CS 532: Assignment 8

Dinesh Kumar Paladhi

Spring 2016

Contents

1 Problem 1 . . . 2
1.1 Solution . . . 2
1.2 Code Listing . . . 3
1.3 Code Listing . . . 4
1.4 Code Listing . . . 5
1.5 Outputs . . . 7
2 Problem 2 . . . 11
2.1 Solution . . . 11
2.2 Code Listing . . . 12
2.3 Code Listing . . . 17
2.4 Outputs . . . 18
3 Problem 3 . . . 20
3.1 Solution . . . 20
3.2 Code Listing . . . 21
3.3 Output . . . 22
4 Problem 4 . . . 23
4.1 Solution . . . 23
4.2 Code Listing . . . 24
4.3 Outputs . . . 25
5 Problem 5 . . . 27
5.1 Solution . . . 27
5.2 Code Listing . . . 28
5.3 Outputs . . . 32

1 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.
```

1.1 Solution

1. My task was to create a blog matrix and for that first I need to get all the blog URIs.
2. I appended each URI with "/feeds/posts/default?alt=rss". The code for doing this can be found in listing1.
3. Sample list of blog URIs can be found in fig1 and sample list of URIs after appending can be found in fig2.
4. Now I found the number of pages in each blog using the code in the listing2. Sample list of the number of pages in each blog can be found in fig3.
5. Now In order to find the blog matrix I used "generatefeedvector.py" code from Programming Collective Intelligence text.
6. I used feedparser library in order to parse the URI. I have made modification to the code to limit the words count to 500.
7. Each row in the blog represents a blog with blogname and each column is a specific word. Every cell in the matrix represent the number of times a word in present in that particular blog.
8. Some URIs did not allow the feedparser to parse through them. So I added some other URIs to my list in order to get 100 URIs.
9. I have also used stopwords so that irrelevant data is not entered into the blog matrix.
10. In the blog matrix there are 100 rows(blogs) and 500 columns(words).
11. The frequency of each word in indicated in each cell of the matrix.
12. Python code for generating the blog matrix can be found in the listing3.
13. Sample blog matrix can be seen in the fig??.

1.2 Code Listing

```
import requests


def get100blogurls():
        file_1 = open('100blogurls','w')
        unique = set()
        while (len(unique) < 98) :
                url="http://www.blogger.com/next-blog?navBar=true&blogID=3471633091411211117
                    "
                gen=requests.get(url)
                final_url= gen.url.strip('?expref=next-blog/')
                unique.add(final_url)
                # print final_url
                # print count
                print len(unique)
        for element in unique :
                print element
                file_1.write(element+'\n')
                file_1.flush()
        file_1.write('http://f-measure.blogspot.com'+'\n')
        file_1.write('http://ws-dl.blogspot.com')

def getatomurls():
        input= open('100blogurls','r')
        output= open('100atomurls','w')
        for element in input :
                #print element

                add= "/feeds/posts/default?alt=rss"
                item= element.strip() + add
                output.write(item+'\n')
                # output.write('\n')
                print item

#get100blogurls()
getatomurls()
```

Listing 1: Python code for getting 100 unique blog URIs and atom URIs

1.3 Code Listing

```
#! /usr/bin/env python
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 main():
    feedlist   = open('100atomurls').readlines()

    for url in feedlist:
        try:
            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()



        except KeyError:
            pass

if __name__ == '__main__':
    main()
```

Listing 2: Python code to find the number of pages in each blog

1.4 Code Listing

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

import re
import sys
import feedparser
from bs4 import BeautifulSoup
import urllib


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

# Returns title and dictionary of word counts for an RSS feed
def getwordcounts(url):
        # Parse the feed
        d=feedparser.parse(url)
        wc={}
        stopwords = []
        stopWordList = open('stopWordList.txt').readlines()

        for stopWord in stopWordList:
                stopWord = stopWord.strip()
                stopwords.append(stopWord)

        # 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)
                        if word not in stopwords:
                                wc[word] += 1
        nextLink = checkNextPage( url )
        while nextLink:
            nextLink = checkNextPage( nextLink )
            d        = feedparser.parse(nextlink)
            pages    = len(d['entries'])
            for e in d.entries:
                if 'summary' in e:
                    summary = e.summary
                else:
                    summary = e.description

                words = getwords('%s %s' % (e.title, summary))

                for word in words:
                    if word not in stopwords:
                        #print word
                        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
```

```
71    return [word.lower() for word in words if word!='']
72
73  def main():
74
75    apcount={}
76    wordcounts={}
77    feedlist=[line for line in file('100atomurls')]
78    for feedurl in feedlist:
79      try:
80        title,wc=getwordcounts(feedurl)
81        wordcounts[title]=wc
82        for word,count in wc.items():
83          apcount.setdefault(word,0)
84          if count>1:
85            apcount[word]+=1
86      except:
87        print 'Failed to parse feed %s' % feedurl
88
89    wordlist=[]
90    countFrequentWords = []
91    for w,bc in apcount.items():
92      frac=float(bc)/len(feedlist)
93      if frac>0.1 and frac<0.5:
94        countFrequentWords.append((w,bc))
95
96    countFrequentWords=sorted(countFrequentWords,key=lambda x:x[1], reverse = True)
97
98    for value in countFrequentWords:
99      # word
100     value1 = value[0]
101     #count
102     value2 = value[1]
103     length = len(wordlist)
104     if(length < 500):
105       wordlist.append(value1)
106     else:
107       break
108
109   out=file('blogdata.txt','w')
110   out.write('Blog')
111
112   for word in wordlist:
113     word1 = word.encode('UTF-8')
114     out.write('\t%s' % word1)
115   out.write('\n')
116
117   for blog,wc in wordcounts.items():
118     blogName = blog.encode('UTF-8')
119     print blog
120     out.write(blogName)
121     for word in wordlist:
122       if word in wc: out.write('\t%d' % wc[word])
123       else: out.write('\t0')
124     out.write('\n')
125
126 if __name__ == "__main__":
127   try:
128       main()
129   except KeyboardInterrupt:
130       sys.exit(1)
```

Listing 3: Python code for getting blog matrix

1.5 Outputs

Sample Blog URIs

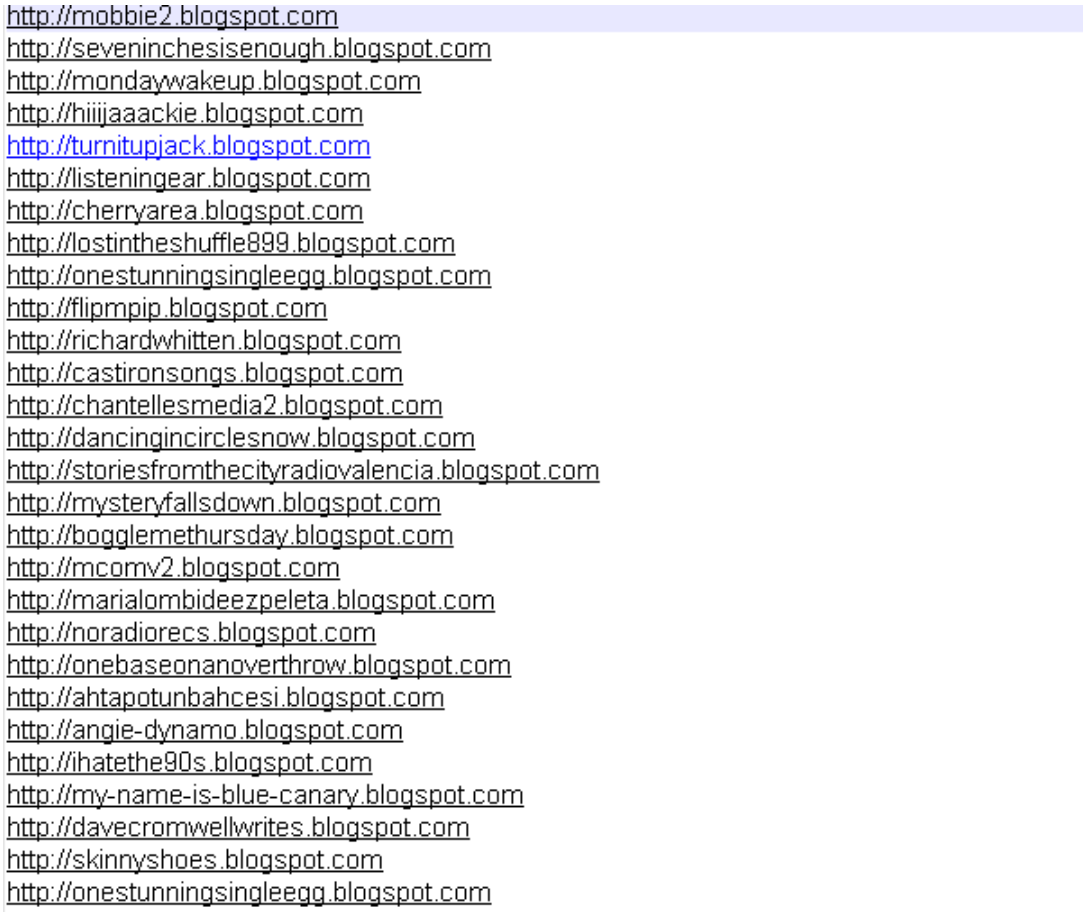

Figure 1: Sample list of Blog URIs

Sample atom URIs

http://mobbie2.blogspot.com/feeds/posts/default?alt=rss
http://seveninchesisenough.blogspot.com/feeds/posts/default?alt=rss
http://mondaywakeup.blogspot.com/feeds/posts/default?alt=rss
http://hiiijaaackie.blogspot.com/feeds/posts/default?alt=rss
http://turnitupjack.blogspot.com/feeds/posts/default?alt=rss
http://listeningear.blogspot.com/feeds/posts/default?alt=rss
http://cherryarea.blogspot.com/feeds/posts/default?alt=rss
http://lostintheshuffle899.blogspot.com/feeds/posts/default?alt=rss
http://onestunningsingleegg.blogspot.com/feeds/posts/default?alt=rss
http://flipmpip.blogspot.com/feeds/posts/default?alt=rss
http://richardwhitten.blogspot.com/feeds/posts/default?alt=rss
http://chantellesmedia2.blogspot.com/feeds/posts/default?alt=rss
http://dancingincirclesnow.blogspot.com/feeds/posts/default?alt=rss
http://storiesfromthecityradiovalencia.blogspot.com/feeds/posts/default?alt=rss
http://mysteryfallsdown.blogspot.com/feeds/posts/default?alt=rss
http://bogglemethursday.blogspot.com/feeds/posts/default?alt=rss
http://mcomv2.blogspot.com/feeds/posts/default?alt=rss
http://marialombideezpeleta.blogspot.com/feeds/posts/default?alt=rss
http://noradiorecs.blogspot.com/feeds/posts/default?alt=rss
http://onebaseonanoverthrow.blogspot.com/feeds/posts/default?alt=rss
http://ahtapotunbahcesi.blogspot.com/feeds/posts/default?alt=rss
http://angie-dynamo.blogspot.com/feeds/posts/default?alt=rss
http://ihatethe90s.blogspot.com/feeds/posts/default?alt=rss
http://my-name-is-blue-canary.blogspot.com/feeds/posts/default?alt=rss
http://davecromwellwrites.blogspot.com/feeds/posts/default?alt=rss
http://skinnyshoes.blogspot.com/feeds/posts/default?alt=rss

Figure 2: Sample list of Atom URIs

Sample number of pages

```
1|Our Podcast Could Be Your Life
1|funky little demons
1|Riley Haas' blog
1|MAGGOT CAVIAR
1|Blog Name Pending
16|Karl Drinkwater
1|But She's Not Stupid
1|KiDCHAIR
1|Time Is Poetry
1|Stonehill Sketchbook
1|Encore
14|THE HUB
1|Rod Shone
1|Kid F
1|La espiral de Joseph K
1|The Jeopardy of Contentment
1|MarkEOrtega's Journalism Portfolio
1|bittersweet
1|MR. BEAUTIFUL TRASH ART
1|forget about it
1|We Got Shit...A Pearl Jam Bootleg Site
1|The Campus Buzz on WSOU
1|Floorshime Zipper Boots
1|Desolation Row Records
1|Rants from the Pants
1|sweeping the kitchen
1|Lo importante es que estes tú bien
1|A2 MEDIA COURSEWORK JOINT BLOG
1|The World's First Internet Baby
1|Samtastic! Review
1|Azul Valentina
1|Party Full of Strangers
1|from a voice plantation
1|The Girl at the Rock Show
```

Figure 3: Sample list of the number of pages in each blog

Sample Blog matrix

Blog	doesn	found	young	light	real	pretty	kind	heart	hard	lot	friends	high	left	track	set	girl	thought	didn	play
Flatbasset	12	7	22	3	7	6	3	3	4	10	6	13	5	13	4	2	3	5	7
Riley Haas' blog	9	3	6	2	1	8	18	2	14	9	1	0	3	6	2	2	3	14	3
Party Full of Strangers	1	0	4	0	4	11	0	1	4	1	0	1	0	11	1	3	1	1	0
SEM REGRAS	0	0	1	1	0	1	1	7	0	0	0	0	1	0	0	3	0	0	0
Pithy Title Here	23	10	2	7	26	29	27	2	12	36	11	22	6	27	27	9	12	22	10
Morgan's Blog	2	0	11	1	2	0	2	1	3	1	2	2	1	2	2	1	1	1	1
MARISOL	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
THE HUB	0	0	1	0	0	0	0	0	0	2	1	2	1	0	0	0	0	0	0
Brian's Music Blog!!!	0	1	6	1	6	10	4	1	4	10	1	3	3	6	1	3	2	1	1
Web Science and Digital Li	4	26	1	10	4	3	2	0	9	2	5	10	9	8	16	1	5	2	2
Steel City Rust	4	7	6	0	7	13	3	1	4	6	2	5	6	11	1	1	5	11	1
MR. BEAUTIFUL TRASH ART	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ORGANMYTH	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
MarkEOrtega's Journalism	1	2	0	0	0	2	0	0	1	1	3	2	2	1	6	0	0	1	4
Green Eggs and Ham Monc	0	2	12	4	2	1	0	4	0	0	0	1	4	3	0	5	0	0	1
turnitup!	2	1	3	4	6	0	7	6	4	2	2	2	3	6	1	4	3	3	2
Stories From the City, Stor	1	2	6	5	2	0	1	5	0	1	7	2	1	1	8	7	0	0	5
Lost in the Shuffle	0	1	2	6	1	1	1	6	1	1	0	3	0	0	0	5	0	1	0
A H T A P O T	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Diagnosis: No Radio	19	2	11	9	8	32	44	4	12	27	6	8	12	5	13	8	13	9	7
Floorshime Zipper Boots	0	1	1	2	1	0	0	0	1	0	1	1	0	8	0	0	0	0	0
Did Not Chart	4	2	8	0	1	4	2	3	3	13	4	4	9	2	3	4	3	6	3
The Stearns Family	4	2	2	0	1	21	7	0	4	5	3	2	0	1	1	6	7	12	2
IoTube :)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Stonehill Sketchbook	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0
forget about it	1	1	0	0	1	1	0	0	1	1	0	0	1	0	0	0	6	5	0
DaveCromwell Writes	7	35	18	14	13	25	45	14	20	45	22	20	18	94	61	11	10	12	41
T H E V O I D S	2	5	0	0	7	0	0	0	1	3	1	2	1	2	2	0	2	5	0
Chantelle Swain A2 Media	0	3	0	3	1	0	0	0	0	0	4	0	0	7	9	0	0	0	0
The Campus Buzz on WSOI	2	2	20	3	0	3	0	2	1	0	1	1	15	0	0	6	0	0	2
jaaackie.	3	0	1	0	3	1	4	1	1	4	4	3	4	0	0	1	13	20	0
A2 MEDIA COURSEWORK J	0	8	0	0	1	0	0	0	1	5	0	0	5	2	4	0	5	17	0
The Girl at the Rock Show	2	3	4	0	2	6	2	4	4	7	5	12	0	4	2	2	2	6	2
Samtastic! Review	0	0	20	0	2	3	1	1	2	3	1	1	1	10	0	4	2	2	2
The Listening Ear	4	6	12	1	7	18	10	4	19	21	4	4	6	2	8	4	4	13	11
FlowRadio Playlists (and B	0	0	1	1	1	2	0	1	0	0	0	0	0	0	0	1	0	0	0
FOLK IS NOT HAPPY	2	0	1	6	5	1	5	2	4	0	2	1	3	3	4	0	0	0	3

Figure 4: Sample blogdata file

2 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).
```

2.1 Solution

1. In this question I was asked to create ASCII and JPEG Dendograms that clusters the most similar blogs.

2. In order to do this I used "clusters.py" code from Programming Collective Intelligence text. This code can be found in the listing4.

3. I imported this code into my python file and wrote small code for generating the ASCII and Dendograms. I wrote the code by referencing to the lecture slides.

4. This code can be seen in listing5.

5. Dendogram file can be seen in the fig5 and ASCII file can be seen in the fig6.

6. Unfortunately, Dendogram is difficult to see, but from this Dendogram we can infer that the blogs similar to "F-Measure" is "Samtastic! Review" and the blog similar to "'Web Science and Digital Library Research Group " is "Mile In Mine".

2.2 Code Listing

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
from PIL import Image, ImageDraw
from math import sqrt
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)


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)
```

```
71          closest = distance(clust[0].vec, clust[1].vec)
72
73      # loop through every pair looking for the smallest distance
74          for i in range(len(clust)):
75              for j in range(i + 1, len(clust)):
76          # distances is the cache of distance calculations
77                  if (clust[i].id, clust[j].id) not in distances:
78                      distances[(clust[i].id, clust[j].id)] = \
79                          distance(clust[i].vec, clust[j].vec)
80
81                  d = distances[(clust[i].id, clust[j].id)]
82
83                  if d < closest:
84                      closest = d
85                      lowestpair = (i, j)
86
87      # calculate the average of the two clusters
88          mergevec = [(clust[lowestpair[0]].vec[i] + clust[lowestpair[1]].vec[i])
89                      / 2.0 for i in range(len(clust[0].vec))]
90
91      # create the new cluster
92          newcluster = bicluster(mergevec, left=clust[lowestpair[0]],
93                                 right=clust[lowestpair[1]], distance=closest,
94                                 id=currentclustid)
95
96      # cluster ids that weren't in the original set are negative
97          currentclustid -= 1
98          del clust[lowestpair[1]]
99          del clust[lowestpair[0]]
100         clust.append(newcluster)
101
102     return clust[0]
103
104
105 def printclust(clust, labels=None, n=0):
106   # indent to make a hierarchy layout
107     for i in range(n):
108         print ' ',
109     if clust.id < 0:
110     # negative id means that this is branch
111         print '-'
112     else:
113     # positive id means that this is an endpoint
114         if labels == None:
115             print clust.id
116         else:
117             print labels[clust.id]
118
119   # now print the right and left branches
120     if clust.left != None:
121         printclust(clust.left, labels=labels, n=n + 1)
122     if clust.right != None:
123         printclust(clust.right, labels=labels, n=n + 1)
124
125
126 def getheight(clust):
127   # Is this an endpoint? Then the height is just 1
128     if clust.left == None and clust.right == None:
129         return 1
130
131   # Otherwise the height is the same of the heights of
132   # each branch
133     return getheight(clust.left) + getheight(clust.right)
134
135
136 def getdepth(clust):
137   # The distance of an endpoint is 0.0
138     if clust.left == None and clust.right == None:
139         return 0
140
141   # The distance of a branch is the greater of its two sides
142   # plus its own distance
```

```
143     return max(getdepth(clust.left), getdepth(clust.right)) + clust.distance
144
145
146 def drawdendrogram(clust, labels, jpeg='clusters.jpg'):
147   # height and width
148     h = getheight(clust) * 20
149     w = 1200
150     depth = getdepth(clust)
151
152   # width is fixed, so scale distances accordingly
153     scaling = float(w - 150) / depth
154
155   # Create a new image with a white background
156     img = Image.new('RGB', (w, h), (255, 255, 255))
157     draw = ImageDraw.Draw(img)
158
159     draw.line((0, h / 2, 10, h / 2), fill=(255, 0, 0))
160
161   # Draw the first node
162     drawnode(
163         draw,
164         clust,
165         10,
166         h / 2,
167         scaling,
168         labels,
169         )
170     img.save(jpeg, 'JPEG')
171
172
173 def drawnode(
174     draw,
175     clust,
176     x,
177     y,
178     scaling,
179     labels,
180     ):
181     if clust.id < 0:
182         h1 = getheight(clust.left) * 20
183         h2 = getheight(clust.right) * 20
184         top = y - (h1 + h2) / 2
185         bottom = y + (h1 + h2) / 2
186     # Line length
187         ll = clust.distance * scaling
188     # Vertical line from this cluster to children
189         draw.line((x, top + h1 / 2, x, bottom - h2 / 2), fill=(255, 0, 0))
190
191     # Horizontal line to left item
192         draw.line((x, top + h1 / 2, x + ll, top + h1 / 2), fill=(255, 0, 0))
193
194     # Horizontal line to right item
195         draw.line((x, bottom - h2 / 2, x + ll, bottom - h2 / 2), fill=(255, 0,
196                   0))
197
198     # Call the function to draw the left and right nodes
199         drawnode(
200             draw,
201             clust.left,
202             x + ll,
203             top + h1 / 2,
204             scaling,
205             labels,
206             )
207         drawnode(
208             draw,
209             clust.right,
210             x + ll,
211             bottom - h2 / 2,
212             scaling,
213             labels,
214             )
```

```
215     else:
216     # If this is an endpoint, draw the item label
217         draw.text((x + 5, y - 7), labels[clust.id], (0, 0, 0))
218
219
220 def rotatematrix(data):
221     newdata = []
222     for i in range(len(data[0])):
223         newrow = [data[j][i] for j in range(len(data))]
224         newdata.append(newrow)
225     return newdata
226
227
228 def kcluster(rows, distance=pearson, k=4):
229   # Determine the minimum and maximum values for each point
230     ranges = [(min([row[i] for row in rows]), max([row[i] for row in rows]))
231               for i in range(len(rows[0]))]
232
233   # Create k randomly placed centroids
234     clusters = [[random.random() * (ranges[i][1] - ranges[i][0]) + ranges[i][0]
235                 for i in range(len(rows[0]))] for j in range(k)]
236
237     lastmatches = None
238     for t in range(100):
239         print 'Iteration %d' % t
240         bestmatches = [[] for i in range(k)]
241
242     # Find which centroid is the closest for each row
243         for j in range(len(rows)):
244             row = rows[j]
245             bestmatch = 0
246             for i in range(k):
247                 d = distance(clusters[i], row)
248                 if d < distance(clusters[bestmatch], row):
249                     bestmatch = i
250             bestmatches[bestmatch].append(j)
251
252     # If the results are the same as last time, this is complete
253         if bestmatches == lastmatches:
254             break
255         lastmatches = bestmatches
256
257     # Move the centroids to the average of their members
258         for i in range(k):
259             avgs = [0.0] * len(rows[0])
260             if len(bestmatches[i]) > 0:
261                 for rowid in bestmatches[i]:
262                     for m in range(len(rows[rowid])):
263                         avgs[m] += rows[rowid][m]
264                 for j in range(len(avgs)):
265                     avgs[j] /= len(bestmatches[i])
266                 clusters[i] = avgs
267
268     return bestmatches
269
270
271 def tanamoto(v1, v2):
272     (c1, c2, shr) = (0, 0, 0)
273
274     for i in range(len(v1)):
275         if v1[i] != 0:  # in v1
276             c1 += 1
277         if v2[i] != 0:  # in v2
278             c2 += 1
279         if v1[i] != 0 and v2[i] != 0:  # in both
280             shr += 1
281
282     return 1.0 - float(shr) / (c1 + c2 - shr)
283
284
285 def scaledown(data, distance=pearson, rate=0.01):
286
```

```
287     n = len(data)
288
289   # The real distances between every pair of items
290     realdist = [[distance(data[i], data[j]) for j in range(n)] for i in
291                 range(0, n)]
292
293   # Randomly initialize the starting points of the locations in 2D
294     loc = [[random.random(), random.random()] for i in range(n)]
295     fakedist = [[0.0 for j in range(n)] for i in range(n)]
296
297     lasterror = None
298     for m in range(0, 1000):
299         print 'Iteration %d' % m
300     # Find projected distances
301         for i in range(n):
302             for j in range(n):
303                 fakedist[i][j] = sqrt(sum([pow(loc[i][x] - loc[j][x], 2)
304                                            for x in range(len(loc[i]))]))
305
306     # Move points
307         grad = [[0.0, 0.0] for i in range(n)]
308
309         totalerror = 0
310         for k in range(n):
311             for j in range(n):
312                 if j == k:
313                     continue
314         # The error is percent difference between the distances
315                 errorterm = (fakedist[j][k] - realdist[j][k]) / realdist[j][k]
316
317         # Each point needs to be moved away from or towards the other
318         # point in proportion to how much error it has
319                 grad[k][0] += (loc[k][0] - loc[j][0]) / fakedist[j][k] \
320                     * errorterm
321                 grad[k][1] += (loc[k][1] - loc[j][1]) / fakedist[j][k] \
322                     * errorterm
323
324         # Keep track of the total error
325                 totalerror += abs(errorterm)
326         print totalerror
327
328     # If the answer got worse by moving the points, we are done
329         if lasterror and lasterror < totalerror:
330             break
331         lasterror = totalerror
332
333     # Move each of the points by the learning rate times the gradient
334         for k in range(n):
335             loc[k][0] -= rate * grad[k][0]
336             loc[k][1] -= rate * grad[k][1]
337
338     return loc
339
340
341 def draw2d(data, labels, jpeg='mds2d.jpg'):
342     img = Image.new('RGB', (2000, 2000), (255, 255, 255))
343     draw = ImageDraw.Draw(img)
344     for i in range(len(data)):
345         x = (data[i][0] + 0.5) * 1000
346         y = (data[i][1] + 0.5) * 1000
347         draw.text((x, y), labels[i], (0, 0, 0))
348     img.save(jpeg, 'JPEG')
```

Listing 4: Python Code for clustering from PCI text

2.3 Code Listing

```
1 import clusters
2 blog,words,data=clusters.readfile('blogdata.txt')
3 variable = clusters.hcluster(data)
4
5 # print ASCII dendrogram
6 clusters.printclust(variable, labels=blog)
7
8 # save JPEG dendrogram
9 clusters.drawdendrogram(variable, blog, jpeg='clusterblog.jpg')
```

Listing 5: Python Code for getting ASCII and JPEG dendogram

2.4 Outputs

Dendogram

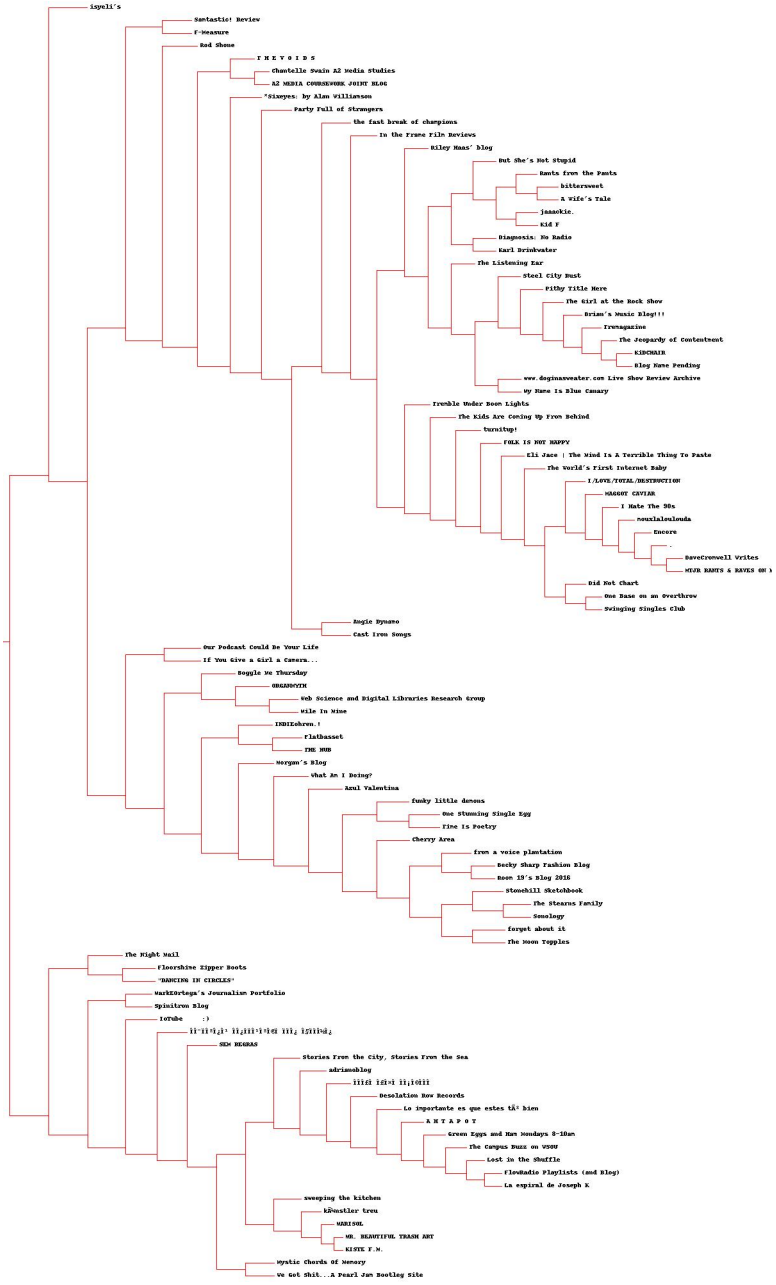

Figure 5: Dendogram clustering most similar blogs

ASCII file

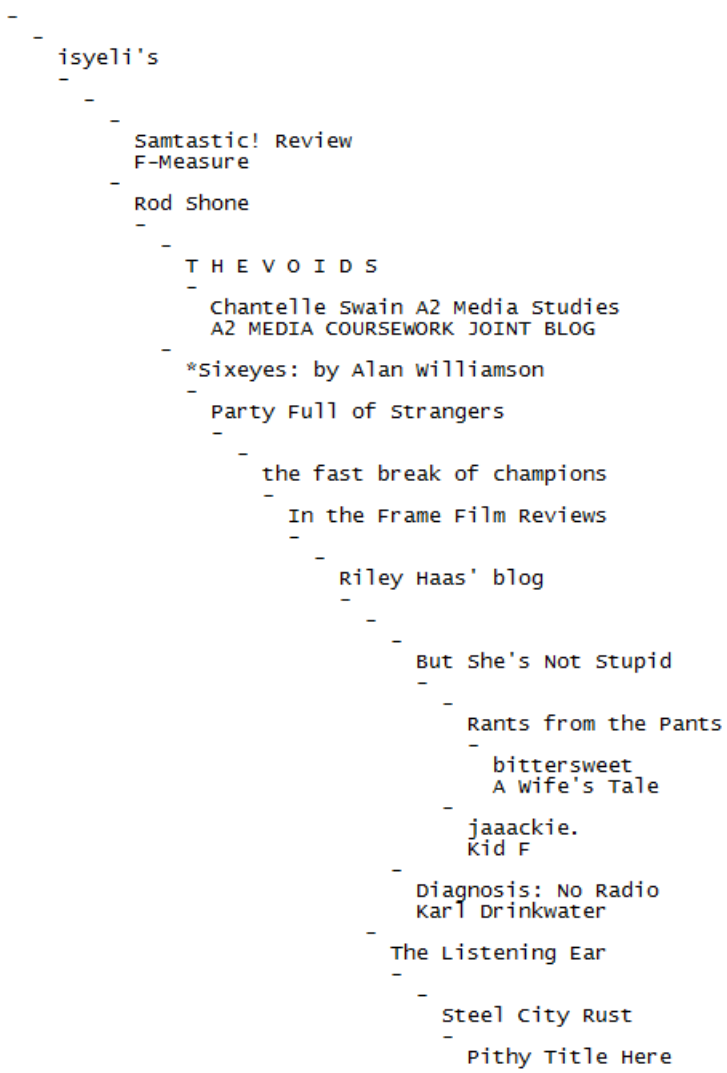

Figure 6: ASCII file showing clustering of most similar blogs

3 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?
```

3.1 Solution

1. In this question I was asked to cluster the blogs using K-Means, using K=5,10,20.
2. For doing this I used "clusters.py" code from Programming Collective Intelligence text.
3. My code for printing the values in each centroid and the number of iterations can be found in the fig6.
4. Number of Iterations for K=5 is 6.
5. Number of Iterations for K=10 is 7.
6. Number of Iterations for K=20 is 6.
7. These Iterations count change if we execute the program again. Sample list of list of Iterations can be found in the fig7.

3.2 Code Listing

```
1
2  import clusters
3  blog,words,data=clusters.readfile('blogdata.txt')
4
5  print "For k=5"
6  kclust=clusters.kcluster(data, k=5)
7  for i in range(0,5):
8          print "-----Blognames in centroid"+' '+str(i+1)+'-------'
9          for r in kclust[i]:
10                 print blog[r]
11         print '\n'
12 print "For k=10"
13 kclust=clusters.kcluster(data, k=10)
14 for i in range(0,10):
15         print "-----Blognames in centroid"+' '+str(i+1)+'-------'
16         for r in kclust[i]:
17                 print blog[r]
18         print '\n'
19
20 print "For k=20"
21 kclust=clusters.kcluster(data, k=20)
22 for i in range(0,20):
23         print "-----Blognames in centroid"+' '+str(i+1)+'-------'
24         for r in kclust[i]:
25                 print blog[r]
26         print '\n'
```

Listing 6: Python Code for generating top 5 and least 5 movie recommendations that my substitute should see

3.3 Output

Sample output file

```
sirius:~/Webscience/cs532-s16/Assignment 8/3> python q3.py
For k=5
Iteration 0
Iteration 1
Iteration 2
Iteration 3
Iteration 4
Iteration 5
-----Blognames in centroid 1-------
Party Full of Strangers
Brian's Music Blog!!!
Steel City Rust
MarkEOrtega's Journalism Portfolio
turnitup!
Floorshime Zipper Boots
Did Not Chart
DaveCromwell Writes
The Girl at the Rock Show
FOLK IS NOT HAPPY
MAGGOT CAVIAR
The World's First Internet Baby
MTJR RANTS & RAVES ON MUSIC
Tremble Under Boom Lights
One Base on an Overthrow
Eli Jace | The Mind Is A Terrible Thing To Paste
The Jeopardy of Contentment
.
*Sixeyes: by Alan Williamson
"DANCING IN CIRCLES"
KiDCHAIR
Blog Name Pending
Tremagazine
Swinging Singles Club
sweeping the kitchen
The Kids Are Coming Up From Behind
mouxlaloulouda
I/LOVE/TOTAL/DESTRUCTION
F-Measure
Encore
Cast Iron Songs
I Hate The 90s


-----Blognames in centroid 2-------
SEM REGRAS
ORGANMYTH
Diagnosis: No Radio
IoTube     :)
T H E V O I D S
Chantelle Swain A2 Media Studies
A2 MEDIA COURSEWORK JOINT BLOG
Angie Dynamo
Spinitron Blog
My Name Is Blue Canary
Boggle Me Thursday
Becky Sharp Fashion Blog
```

Figure 7: Sample values in each centroid and the number of iterations

4 Problem 4

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

4.1 Solution

1. I was asked to use Multi-dimensional scaling to create a JPEG of the blogs similar to the figure in the lecture slides.
2. To do this I used "clusters.py" code from Programming Collective Intelligence text.
3. I have made modifications to "cluster.py" code to print the number of iterations. Number of Iterations taken are "269".
4. My code for generating the JPEG and printing the iterations can be found in listing7.
5. JPEG image can be found in the following fig9 and sample list of the output can be found in this fig8.

4.2 Code Listing

Code Listing 1

```
#!/usr/local/bin/python

import clusters

blog,words,data=clusters.readfile('blogdata.txt')

coordinates = clusters.scaledown(data)

clusters.draw2d(coordinates, blog, jpeg='blogs.jpg')
```

Listing 7: Python code for creating a jpeg using mds

4.3 Outputs

Sample output file

```
sirius:~/Webscience/cs532-s16/Assignment 8/4> python q4.py
Iteration 0
4297.72408382
Iteration 1
3555.35920346
Iteration 2
3499.67746544
Iteration 3
3470.04208969
Iteration 4
3443.42045151
Iteration 5
3422.33321059
Iteration 6
3405.81659158
Iteration 7
3393.53119782
Iteration 8
3383.42760063
Iteration 9
3374.54720876
Iteration 10
3365.89998038
Iteration 11
3357.96516504
Iteration 12
3350.45565257
Iteration 13
3342.8696973
Iteration 14
3336.31179825
Iteration 15
3329.84152751
Iteration 16
3323.46976002
Iteration 17
3317.57138593
Iteration 18
3312.62413592
Iteration 19
3308.09688006
Iteration 20
3303.58275212
Iteration 21
3298.69188175
Iteration 22
```

Figure 8: Sample list of iterations

jpeg file

Figure 9: Sample jpeg file representing blogs

5 Problem 5

```
----Extra Credit 5 points----
 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.
```

5.1 Solution

1. In this question I have to compute a blog matrix again but this time with the proper TFIDF calculations instead of the frequency of occurrences of the words.
2. For this I have used concept from previous assignment on how to calculate TFIDF.Words having TFIDF values are chosen to be in the matrix.
3. I have calculated TF value for each word with respect to each blog which is inturn used to calculate TF values.
4. have used "generatefeedvector.py" again to generate the blog matrix and for generating the Dendogram I have used code from question 2.
5. My code for generating the blog matrix can be found in the listing8 and code for generating Dendogram and ASCII can be found in listing9.
6. Sample blog data can be found in the fig10.
7. Once I got the blog data I processed this data to be ASCII file and a Dendogram. This process is similar to the process in question 2.
8. Dendogram can be found in the fig12 and sample ASCII file can be found in the fig11.
9. When compared to the Dendogram in the question 2, similarties for the "F-Measure" and "Web Science" have changed.
10. In the previous Dendogram "F-Measure" is similar to "Samtastic! Review" and now it is similar to "I/Love/Total/Distruction".
11. In the previous Dendogram "Web Science and Digital Library Research Group" is similar to "Mile In Mine" and now it is similar to "The Moon Topples".
12. Comparitively number of clusters in the present Dendogram are more than the number of clusters in the Dendogram generated in question 2.
13. I have also observed that the almost all the clusters are around the same blogs but with different hierarchy.

5.2 Code Listing

Code Listing 1

```
import feedparser
import collections
import re
import operator
import math


file_out = "tfidf.txt"

def getwordcounts(url):

    fd = feedparser.parse(url)
    wc        = collections.defaultdict(int)
    stopwords = []

    stopWordList = open('stopWordList.txt').readlines()
    pages = len(fd['entries'])

    for stopWord in stopWordList:
        stopWord = stopWord.strip()
        stopwords.append(stopWord)

    for e in fd.entries:
        if 'summary' in e:
            summary = e.summary
        else:
            summary = e.description
        words = getwords('%s %s' % (e.title, summary))

        for word in words:
            if word not in stopwords:
                wc[word] += 1

    if pages == 500:
        next_link = url + "?start-index=501"
        d         = feedparser.parse(next_link)
        pages     = len(d['entries'])
        for e in d.entries:
            if 'summary' in e:
                summary = e.summary
            else:
                summary = e.description

            words = getwords('%s %s' % (e.title, summary))

            for word in words:
                if word not in stopwords:
                    #print word

                    wc[word] += 1

        if pages == 500:
            next_link = url + "?start-index=1001"
            for e in d.entries:
                if 'summary' in e:
                    summary = e.summary
                else:
                    summary = e.description

                words = getwords('%s %s' % (e.title, summary))

                for word in words:
                    if word not in stopwords:
                        #print word

                        wc[word] += 1
            if pages == 500:
                next_link = url + "?start-index=1501"
```

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

                    words = getwords('%s %s' % (e.title, summary))

                    for word in words:
                        if word not in stopwords:
                            wc[word] += 1

    if 'title' not in fd.feed:
        print 'Invalid url', url
        return 'bogus data', wc

    return fd.feed.title, wc

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

def main():

    # XXX: break this up into smaller functions, write tests for them

    apcount    = collections.defaultdict(int)
    wordcounts = {}
    feedlist   = open('100atomurls').readlines()
    totalWordCount = {}

    for url in feedlist:
        title, wc = getwordcounts(url)
        wordcounts[title] = wc

        for word, count in wc.iteritems():
            if count > 1:
                apcount[word] += 1

                try:
                    totalWordCount[word] += count
                except KeyError:
                    totalWordCount[word] = count

    wordlist = []


    for w, bc in apcount.iteritems():
        frac = float(bc)/len(feedlist)
        #print frac
        if frac > 0.1 and frac < 0.5:
            wordlist.append(w)

    countOfWords = []

    for word in wordlist:
        countOfWords.append((word,totalWordCount[word]))

    countOfWords.sort(key=lambda rating: rating[1], reverse = True )

    countOfWords = countOfWords[0:500]

    out = file(file_out, 'w')
    out.write('Blog')

    idfWordCount = {}

    for w in countOfWords:
        word = w[0]
        noOfBlogs = 0
        for blogname, counts in wordcounts.iteritems():
```

```
141
142                 if word in counts:
143                     noOfBlogs += 1
144                 #print noOfBlogs
145             idf = math.log( 100.0 / noOfBlogs   , 2 )
146
147
148             idfWordCount[word]  = idf
149
150
151         for w in countOfWords:
152             #print w
153             out.write('\t' + w[0])
154
155         out.write('\n')
156
157         for blogname, counts in wordcounts.iteritems():
158             blogname = blogname.encode('UTF-8')
159             out.write(blogname)
160
161             for w in countOfWords:
162                 word = w[0]
163                 occurance = w[1]
164
165                 tf = float(counts[word]) / occurance
166                 tfidf = tf * idfWordCount[word]
167
168                 # d is integer, f is float
169                 out.write('\t%f' % tfidf)
170
171             out.write('\n')
172
173         out.close()
174
175 if __name__ == '__main__':
176     main()
```

Listing 8: Python code for getting blog matrix using TFIDF scores

Code Listing 2

```
import clusters
blog,words,data=clusters.readfile('tfidf.txt')
variable = clusters.hcluster(data)

# print ASCII dendrogram
clusters.printclust(variable, labels=blog)

# save JPEG dendrogram
clusters.drawdendrogram(variable, blog, jpeg='clusterblogtfidf.jpg')
```

Listing 9: Python code for getting dendogram

5.3 Outputs

Sample blog matrix

Blog	mso	vocal	track	pop	subject	guitar	musical	http	style	review	record	voice	set	kind	www	sounds	lot	pretty	tracks	albums	didn	single	power	real
Flatbasset	0	0.003364	0.012127	0.020445	0.008454	0.007567	0	0	0.006408	0.00256	0.008912	0.001817	0.005266	0.00346	0	0.00402	0.013288	0.006445	0.014409	0.082252	0.008465	0.005048	0.013818	0.00856
Riley Haas' blog	0	0.003364	0.005597	0.000852	0.002818	0	0	0	0.00801	0.01024	0.015279	0.001817	0.002633	0.020759	0	0.010719	0.011959	0.008593	0.002402	0.00235	0.023703	0	0	0.001223
Party Full of Strangers	0	0.001682	0.010261	0.005963	0	0.001513	0.003708	0	0	0	0	0.005451	0.001317	0	0	0.00938	0.001329	0.011815	0	0.00235	0.001693	0.003366	0.001974	0.004891
SEM REGRAS	0	0	0	0.002556	0	0	0	0	0	0	0	0	0	0.001153	0	0	0	0.001074	0	0	0	0	0	0
Pithy Title Here	0.046024	0	0.025186	0.028111	0.008454	0.019675	0.009271	0	0.019223	0.00256	0.011459	0.016353	0.035546	0.031138	0	0.025459	0.047838	0.031149	0.026417	0.058751	0.037247	0.006731	0.005922	0.031794
Morgan's Blog	0	0.001682	0.001866	0	0	0.003027	0.001854	0	0	0	0	0	0.002633	0.002307	0	0.00134	0.001329	0	0.007205	0	0.001693	0.001683	0.001974	0.002446
MARISOL	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
THE HUB	0.06298	0	0	0.001704	0	0	0	0	0.01602	0	0	0	0	0	0	0	0.002658	0	0.002402	0	0	0	0	0
Brian's Music Blog!!!	0	0	0.005597	0.010222	0	0.003027	0.003708	0.019591	0.003204	0	0.001273	0.009085	0.001317	0.004613	0.036261	0.00134	0.013288	0.010741	0.007205	0.035251	0.001693	0.001683	0	0.007337
Web Science and Digital Li	0	0	0.007462	0	0	0	0	0.254686	0.00801	0.017921	0.012732	0	0.021064	0.002307	0.081587	0	0.002658	0.003222	0.002402	0	0.003386	0.037022	0	0.004891
Steel City Rust	0	0.003364	0.010261	0.008519	0.002818	0.007567	0.005562	0	0.003204	0.00512	0.014005	0.005451	0.001317	0.00346	0	0.00402	0.007973	0.013964	0.007205	0.0047	0.018623	0.005048	0.00987	0.00856
MR. BEAUTIFUL TRASH ART	0	0	0	0.005111	0	0	0	0.003265	0	0	0	0	0	0	0.013598	0	0	0	0	0	0	0	0	0
ORGANMYTH	0	0	0	0	0	0	0	0	0.001602	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
MarkEOrtega's Journalism	0	0.001682	0.000933	0.008519	0	0	0	0	0	0	0.002546	0	0.007899	0	0	0.00134	0.001329	0.002148	0.002402	0.0094	0.001693	0.001683	0	0
Green Eggs and Ham Monc	0.033912	0	0.002798	0.000852	0	0	0	0	0.00801	0.00256	0	0	0	0	0	0	0	0.001074	0	0	0	0	0.003948	0.002446
turnitup!	0	0.003364	0.005597	0.014482	0	0.007567	0.020395	0	0.004806	0	0.001273	0.001817	0.001317	0.008073	0	0.00938	0.002658	0	0.019212	0.01645	0.005079	0.005048	0.001974	0.007337
Stories From the City, Stor	0	0	0.000933	0.007667	0	0	0	0.022856	0	0	0.001273	0	0.010532	0.001153	0	0.00134	0.001329	0	0	0	0	0	0	0.002446
Lost in the Shuffle	0	0	0	0	0	0	0	0	0	0	0	0	0	0.001153	0	0	0.001329	0.001074	0	0	0.001693	0	0.001974	0.001223
A H T A P O T	0	0	0	0	0	0.001513	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.001223
Diagnosis: No Radio	0.072669	0	0.004664	0.000852	0.002818	0.001513	0.007417	0.003265	0.040049	0.00768	0.007639	0	0.017115	0.050744	0	0.00134	0.035878	0.034372	0.012008	0.0094	0.015237	0.006731	0.013818	0.009783
Floorshime Zipper Boots	0	0.001682	0.007462	0.004259	0	0.003027	0.003708	0.003265	0.001602	0	0.006366	0	0	0	0	0.00268	0	0	0.074448	0.00235	0	0.005048	0.001974	0.001223
Did Not Chart	0	0.005046	0.001866	0.017889	0	0.010594	0.001854	0	0.00801	0	0.015279	0.001817	0.00395	0.002307	0	0.00804	0.017275	0.004296	0	0.01645	0.010158	0.035339	0	0.001223
The Stearns Family	0	0	0.000933	0	0	0	0	0	0	0	0	0	0.001317	0.008073	0	0.00134	0.006644	0.022556	0	0	0.020316	0	0	0.001223
IoTube :)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Stonehill Sketchbook	0	0	0	0	0	0.001513	0	0	0	0	0	0	0	0	0	0	0	0.001074	0	0	0	0	0	0
forget about it	0	0	0	0	0	0	0	0	0	0	0	0.001817	0	0	0	0	0.001329	0.001074	0	0	0.008465	0	0	0.001223
DaveCromwell Writes	0	0.104289	0.087684	0.034926	0.028179	0.198262	0.040791	0.042448	0.097719	0.033281	0.10313	0.038158	0.080307	0.051897	0.027196	0.048238	0.059797	0.026853	0.132086	0.047001	0.020316	0.070678	0.03948	0.015897
T H E V O I D S	0.171983	0	0.001866	0.001704	0	0.003027	0	0.003265	0.059272	0	0.002546	0	0.002633	0	0	0	0.003986	0	0	0.00235	0.008465	0	0.001974	0.00856
Chantelle Swain A2 Media	0	0	0.00653	0.000852	0	0	0	0	0	0	0.002546	0	0.011849	0	0	0	0	0	0.002402	0.00235	0	0.003366	0	0.001223
The Campus Buzz on WSOI	0	0	0	0.005963	0	0	0	0	0	0	0.006366	0	0	0	0	0	0	0.003222	0	0.00235	0	0	0.001974	0
jaaackie.	0	0	0	0	0	0	0	0	0	0	0	0	0	0.004613	0	0.00268	0.005315	0.001074	0	0	0.033861	0	0	0.003669
A2 MEDIA COURSEWORK J	0	0	0.001866	0	0	0.00454	0.001854	0.003265	0.003204	0	0.007639	0	0.005266	0	0.004533	0	0.006644	0	0	0	0.028782	0	0.001974	0.001223
The Girl at the Rock Show	0.036334	0	0.003731	0.008519	0	0.001513	0.012979	0	0.012816	0.00512	0.006366	0.001817	0.002633	0.002307	0	0.00402	0.009302	0.006445	0.004803	0.032901	0.010158	0.001683	0.003948	0.002446

Figure 10: Sample blog matrix formed using TFIDF values

Sample ASCII file

```
-
  The Night Mail
  -
    -
      -
        -
          -
            SEM REGRAS
            -
              What Am I Doing?
              Time Is Poetry
          -
            If You Give a Girl a Camera...
            In the Frame Film Reviews
        -
          -
            Steel City Rust
            -
              My Name Is Blue Canary
              -
                MARISOL
                Stonehill Sketchbook
          -
            Desolation Row Records
            -
              Swinging Singles Club
              -
                -
                  I/LOVE/TOTAL/DESTRUCTION
                  F-Measure
                -
                  Tremble Under Boom Lights
                  One Base on an Overthrow
      -
        -
          Spinitron Blog
          Azul Valentina
        -
          -
            Cherry Area
            -
              -
                -
                  Kid F
                  -
                    Sonology
                    Rants from the Pants
                -
                  forget about it
                  -
                    jaaackie.
                    -
```

Figure 11: Sample ASCII file showing clustering of most similar blogs

Dendogram

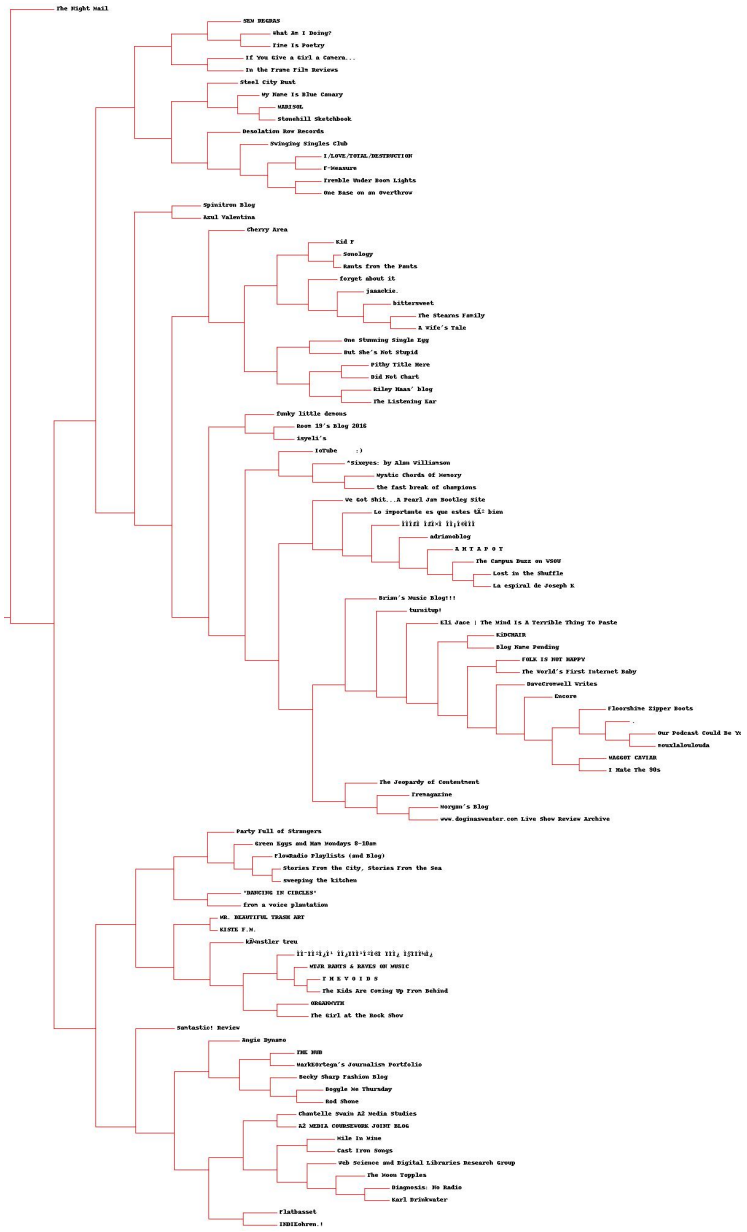

Figure 12: Dendogram showing clustering of most similar blogs

Bibliography

[1] cataska, programming-collective-intelligence-code, https://github.com/cataska/programming-collective-intelligence-code/tree/master/chapter3, 2011

[2] kurtmckee,feedparser 5.2.1,https://pypi.python.org/pypi/feedparser, 2015

[3] Renato Alves, python-feedparser, https://github.com/Unode/python-feedparser/tree/master/feedparser, 2012

[4] Wikipedia, tfidf, https://en.wikipedia.org/wiki/Tf%E2%80%93idf, 2016