HAPPY]

Figure 15- Format k = 5

After performing the cluster for blogs using k =5, 10, and 20, I stored the results in text files of k_5.txt, k_10.txt, and k_20.txt. These files contained the centroids with the blog names inside as well as the number of iterations that happened from the kcluster function. For k =5, the number of iterations were 4. For k=10, the number of iterations were 4, as well. For k = 20, the number of iterations was 5. It seems like the right number for centroids is between 5 and 10 while 20 contains centroids that do not have any blog names in them. This is also true for k = 10, however, the number of centroids with 0 blogs is significantly less for 10 than it is for 20 which makes me think that the ideal value is between 5 and 10.

Question 4:

In order to accomplish the goal of using Use MDS to create a JPEG of the blogs similar to that in the slides, I created the python file A8Q4.py which takes two function of scaledown and draw2d from the PCI textbook. The Scaledown function takes the difference between every pair of items and tries to make a chart in which the distances between the items match those differences. The process of the function is shown in Figure 16. The draw2d function works in a way to generate an image with all the labels of all the different items plotted at the new coordinates of that blog names.

```
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
```

Figure 16

After performing the test, the jpeg created is named blog2d.jpeg and is shown in Figure 17. The number of iterations that were performed were 62. I ran this a couple of times and the final value was about the lowest number of iterations while the other runs were closer to 150 or even 228 in one case which was peculiar to me.


Spinitron Blog
IoTube :)
Stonehill Sketchbook
Room 19's Blog 2016
GLI Press
The Campus Buzz on WSOU
T H E V O I D S
Desolation Row Records
The Stearns Family
"Sixeyes: by Alan Williamson
Karl Drinkwater
Rod Shone
THE HUB
MAGGOT CAVIAR
Sonology
Riley Haas' blog
Stories From the City, Stories From the Sea
Web Science and Digital Libraries Research Group
Green Eggs and Ham Mondays 8-10am
Rants from the Pants
FOLK IS NOT HAPPY
CRUZANDO EL UNIVERSO...
A Wife's Tale
Eli Jace | The Mind Is A Terrible Thing To Paste
NEW REGRAS
Deadbeat
I Hate The 90s
from a voice plantation
The Power of Independent Trucking
FlowRadio Playlists (and Blog)
forget about it
THE BEAUTIFUL TRASH ART
The Girl at the Rock Show
Pithy Title Here
Our Podcast Could Be Your Life
The World's First Internet Baby
But She's Not Stupid
Encore
funky little demons
Floorshine Zipper Boots
isyeli's
Love in News
F-Measure
adrianoblog
turnitup!
60860 Sounding Booth
Cherry Area
"DANCING IN CIRCLES"
A2 MEDIA COURSEWORK JOINT BLOG
Becky Sharp Fashion Blog
this time tomorrow
sweeping the kitchen
INDIEohren.!
One Stunning Single Egg
Angie Dynuro

Figure 17