CS532 Web Science: Assignment 8

Finished on April 7, 2016

Dr. Michael L. Nelson

Naina Sai Tipparti
ntippart@cs.odu.edu

Contents

Listings

List of Figures

Problem 1

Question

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 "blog-data.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.

Answer

To complete this assignment, a blog word count matrix was required. To start off, a list of blog URIs was obtained using the method described in class, implemented as the `get_uris.py` script, which can be found in Appendix A, Listing 27. Two default blogs, F-Measure and the Old Dominion Web Science and Digital Libraries blogs, were added as defaults to the initial URI list and then, using the seed URI provided (Listing 1), the remaining 98 URIs from random blogs within the blogger.com family were added.

```
default = 'http://www.blogger.com/next-blog?navBar=true&blogID=3471633091411211117'
must_haves = ['http://f-measure.blogspot.com/', 'http://ws-dl.blogspot.com/']
```

Listing 1: referenced variables in get_uris.py

The `get_uris main` function in Listing 3 was the driver that called the `get_atom` function (shown in Listing 4) to extract the atom [1] URIs from each blog and add them to the set of URIs with the `add_uri` function, shown in Listing 5.

```
http://jbalow.blogspot.com/feeds/posts/default
http://toxicreine.blogspot.com/feeds/posts/default
http://beautifulsweetpea.blogspot.com/feeds/posts/default
http://brittanyvsutah.blogspot.com/feeds/posts/default
http://thewoolfpack6.blogspot.com/feeds/posts/default
http://jennie-tabs-wedding.blogspot.com/feeds/posts/default
http://kamielverwer.blogspot.com/feeds/posts/default
http://azstampcrazy.blogspot.com/feeds/posts/default
http://deneiserothenberger.blogspot.com/feeds/posts/default
http://gobobgo-anewday.blogspot.com/feeds/posts/default
```

Listing 2: Sample list of Blog URI's

```
if __name__ == '__main__':
    uris = set()
    with open('blog_uris', 'a') as outfile:
        if len(sys.argv) > 1 and sys.argv[1] == 'new':
            for must_have in must_haves:
                uri = get_atom(must_have)
                add_uri(uri, uris, outfile)
        else:
            with open('blog_uris') as infile:
                [uris.add(line.strip()) for line in infile]
        while len(uris) < 100:
            uri = get_atom(default)
            add_uri(uri, uris, outfile)
```

Listing 3: main for get_uris.py

```
def get_atom(uri):
    try:
        r = requests.get(uri)
    except Exception, e:
        return None
    soup = BeautifulSoup(r.text)
    links = soup.find_all('link', {'type':'application/atom+xml'})
    if links:
        return str(links[0]['href'])
    return None
```

Listing 4: get_atom function

```
def add_uri(uri, uris, outfile):
    if uri and uri not in uris:
        uris.add(uri)
        outfile.write(uri + '\n')
        print len(uris), uri
```

Listing 5: add_uri function

After the full list of 100 URIs was obtained, page counts for each blog were extracted and saved to a file called `pagecounts` using the `matrix.py` script. This script is a modified version of `generatefeedvectors.py` from the book *Programming Collective Intelligence* [2] and can be found in full in Appendix A, Listing 28.

The code responsible for downloading the blogs and counting the words in each is shown in Listing 6, which calls the `get_titles`, `get_words` and `get_next` functions found in Listing 7. This code loops over the list of URIs that was obtained with the `get_uris.py` script (Listing 27), parses each entry, and extracts all the words in each entry's title. These word counts are then saved as a python dictionary to the hard drive for later use.

```
if __name__ == '__main__':
    with open('blog_uris') as infile:
        uris = [line.strip() for line in infile if line.strip()]
    if len(sys.argv) == 2 and sys.argv[1] == 'get':
        with futures.ThreadPoolExecutor(max_workers=8) as executor:
            uri_futures = [executor.submit(get_titles, uri) for uri in uris]
            for future in futures.as_completed(uri_futures):
                uri, title, subtitle, pages, wc = future.result()
                with open('wcs/' + md5.new(uri).hexdigest(), 'w') as out:
                    out.write(title + ': ' + subtitle + '\t' + str(pages) + '\t')
                    json.dump(wc, out)
```

Listing 6: looping over the URIs

```
def get_next(d):
    for item in d.feed.links:
        if item['rel'] == u'next':
            return item['href']
    return None

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

def get_titles(uri):
    print('processing {}'.format(uri))
    next = uri
    wc = {}
    pages = 0
    while next is not None:
        d = feedparser.parse(next)
        for e in d.entries:
            words = get_words(e.title.encode('utf-8'))
            for word in words:
                wc.setdefault(word, 0)
                wc[word] += 1
        pages += 1
        next = get_next(d)
        print('next {}'.format(next))
    title = d.feed.title.encode('utf-8')
    subtitle = d.feed.subtitle[:50].encode('utf-8')
    print('finished: {}: {}'.format(title, subtitle))
    return uri, title, subtitle, pages, wc
```

Listing 7: processing each blog

The parsed results were then read by the code in Listing 8. This code used the `load_data` and `build_wordlist` functions in Listing 9 and 10 to read each of the blog word counts and then created four collections to organize them all:

1. `apcount`: A dictionary containing the count for all words combined
2. `wordcounts`: A dictionary containing each blog's individual word count
3. `pagecounts`: A dictionary containing each blog's page count
4. `wordlist`: A list containing all of the words found in each blog

```
        apcount, wordcounts, pagecounts = load_data(uris)
        wordlist = build_wordlist(apcount, uris)
        if len(sys.argv) == 2 and sys.argv[1] == 'pages':
            with open('pagecounts', 'w') as outfile:
                outfile.write('blog\tpages\n')
                for blog, pagecount in pagecounts.iteritems():
                    outfile.write("\"" + blog.replace("\"", "") + "\"" + '\t' + str(
                        pagecount) + '\n')
        elif len(sys.argv) == 2 and sys.argv[1] == 'wc':
            write_data('blogdata1.txt', wordlist, wordcounts)
```

Listing 8: creating the blog data matrix

```
def load_data(uris):
    apcount = {}
    wordcounts = {}
    pagecounts = {}
    for uri in uris:
        with open('wcs/' + md5.new(uri).hexdigest()) as infile:
            try:
                lines = infile.read().split('\t')
                title = lines[0]
                pages = int(lines[1])
                wc = json.loads(lines[2])
            except Exception, e:
                print('*** {} generated an exception: {}'.format(uri, e))
                continue
        wordcounts[title] = wc
        pagecounts[title] = pages
        for word, count in wc.items():
            apcount.setdefault(word, 0)
            apcount[word] += count
    return apcount, wordcounts, pagecounts
```

Listing 9: loading the data

```
def build_wordlist(apcount, uris):
    wordlist = []
    for w, bc in sorted(apcount.items(), key=lambda x: x[1], reverse=True):
            frac = float(bc) / len(uris)
            if frac > 0.1 and frac < 0.5:
                wordlist.append(w)
    return wordlist
```

Listing 10: building the master wordlist

The code in Listing 25 then created the matrix using the `write_data` function using the data structures that store the blog word counts.

```
def write_data(filename, wordlist, wordcounts, form=lambda wc, word, wordcounts: wc[word]):
    with open(filename, 'w') as out:
        out.write('Blog')
        for word in wordlist[:500]:
            out.write('\t%s' % word)
        out.write('\n')
        for blog, wc in wordcounts.items():
            print blog
            out.write(blog)
            for word in wordlist[:500]:
                if word in wc:
                    out.write('\t{}'.format(form(wc,word,wordcounts)))
                else:
                    out.write('\t0')
            out.write('\n')
```

Listing 11: writing the data

Problem 2

Question

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).

Answer

The ascii and jpeg dendrograms were created using the code shown in Listing 12, which is modeled after the example from class.

```
    blognames, words, data = readfile('q1/blogdata1.txt')
    clust = hcluster(data)
    with open('dendrogram.txt', 'w') as outfile:
        stdout = sys.stdout
290     sys.stdout = outfile
        printclust(clust, labels=blognames)
        sys.stdout = stdout
    drawdendrogram(clust, blognames, jpeg='blogclust.jpg')
```

Listing 12: creating the dendrograms

The `readfile` function shown in Listing 13 was used to read the data that was compiled from Question 1 into memory where it is then processed by the `hcluster` function found in Listing 14 to produce the clustered representation of the blogs.

```
def readfile(filename):
  lines=[line for line in file(filename)]
5
  # First line is the column titles
  colnames=lines[0].strip().split('\t')[1:]
  rownames=[]
  data=[]
10 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
15  data.append([float(x) for x in p[1:]])
  return rownames,colnames,data
```

Listing 13: creating the dendrograms

```
def hcluster(rows,distance=pearson):
  distances={}
50 currentclustid=-1

  # Clusters are initially just the rows
  clust=[bicluster(rows[i],id=i) for i in range(len(rows))]

55 while len(clust)>1:
    lowestpair=(0,1)
    closest=distance(clust[0].vec,clust[1].vec)

    # loop through every pair looking for the smallest distance
60  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]
```

Listing 14: hcluster function

The `printclust` function from Listing 15 prints the ascii dendrogram of the cluster object parameter to sys.stdout, which is redirected to write to a file with the code in Listing 12.

```
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)
```

Listing 15: printclust function

The `drawdendrogram` function from Listing 16 creates a jpeg image of the cluster, which is shown in Figure 1.

```
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')
```

Listing 16: drawdendrogram function

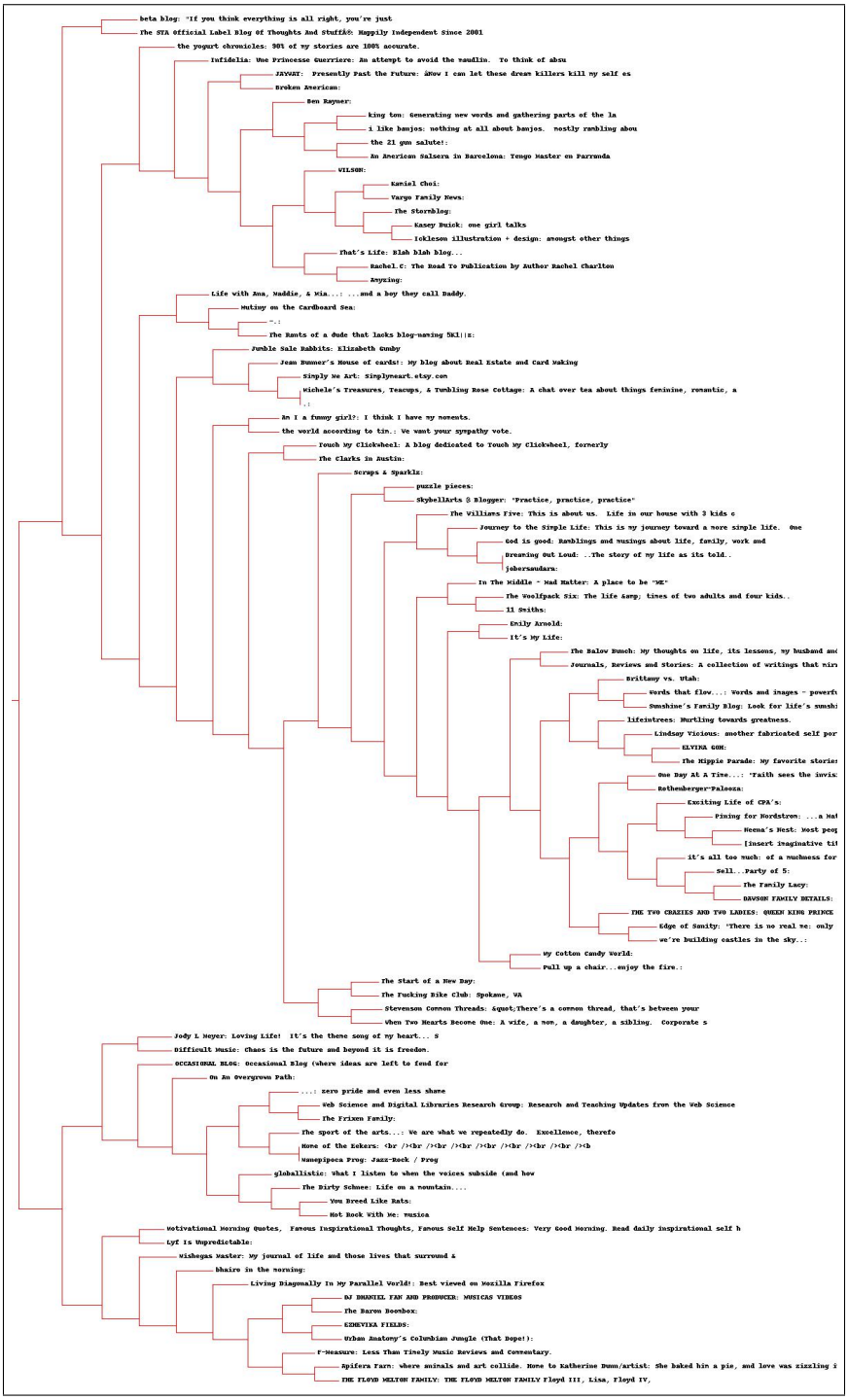

Figure 1: blog dendrogram

Problem 3

Question

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?

Answer

Using the code in Listing 17 kclustering was performed with values for $n = 5$, $n = 10$ and $n = 20$. The main function calls the kcluster function, which is shown in Listing 18.

```
295     outfile=open('kclust.txt','w')
        for i in [5,10,20] :
            kclust,iternum=kcluster(data,k=i)
            outfile.write('\n\n k = %d'%i)
            outfile.write('Iterations = %d\n'%iternum)
300         for cluster in kclust:
                outfile.write('[')
                for blogidx in cluster:
                    outfile.write(blognames[blogidx]+', ')
                outfile.write(']\n')
305     outfile.close()
```

Listing 17: kclustering main

```
def kcluster(rows,distance=pearson,k=4):
175   # 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
180 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):
185   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)):
190     row=rows[j]
        bestmatch=0
        for i in range(k):
          d=distance(clusters[i],row)
          if d<distance(clusters[bestmatch],row): bestmatch=i
195     bestmatches[bestmatch].append(j)

      # If the results are the same as last time, this is complete
      if bestmatches==lastmatches: break
      lastmatches=bestmatches
200
      # 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:
205       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])
210       clusters[i]=avgs
```

```
    return bestmatches
```

Listing 18: kcluster function

The output is shown in Listing 19. As the output reads, a kcluster with $n = 5$ required eight iterations, $n = 10$ required five iterations and $n = 20$ also required five iterations.

```
ïż£Done with dendrograms
K=5
Iteration 0
Iteration 1
Iteration 2
Iteration 3
Iteration 4
Iteration 5
Iteration 6
Iteration 7
K=10
Iteration 0
Iteration 1
Iteration 2
Iteration 3
Iteration 4
K=20
Iteration 0
Iteration 1
Iteration 2
Iteration 3
Iteration 4
 k= 5[Michele's Treasures, Teacups, & Tumbling Rose Cottage: A chat over tea about things
    feminine, romantic, a, jobersaudara: , The Baron Boombox: , Am I a funny girl?: I think
     I have my moments., Sunshine's Family Blog: Look for life's sunshine and you'll find
    it!!!, Manepipoca Prog: Jazz-Rock / Prog, Urban Anatomy's Columbian Jungle (That Dope!)
    : , Exciting Life of CPA's: , Mutiny on the Cardboard Sea: , Touch My Clickwheel: A
    blog dedicated to Touch My Clickwheel, formerly , The Hippie Parade: My favorite
    stories lined up one right after anoth, The STA Official Label Blog Of Thoughts And
    StuffÂő: Happily Independent Since 2001, The Rants of a dude that lacks blog-naming 5K1
    ||z: , Simply Me Art: Simplymeart.etsy.com, .: , Scraps & Sparklz: , the world
    according to tim.: We want your sympathy vote., ]
[bhairo in the morning: , The Start of a New Day: , beta blog: "If you think everything is
    all right, you're just, Apifera Farm: where animals and art collide. Home to Katherine
    Dunn/artist: She baked him a pie, and love was zizzling inside , Jumble Sale Rabbits:
    Elizabeth Gunby, lifeintrees: Hurtling towards greatness., THE FLOYD MELTON FAMILY: THE
    FLOYD MELTON FAMILY Floyd III, Lisa, Floyd IV,, THE TWO CRAZIES AND TWO LADIES: QUEEN
    KING PRINCE PRINCESS, An American Salsera in Barcelona: Tengo Master en Parranda,
    OCCASIONAL BLOG: Occasional Blog (where ideas are left to fend for, It's My Life: , F-
    Measure: Less Than Timely Music Reviews and Commentary., ]
[DJ DHANIEL FAN AND PRODUCER: MUSICAS VIDEOS, The Dirty Schnee: Life on a mountain...,
    Living Diagonally In My Parallel World!: Best viewed on Mozilla Firefox, Edge of Sanity:
     "There is no real me: only an entity, something il, When Two Hearts Become One: A wife,
     a mom, a daughter, a sibling. Corporate s, Sell...Party of 5: , On An Overgrown Path:
    , Infidelia: Une Princesse Guerriere: An attempt to avoid the maudlin. To think of absu
    , Life with Ana, Maddie, & Mia...: ...and a boy they call Daddy., ]
[king ton: Generating new words and gathering parts of the la, JAYWAT: Presently Past the
    Future: âĂIJNow I can let these dream killers kill my self es, The Balow Bunch: My
    thoughts on life, its lessons, my husband and p, The Williams Five: This is about us.
    Life in our house with 3 kids c, the yogurt chronicles: 90% of my stories are 100%
    accurate., You Breed Like Rats: , Motivational Morning Quotes, Famous Inspirational
    Thoughts, Famous Self Help Sentences: Very Good Morning. Read daily inspirational self h
    , God is good: Ramblings and musings about life, family, work and, The Clarks in Austin:
     , Lyf Is Unpredictable: , Journey to the Simple Life: This is my journey toward a more
    simple life. One, globallistic: What I listen to when the voices subside (and how , The
     Fucking Bike Club: Spokane, WA, ]
[Dreaming Out Loud: ..The story of my life as its told.., Home of the Eekers: <br /><br /><
    br /><br /><br /><br /><br /><br /><b, Words that flow...: Words and images - powerful
    elements of our everyd, -.: , EZHEVIKA FIELDS: , That's Life: Blah blah blog..., The
```

```
    Family Lacy: , Pining for Nordstrom: ...a Materialist Misplaced in a Nordstrom-less Tow,
     Mishegas Master: My journal of life and those lives that surround &, DAWSON FAMILY
    DETAILS: , Neena's Nest: Most people are just about as happy as they make u, Kamiel Choi
    : , The Woolfpack Six: The life &amp;amp; times of two adults and four kids.., Stevenson
    Common Threads: &amp;quot;There's a common thread, that's between your, Jody L Meyer: Loving
     Life!  It's the theme song of my heart... S, Ben Rayner: , Vargo Family News: , One Day
     At A Time...: "Faith sees the invisible, believes the unbelievab, ...: zero pride and
    even less shame, My Cotton Candy World: , it's all too much: of a muchness for me., Pull
     up a chair...enjoy the fire.: , i like banjos: nothing at all about banjos.  mostly
    rambling abou, Emily Arnold: , Web Science and Digital Libraries Research Group:
    Research and Teaching Updates from the Web Science, ELVINA GOH: , Kasey Buick: one girl
    talks, Jean Bunner's House of cards!: My blog about Real Estate and Card Making, WILSON:
     , Ickleson illustration + design: amongst other things, Journals, Reviews and Stories:
    A collection of writings that mirror my thoughts a, 11 Smiths: , the 21 gun salute!: ,
    puzzle pieces: , SkybellArts @ Blogger: "Practice, practice, practice", Brittany vs.
    Utah: , Rothenberger~Palooza: , [insert imaginative title here]: , In The Middle ~ Mad
    Hatter: A place to be "ME", Hot Rock With Me: musica, The sport of the arts...: We are
    what we repeatedly do.  Excellence, therefo, The Frixen Family: , Difficult Music: Chaos
     is the future and beyond it is freedom., The Stormblog: , Rachel.C: The Road To
    Publication by Author Rachel Charlton, we're building castles in the sky..: , Amyzing: ,
     Broken American: , Lindsay Vicious: another fabricated self portrait, ]
28
 k= 10[Michele's Treasures, Teacups, & Tumbling Rose Cottage: A chat over tea about things
     feminine, romantic, a, jobersaudara: , The Baron Boombox: , Jody L Meyer: Loving Life!
      It's the theme song of my heart... S, The Dirty Schnee: Life on a mountain...., ,
     Sunshine's Family Blog: Look for life's sunshine and you'll find it!!!, Apifera Farm:
     where animals and art collide. Home to Katherine Dunn/artist: She baked him a pie, and
     love was zizzling inside , Manepipoca Prog: Jazz-Rock / Prog, lifeintrees: Hurtling
     towards greatness., Simply Me Art: Simplymeart.etsy.com, Hot Rock With Me: musica, .: ,
      ]
[-.: , DJ DHANIEL FAN AND PRODUCER: MUSICAS VIDEOS, JAYWAT:  Presently Past the Future:
    âĂIJNow I can let these dream killers kill my self es, The Woolfpack Six: The life &amp;amp;
    times of two adults and four kids.., the yogurt chronicles: 90% of my stories are 100%
    accurate., THE FLOYD MELTON FAMILY: THE FLOYD MELTON FAMILY Floyd III, Lisa, Floyd IV,,
    ]
[Dreaming Out Loud: ..The story of my life as its told.., Home of the Eekers: <br /><br /><
    br /><br /><br /><br /><br /><br /><b, The Family Lacy: , Neena's Nest: Most people are
    just about as happy as they make u, Stevenson Common Threads: &amp;quot;There's a common
    thread, that's between your, Vargo Family News: , One Day At A Time...: "Faith sees the
    invisible, believes the unbelievab, The Balow Bunch: My thoughts on life, its lessons,
    my husband and p, it's all too much: of a muchness for me., Pull up a chair...enjoy the
    fire.: , Kasey Buick: one girl talks, Edge of Sanity: "There is no real me: only an
    entity, something il, Jean Bunner's House of cards!: My blog about Real Estate and Card
    Making, Ickleson illustration + design: amongst other things, Journals, Reviews and
    Stories: A collection of writings that mirror my thoughts a, When Two Hearts Become One:
     A wife, a mom, a daughter, a sibling.  Corporate s, God is good: Ramblings and musings
    about life, family, work and, THE TWO CRAZIES AND TWO LADIES: QUEEN KING PRINCE PRINCESS
    , Rothenberger~Palooza: , The sport of the arts...: We are what we repeatedly do.
    Excellence, therefo, OCCASIONAL BLOG: Occasional Blog (where ideas are left to fend for,
     Journey to the Simple Life: This is my journey toward a more simple life.  One, The
    Stormblog: , Rachel.C: The Road To Publication by Author Rachel Charlton, Lindsay
    Vicious: another fabricated self portrait, ]
[bhairo in the morning: , Mishegas Master: My journal of life and those lives that surround
    &, Kamiel Choi: , king ton: Generating new words and gathering parts of the la, Urban
    Anatomy's Columbian Jungle (That Dope!): , Exciting Life of CPA's: , Difficult Music:
    Chaos is the future and beyond it is freedom., Amyzing: , Infidelia: Une Princesse
    Guerriere: An attempt to avoid the maudlin.  To think of absu, ]
33 [DAWSON FAMILY DETAILS: , The Start of a New Day: , My Cotton Candy World: , The Williams
    Five: This is about us.  Life in our house with 3 kids c, WILSON: , the 21 gun salute!:
    , Mutiny on the Cardboard Sea: , SkybellArts @ Blogger: "Practice, practice, practice",
    The Rants of a dude that lacks blog-naming 5K1||z: , In The Middle ~ Mad Hatter: A place
     to be "ME", The Fucking Bike Club: Spokane, WA, Broken American: , Life with Ana,
    Maddie, & Mia...: ...and a boy they call Daddy., ]
[Am I a funny girl?: I think I have my moments., You Breed Like Rats: , 11 Smiths: , Touch
    My Clickwheel: A blog dedicated to Touch My Clickwheel, formerly , The Clarks in Austin:
     , An American Salsera in Barcelona: Tengo Master en Parranda, The Frixen Family: ,
    globallistic: What I listen to when the voices subside (and how , It's My Life: , the
    world according to tim.: We want your sympathy vote., ]
```

```
[Words that flow...: Words and images − powerful elements of our everyd, That's Life: Blah
    blah blog..., i like banjos: nothing at all about banjos.  mostly rambling abou, ELVINA
    GOH: , Brittany vs. Utah: , Scraps & Sparklz: , we're building castles in the sky..: , F
    −Measure: Less Than Timely Music Reviews and Commentary., ]
[Ben Rayner: , ...: zero pride and even less shame, beta blog: "If you think everything is
    all right, you're just, Jumble Sale Rabbits: Elizabeth Gunby, puzzle pieces: , The STA
    Official Label Blog Of Thoughts And StuffÂő: Happily Independent Since 2001, ]
[EZHEVIKA FIELDS: , Emily Arnold: , Web Science and Digital Libraries Research Group:
    Research and Teaching Updates from the Web Science, Sell...Party of 5: , On An Overgrown
     Path: , ]
[Pining for Nordstrom: ...a Materialist Misplaced in a Nordstrom−less Tow, Living Diagonally
    In My Parallel World!: Best viewed on Mozilla Firefox, Motivational Morning Quotes,
    Famous Inspirational Thoughts, Famous Self Help Sentences: Very Good Morning. Read daily
     inspirational self h, The Hippie Parade: My favorite stories lined up one right after
    anoth, [insert imaginative title here]: , Lyf Is Unpredictable: , ]

 k= 20[jobersaudara: , That's Life: Blah blah blog..., One Day At A Time...: "Faith sees the
     invisible, believes the unbelievab, i like banjos: nothing at all about banjos.
    mostly rambling abou, Jean Bunner's House of cards!: My blog about Real Estate and Card
     Making, Apifera Farm: where animals and art collide. Home to Katherine Dunn/artist:
    She baked him a pie, and love was zizzling inside , Manepipoca Prog: Jazz−Rock / Prog,
    Brittany vs. Utah: , Simply Me Art: Simplymeart.etsy.com, .: , ]
[Jody L Meyer: Loving Life!  It's the theme song of my heart... S, lifeintrees: Hurtling
    towards greatness., globallistic: What I listen to when the voices subside (and how , ]
[Words that flow...: Words and images − powerful elements of our everyd, Kamiel Choi: , When
     Two Hearts Become One: A wife, a mom, a daughter, a sibling.  Corporate s, Urban
    Anatomy's Columbian Jungle (That Dope!): , Difficult Music: Chaos is the future and
    beyond it is freedom., Amyzing: , Lindsay Vicious: another fabricated self portrait, ]
[JAYWAT:  Presently Past the Future: âĂIJNow I can let these dream killers kill my self es,
    On An Overgrown Path: , ]
[it's all too much: of a muchness for me., Emily Arnold: , THE FLOYD MELTON FAMILY: THE
    FLOYD MELTON FAMILY Floyd III, Lisa, Floyd IV,, It's My Life: , Scraps & Sparklz: ,
    Broken American: , ]
[The Start of a New Day: , THE TWO CRAZIES AND TWO LADIES: QUEEN KING PRINCE PRINCESS, F−
    Measure: Less Than Timely Music Reviews and Commentary., Infidelia: Une Princesse
    Guerriere: An attempt to avoid the maudlin.  To think of absu, ]
[The Family Lacy: , Neena's Nest: Most people are just about as happy as they make u, The
    Woolfpack Six: The life &amp;amp; times of two adults and four kids.., Stevenson Common
    Threads: &amp;quot;There's a common thread, that's between your, ...: zero pride and even
    less shame, The Williams Five: This is about us.  Life in our house with 3 kids c, Web
    Science and Digital Libraries Research Group: Research and Teaching Updates from the Web
     Science, Mutiny on the Cardboard Sea: , The Hippie Parade: My favorite stories lined up
     one right after anoth, The Frixen Family: , ]
[The Baron Boombox: , DAWSON FAMILY DETAILS: , Vargo Family News: , WILSON: , Rothenberger~
    Palooza: , The Fucking Bike Club: Spokane, WA, ]
[Dreaming Out Loud: ..The story of my life as its told.., The Balow Bunch: My thoughts on
    life, its lessons, my husband and p, ELVINA GOH: , God is good: Ramblings and musings
    about life, family, work and, SkybellArts @ Blogger: "Practice, practice, practice", An
    American Salsera in Barcelona: Tengo Master en Parranda, OCCASIONAL BLOG: Occasional
    Blog (where ideas are left to fend for, Journey to the Simple Life: This is my journey
    toward a more simple life.  One, The Stormblog: , Life with Ana, Maddie, & Mia...: ...
    and a boy they call Daddy., ]
[EZHEVIKA FIELDS: , The STA Official Label Blog Of Thoughts And StuffÂő: Happily Independent
     Since 2001, ]
[Ickleson illustration + design: amongst other things, Exciting Life of CPA's: , The Clarks
    in Austin: , the world according to tim.: We want your sympathy vote., ]
[Kasey Buick: one girl talks, Journals, Reviews and Stories: A collection of writings that
    mirror my thoughts a, Rachel.C: The Road To Publication by Author Rachel Charlton, ]
[Home of the Eekers: <br /><br /><br /><br /><br /><br /><br /><b, Pining for
    Nordstrom: ...a Materialist Misplaced in a Nordstrom−less Tow, Jumble Sale Rabbits:
    Elizabeth Gunby, The sport of the arts...: We are what we repeatedly do.  Excellence,
    therefo, ]
[Michele's Treasures, Teacups, & Tumbling Rose Cottage: A chat over tea about things
    feminine, romantic, a, Pull up a chair...enjoy the fire.: , Sunshine's Family Blog: Look
     for life's sunshine and you'll find it!!!, Sell...Party of 5: , ]
[−.: , DJ DHANIEL FAN AND PRODUCER: MUSICAS VIDEOS, Living Diagonally In My Parallel World!:
     Best viewed on Mozilla Firefox, 11 Smiths: , The Rants of a dude that lacks blog−naming
    5K1||z: , In The Middle ~ Mad Hatter: A place to be "ME", ]
```

```
[bhairo in the morning: , Mishegas Master: My journal of life and those lives that surround
    &, ]
[king ton: Generating new words and gathering parts of the la, Ben Rayner: , Edge of Sanity:
     "There is no real me: only an entity, something il, You Breed Like Rats: , Touch My
    Clickwheel: A blog dedicated to Touch My Clickwheel, formerly , [insert imaginative
    title here]: , Hot Rock With Me: musica, we're building castles in the sky..: , ]
[The Dirty Schnee: Life on a mountain...., My Cotton Candy World: , beta blog: "If you think
     everything is all right, you're just, ]
[Am I a funny girl?: I think I have my moments., the yogurt chronicles: 90% of my stories
    are 100% accurate., ]
[the 21 gun salute!: , puzzle pieces: , Motivational Morning Quotes,  Famous Inspirational
    Thoughts, Famous Self Help Sentences: Very Good Morning. Read daily inspirational self h
    , Lyf Is Unpredictable: , ]
```

Listing 19: output of kclustering algorithm

Problem 4

Question

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

Answer

With the code in Listing 20, multidimensional scaling (MDS) was used to create a two-dimensional visualization of the blog distance graph. This code calls the `scaledown` function, which is shown in Listing 21. The algorithm continues until the error factor stops decreasing, as shown in the output in Listing 23.

```
    coords=scaledown(data)
    draw2d(coords, blognames, jpeg='blogs2d.jpg')
```

Listing 20: main for scaledown

```
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
        if realdist[j][k] != 0:
            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
```

Listing 21: scaledown function

The `scaledown` function returns the coordinates for each of the blogs in 2D space. This data was then used with the `draw2d` function in Listing 22, which produced the two-dimensional visualation created from the MDS algorithm, as shown in Figure 2.

```
def draw2d(data,labels,jpeg='mds2d.jpg'):
  img=Image.new('RGB',(2000,2000),(255,255,255))
  draw=ImageDraw.Draw(img)
  for i in range(len(data)):
    x=(data[i][0]+0.5)*1000
    y=(data[i][1]+0.5)*1000
    draw.text((x,y),labels[i],(0,0,0))
  img.save(jpeg,'JPEG')
```

Listing 22: draw2d function

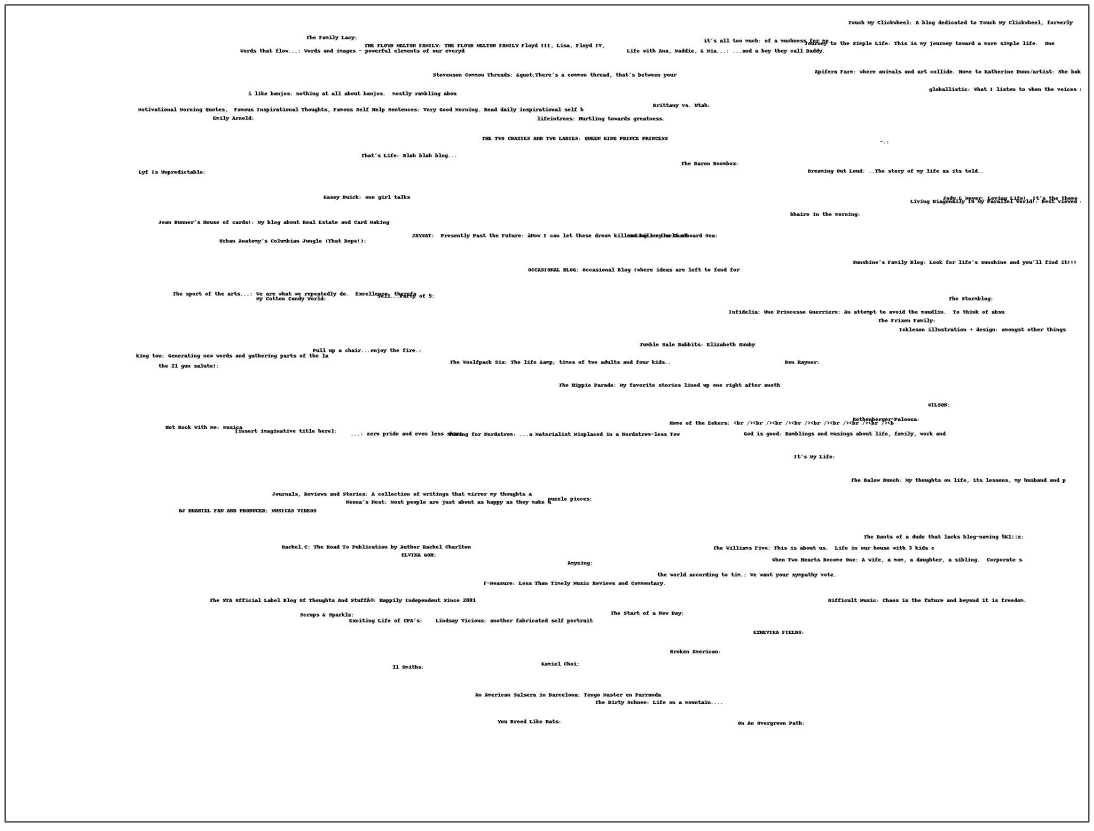

Figure 2: MDS 2d visualization

```
4847.00421927
3435.83928265
3413.33660099
3404.5521876
3399.12677068
3394.80997431
3391.42612501
3388.4842429
3385.74268499
3383.05623689
3380.61738929
3378.33371082
3375.94437389
3375.2807006
3375.8394578
```

Listing 23: scaledown output

Problem 5

Question

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.

Answer

To answer this question, `matrix.py` was modified to add the capability to calculate *Term Frequency Inverse Document Frequency (TF/IDF)*. The added functions for computing TF/IDF are found in Listing 26. These functions use the master word count dictionary (`wordcounts`) and each blog's individual word count (`wc`) for each of the words in the `wordlist` from Question 1/2 to compute the TF/IDF value for each word in the word list.

```
        elif len(sys.argv) == 2 and sys.argv[1] == 'tfidf':
            write_data('blogdata2.txt', wordlist, wordcounts, form=lambda wc, word,
                wordcounts: tf(wc, word) * idf(wordcounts, word))
```

Listing 24: use of tfidf function

```
def write_data(filename, wordlist, wordcounts, form=lambda wc, word, wordcounts: wc[word]):
    with open(filename, 'w') as out:
        out.write('Blog')
        for word in wordlist[:500]:
            out.write('\t%s' % word)
        out.write('\n')
        for blog, wc in wordcounts.items():
            print blog
            out.write(blog)
            for word in wordlist[:500]:
                if word in wc:
                    out.write('\t{}'.format(form(wc,word,wordcounts)))
                else:
                    out.write('\t0')
            out.write('\n')
```

Listing 25: writing the data

```
def tf(wc, word):
    return float(wc[word]) / float(sum(wc.values()))

def idf(wordcounts, word):
    present = 0
    for wc in wordcounts.values():
        if word in wc:
            present += 1
    return math.log(len(wordcounts) / present, 2)
```

Listing 26: tf idf functions

The same clustering was applied to the TF/IDF result matrix as was done in Question 2 and both images are displayed in Figures 3 and 4.

There are a many pairs that were found to be similar in both clusterings. For example, in both dendrograms, the *Web Science and Digitial Libraries Research Group* blog is most similar to the *...: zero pride and even less shame* blog, the *DJ DHANIEL FAN AND PRODUCER* is paired with *The Baron Boombox*, among a few others. In spite of this, the larger groupings do not appear very similar between the two clustings. There are some groups that share blogs in both dendrograms, but this is mostly due to there being only a few distinct groups in the raw count version and many of the TF/IDF clusters seemingly being subsets of the fewer, larger clustering from the raw count dendrogram.

When examining each of the clusters on a larger scale, it seems that the TF/IDF dendrogram clustered blogs are subjectively more alike than the raw count clusters. Looking toward the bottom of the TF/IDF clustering image, one will notice a grouping of blogs that seem closely related to music: *F-Measure: Less Than timely Music Reviews and Commentary.*, *Urban Anatomy's Columbian Jungle (That Dope!)* which seems to be a melding of fashion and music commentary, *Ezhevika Fields* a blog where info and preview samples of "lost album samples of the past" can be found, whereas these blogs are not all grouped together in the raw count dendrogram.

There is another cluster in the TF/IDF driven image that seems to contain family related blogs, with blogs like *Am I a funny girl?: I think I have my moments* which is, *The Frixen Family*, *When Two Hearts Become One: A wife, a mom, a daughter, a sibling.* and *The Clarks in Austin* all being related to a particular family and their everyday lives. Some of these blogs are close to each other in the raw count version, but they are not separated into their own distinct groups. The groups in the raw count version share some of these blogs, but are also mixed with others that seem unrelated. For example, the top cluster in the raw count dendrogram doesn't seem to have much of a unifying subject at all.

When looking at the overall structure of the two dendrograms it becomes apparent that there are more individual clusters grouped together in the TF/IDF version than are present in the raw count image, where there seems to be few small clusters and one mega-cluster in the center. This suggests that the TF/IDF algorithm was better at defining discrete subgroups within the larger context than the simple raw count dendrogram produced.

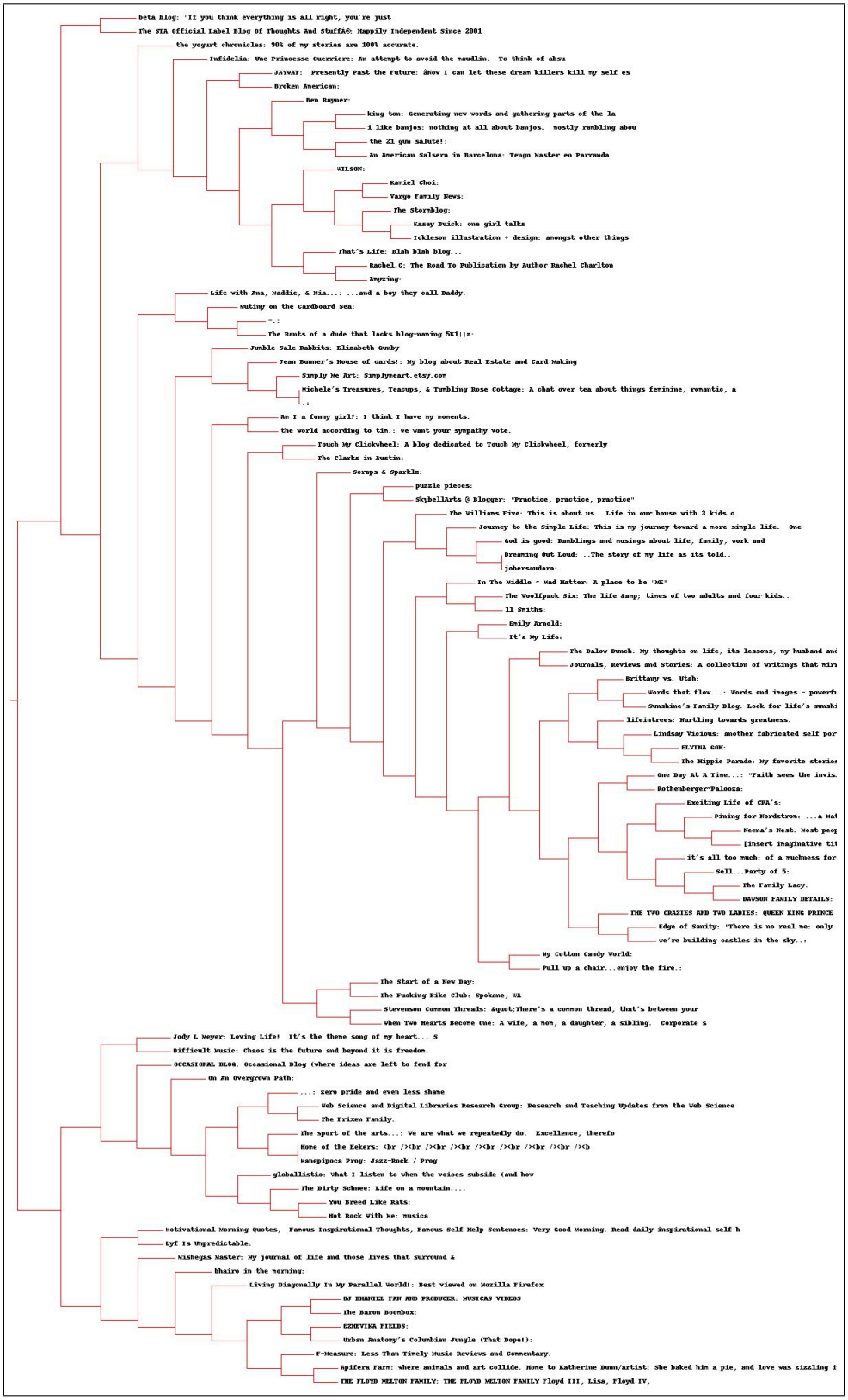

Figure 3: raw count dendrogram

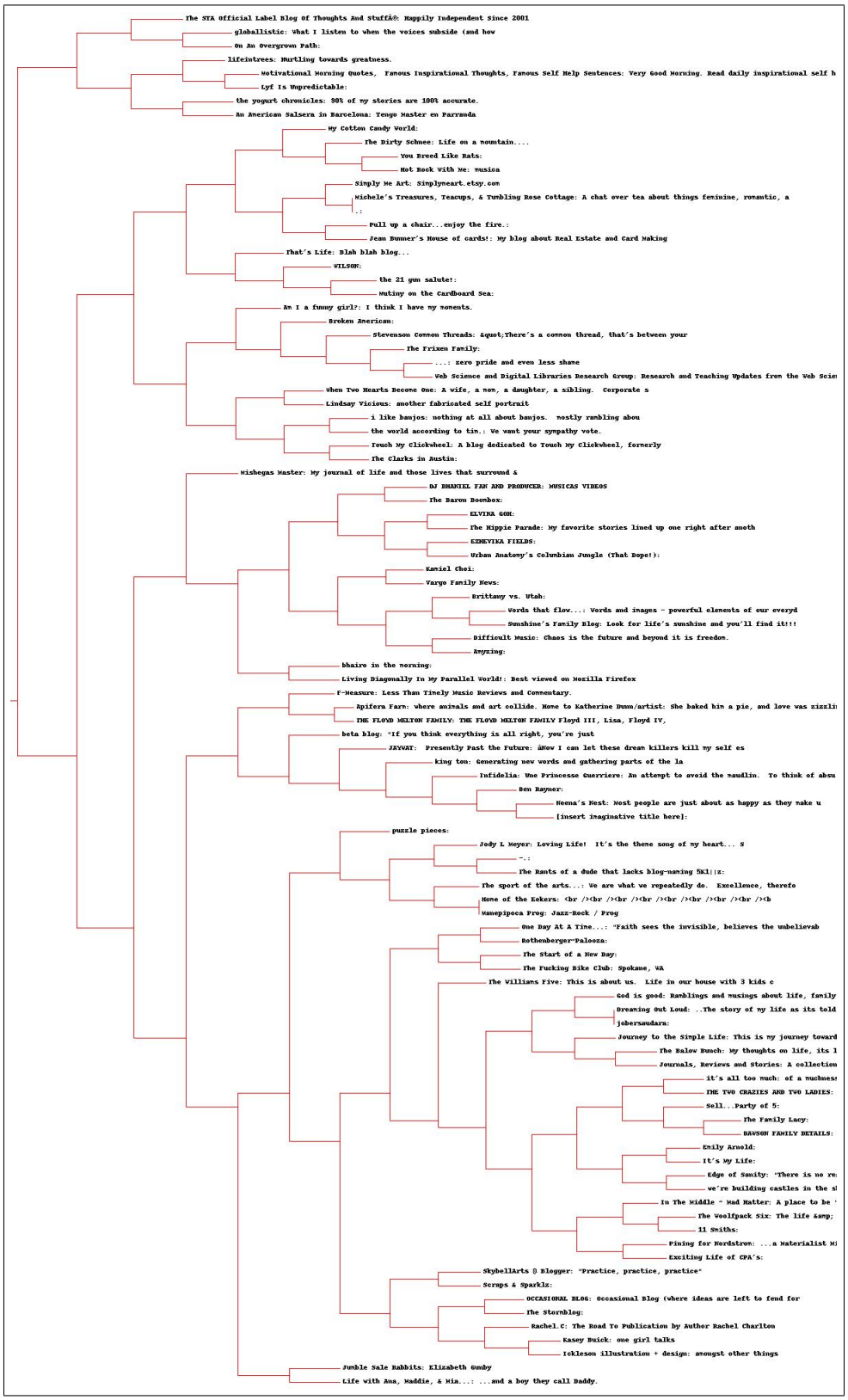

Figure 4: TF/IDF dendrogram

Appendix A

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

import requests
import sys
from bs4 import BeautifulSoup

default = 'http://www.blogger.com/next-blog?navBar=true&blogID=3471633091411211117'
must_haves = ['http://f-measure.blogspot.com/', 'http://ws-dl.blogspot.com/']

def get_atom(uri):
    try:
        r = requests.get(uri)
    except Exception, e:
        return None
    soup = BeautifulSoup(r.text)
    links = soup.find_all('link', {'type':'application/atom+xml'})
    if links:
        return str(links[0]['href'])
    return None

def add_uri(uri, uris, outfile):
    if uri and uri not in uris:
        uris.add(uri)
        outfile.write(uri + '\n')
        print len(uris), uri

if __name__ == '__main__':
    uris = set()
    with open('blog_uris', 'a') as outfile:
        if len(sys.argv) > 1 and sys.argv[1] == 'new':
            for must_have in must_haves:
                uri = get_atom(must_have)
                add_uri(uri, uris, outfile)
        else:
            with open('blog_uris') as infile:
                [uris.add(line.strip()) for line in infile]
        while len(uris) < 100:
            uri = get_atom(default)
            add_uri(uri, uris, outfile)
```

Listing 27: get_uris.py

```
import feedparser
import futures
import math
import md5
import re
import sys
import json

def get_next(d):
    for item in d.feed.links:
        if item['rel'] == u'next':
            return item['href']
    return None

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

def get_titles(uri):
    print('processing {}'.format(uri))
    next = uri
    wc = {}
    pages = 0
    while next is not None:
        d = feedparser.parse(next)
        for e in d.entries:
            words = get_words(e.title.encode('utf-8'))
            for word in words:
                wc.setdefault(word, 0)
                wc[word] += 1
        pages += 1
        next = get_next(d)
        print('next {}'.format(next))
    title = d.feed.title.encode('utf-8')
    subtitle = d.feed.subtitle[:50].encode('utf-8')
    print('finished: {}: {}'.format(title, subtitle))
    return uri, title, subtitle, pages, wc

def tf(wc, word):
    return float(wc[word]) / float(sum(wc.values()))

def idf(wordcounts, word):
    present = 0
    for wc in wordcounts.values():
        if word in wc:
            present += 1
    return math.log(len(wordcounts) / present, 2)

def load_data(uris):
    apcount = {}
    wordcounts = {}
    pagecounts = {}
    for uri in uris:
        with open('wcs/' + md5.new(uri).hexdigest()) as infile:
            try:
                lines = infile.read().split('\t')
                title = lines[0]
                pages = int(lines[1])
                wc = json.loads(lines[2])
            except Exception, e:
                print('*** {} generated an exception: {}'.format(uri, e))
                continue
        wordcounts[title] = wc
        pagecounts[title] = pages
        for word, count in wc.items():
            apcount.setdefault(word, 0)
```

```
            apcount[word] += count
    return apcount, wordcounts, pagecounts

def build_wordlist(apcount, uris):
    wordlist = []
    for w, bc in sorted(apcount.items(), key=lambda x: x[1], reverse=True):
            frac = float(bc) / len(uris)
            if frac > 0.1 and frac < 0.5:
                wordlist.append(w)
    return wordlist

def write_data(filename, wordlist, wordcounts, form=lambda wc, word, wordcounts: wc[word]):
    with open(filename, 'w') as out:
        out.write('Blog')
        for word in wordlist[:500]:
            out.write('\t%s' % word)
        out.write('\n')
        for blog, wc in wordcounts.items():
            print blog
            out.write(blog)
            for word in wordlist[:500]:
                if word in wc:
                    out.write('\t{}'.format(form(wc,word,wordcounts)))
                else:
                    out.write('\t0')
            out.write('\n')

if __name__ == '__main__':
    with open('blog_uris') as infile:
        uris = [line.strip() for line in infile if line.strip()]
    if len(sys.argv) == 2 and sys.argv[1] == 'get':
        with futures.ThreadPoolExecutor(max_workers=8) as executor:
            uri_futures = [executor.submit(get_titles, uri) for uri in uris]
            for future in futures.as_completed(uri_futures):
                uri, title, subtitle, pages, wc = future.result()
                with open('wcs/' + md5.new(uri).hexdigest(), 'w') as out:
                    out.write(title + ': ' + subtitle + '\t' + str(pages) + '\t')
                    json.dump(wc, out)
    else:
        apcount, wordcounts, pagecounts = load_data(uris)
        wordlist = build_wordlist(apcount, uris)
        if len(sys.argv) == 2 and sys.argv[1] == 'pages':
            with open('pagecounts', 'w') as outfile:
                outfile.write('blog\tpages\n')
                for blog, pagecount in pagecounts.iteritems():
                    outfile.write("\"" + blog.replace("\"", "") + "\"" + '\t' + str(
                        pagecount) + '\n')
        elif len(sys.argv) == 2 and sys.argv[1] == 'wc':
            write_data('blogdata1.txt', wordlist, wordcounts)
        elif len(sys.argv) == 2 and sys.argv[1] == 'tfidf':
            write_data('blogdata2.txt', wordlist, wordcounts, form=lambda wc, word,
                wordcounts: tf(wc, word) * idf(wordcounts, word))
```

Listing 28: matrix.py

```
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)]
```

```
68          if d<closest:
              closest=d
              lowestpair=(i,j)

      # calculate the average of the two clusters
73    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
78    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
83    currentclustid-=1
      del clust[lowestpair[1]]
      del clust[lowestpair[0]]
      clust.append(newcluster)

88  return clust[0]

def printclust(clust,labels=None,n=0):
  # indent to make a hierarchy layout
  for i in range(n): print ' ',
93  if clust.id<0:
    # negative id means that this is branch
    print '-'
  else:
    # positive id means that this is an endpoint
98    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)
103 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
108
  # Otherwise the height is the same of the heights of
  # each branch
  return getheight(clust.left)+getheight(clust.right)

113 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
118 # plus its own distance
  return max(getdepth(clust.left),getdepth(clust.right))+clust.distance


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

128 # 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))
133 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
        if realdist[j][k] != 0:
            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

def draw2d(data,labels,jpeg='mds2d.jpg'):
  img=Image.new('RGB',(2000,2000),(255,255,255))
  draw=ImageDraw.Draw(img)
  for i in range(len(data)):
    x=(data[i][0]+0.5)*1000
    y=(data[i][1]+0.5)*1000
    draw.text((x,y),labels[i],(0,0,0))
  img.save(jpeg,'JPEG')

import sys

if __name__ == '__main__':
    blognames, words, data = readfile('q1/blogdata1.txt')
    clust = hcluster(data)
    with open('dendrogram.txt', 'w') as outfile:
        stdout = sys.stdout
        sys.stdout = outfile
        printclust(clust, labels=blognames)
        sys.stdout = stdout
    drawdendrogram(clust, blognames, jpeg='blogclust.jpg')
    print "Done with dendrograms"
    outfile=open('kclust.txt','w')
    for i in [5,10,20] :
        kclust,iternum=kcluster(data,k=i)
        outfile.write('\n\n k = %d'%i)
        outfile.write('Iterations = %d\n'%iternum)
        for cluster in kclust:
            outfile.write('[')
            for blogidx in cluster:
                outfile.write(blognames[blogidx]+', ')
            outfile.write(']\n')
    outfile.close()
    coords=scaledown(data)
    draw2d(coords, blognames, jpeg='blogs2d.jpg')
```

Listing 29: clusters.py

References

[1] Internet Engineering Task Force (IETF). Rfc-4287 the atom syndication format. https://tools.ietf.org/htmlrfc4287, 2016.

[2] Toby Segaran. Programming collective intelligence. oâĂŹreilly, first edition„ 2007.