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 17. 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

```python
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

```python
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

```python
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 18.

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 17), 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.

```python
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 11 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
        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)]

  # 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
```

Listing 13: creating the dendrograms

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

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

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

    # loop through every pair looking for the smallest distance
    for i in range(len(clust)):
      for j in range(i+1,len(clust)):
        # distances is the cache of distance calculations
        if (clust[i].id,clust[j].id) not in distances:
          distances[(clust[i].id,clust[j].id)]=distance(clust[i].vec,clust[j].vec)

        d=distances[(clust[i].id,clust[j].id)]

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

    # calculate the average of the two clusters
    mergevec=[
    (clust[lowestpair[0]].vec[i]+clust[lowestpair[1]].vec[i])/2.0
    for i in range(len(clust[0].vec))]

    # create the new cluster
    newcluster=bicluster(mergevec,left=clust[lowestpair[0]],
                         right=clust[lowestpair[1]],
                         distance=closest,id=currentclustid)

    # cluster ids that weren't in the original set are negative
    currentclustid-=1
    del clust[lowestpair[1]]
    del clust[lowestpair[0]]
    clust.append(newcluster)

  return clust[0]
```

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

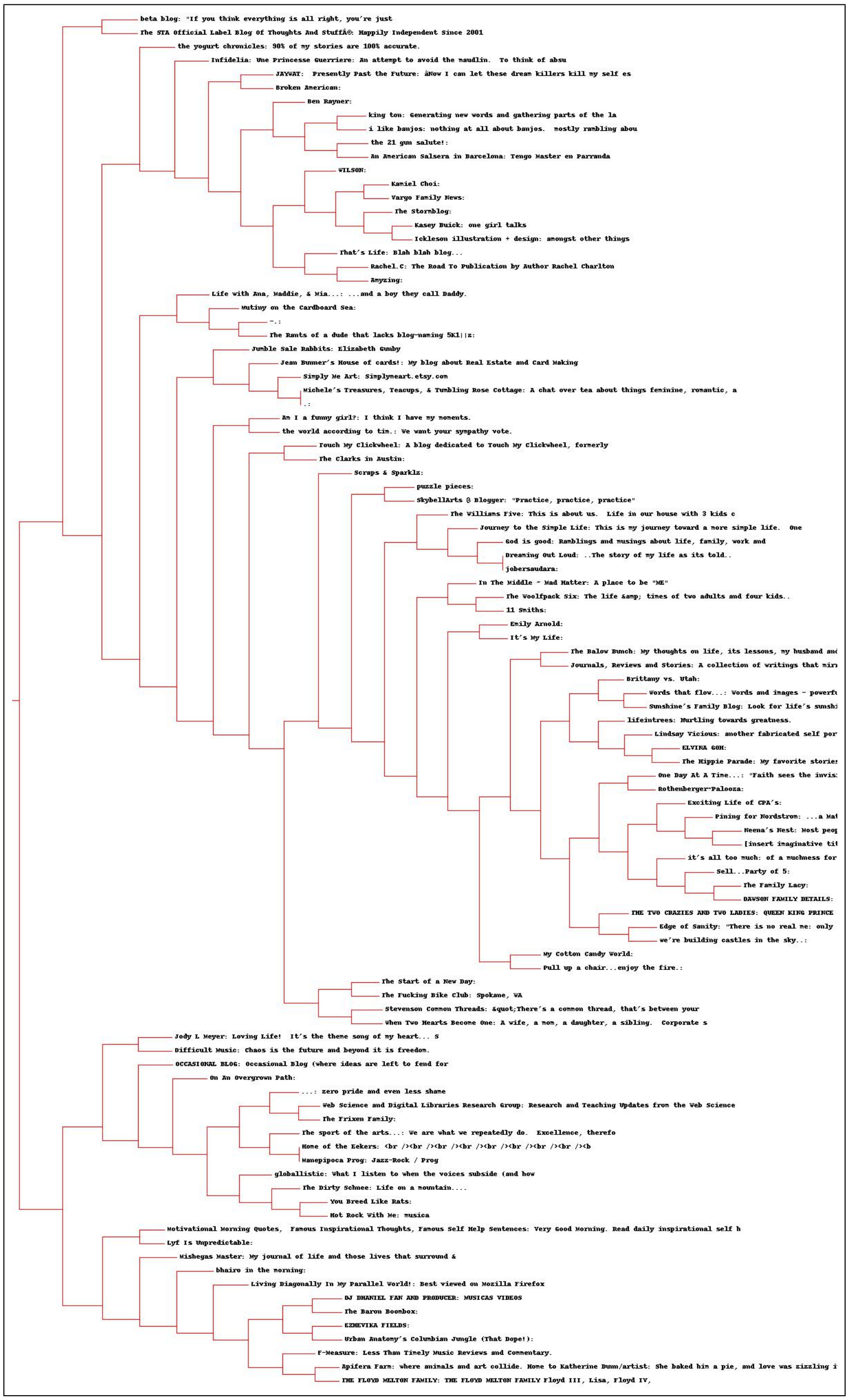
beta blog: "If you think everything is all right, you're just
The STA Official Label Blog Of Thoughts And StuffÂ®: Happily Independent Since 2001
the yogurt chronicles: 90% of my stories are 100% accurate.
Infidelia: Une Princesse Guerriere: An attempt to avoid the maudlin. To think of absu
JAYWAT: Presently Past the Future: âNow I can let these dream killers kill my self es
Broken American:
Ben Rayner:
king ton: Generating new words and gathering parts of the la
i like banjos: nothing at all about banjos. mostly rambling abou
the 21 gun salute!:
An American Salsera in Barcelona: Tengo Master en Parranda
WILSON:
Kamiel Choi:
Vargo Family News:
The Stormblog:
Kasey Buick: one girl talks
Ickleson illustration + design: amongst other things
That's Life: Blah blah blog...
Rachel.C: The Road To Publication by Author Rachel Charlton
Amyzing:
Life with Ana, Maddie, & Mia...: ...and a boy they call Daddy.
Mutiny on the Cardboard Sea:
~.:
The Rants of a dude that lacks blog-naming 5K1||z:
Jumble Sale Rabbits: Elizabeth Gumby
Jean Bunner's House of cards!: My blog about Real Estate and Card Making
Simply Me Art: Simplymeart.etsy.com
Michele's Treasures, Teacups, & Tumbling Rose Cottage: A chat over tea about things feminine, romantic, a
.:
Am I a funny girl?: I think I have my moments.
the world according to tim.: We want your sympathy vote.
Touch My Clickwheel: A blog dedicated to Touch My Clickwheel, formerly
The Clarks in Austin:
Scraps & Sparklz:
puzzle pieces:
SkybellArts @ Blogger: "Practice, practice, practice"
The Williams Five: This is about us. Life in our house with 3 kids c
Journey to the Simple Life: This is my journey toward a more simple life. One
God is good: Ramblings and musings about life, family, work and
Dreaming Out Loud: ..The story of my life as its told..
jobersaudara:
In The Middle ~ Mad Hatter: A place to be "ME"
The Woolfpack Six: The life &amp; times of two adults and four kids..
11 Smiths:
Emily Arnold:
It's My Life:
The Balow Bunch: My thoughts on life, its lessons, my husband anc
Journals, Reviews and Stories: A collection of writings that mir
Brittany vs. Utah:
Words that flow...: Words and images - powerfu
Sunshine's Family Blog: Look for life's sunshi
lifeintrees: Hurtling towards greatness.
Lindsay Vicious: another fabricated self por
ELVINA GOH:
The Hippie Parade: My favorite stories
One Day At A Time...: "Faith sees the invis:
Rothenberger~Palooza:
Exciting Life of CPA's:
Pining for Nordstrom: ...a Mat
Neena's Nest: Most peop
[insert imaginative tit
it's all too much: of a muchness for
Sell...Party of 5:
The Family Lacy:
DAWSON FAMILY DETAILS:
THE TWO CRAZIES AND TWO LADIES: QUEEN KING PRINCE
Edge of Sanity: "There is no real me: only
we're building castles in the sky..:
My Cotton Candy World:
Pull up a chair...enjoy the fire.:
The Start of a New Day:
The Fucking Bike Club: Spokane, WA
Stevenson Common Threads: &quot;There's a common thread, that's between your
When Two Hearts Become One: A wife, a mom, a daughter, a sibling. Corporate s
Jody L Meyer: Loving Life! It's the theme song of my heart... S
Difficult Music: Chaos is the future and beyond it is freedom.
OCCASIONAL BLOG: Occasional Blog (where ideas are left to fend for
On An Overgrown Path:
...: zero pride and even less shame
Web Science and Digital Libraries Research Group: Research and Teaching Updates from the Web Science
The Frixen Family:
The sport of the arts...: We are what we repeatedly do. Excellence, therefo
Home of the Eekers: <br /><br /><br /><br /><br /><br /><br /><br /><b
Manepipoca Prog: Jazz-Rock / Prog
globallistic: What I listen to when the voices subside (and how
The Dirty Schmee: Life on a mountain....
You Breed Like Rats:
Hot Rock With Me: musica
Motivational Morning Quotes, Famous Inspirational Thoughts, Famous Self Help Sentences: Very Good Morning. Read daily inspirational self h
Lyf Is Unpredictable:
Mishegas Master: My journal of life and those lives that surround &
bhairo in the morning:
Living Diagonally In My Parallel World!: Best viewed on Mozilla Firefox
DJ DHANIEL FAN AND PRODUCER: MUSICAS VIDEOS
The Baron Boombox:
EZHEVIKA FIELDS:
Urban Anatomy's Columbian Jungle (That Dope!):
F-Measure: Less Than Timely Music Reviews and Commentary.
Apifera Farm: where animals and art collide. Home to Katherine Dunn/artist: She baked him a pie, and love was zizzling i
THE FLOYD MELTON FAMILY: THE FLOYD MELTON FAMILY Floyd III, Lisa, Floyd IV,

Figure 1: blog dendrogram

Appendix A

```python
#! /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 17: 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 18: 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)]
        if d<closest:
          closest=d
          lowestpair=(i,j)

    # calculate the average of the two clusters
    mergevec=[
    (clust[lowestpair[0]].vec[i]+clust[lowestpair[1]].vec[i])/2.0
    for i in range(len(clust[0].vec))]

    # create the new cluster
    newcluster=bicluster(mergevec,left=clust[lowestpair[0]],
                         right=clust[lowestpair[1]],
                         distance=closest,id=currentclustid)

    # cluster ids that weren't in the original set are negative
    currentclustid-=1
    del clust[lowestpair[1]]
    del clust[lowestpair[0]]
    clust.append(newcluster)

  return clust[0]

def printclust(clust,labels=None,n=0):
  # indent to make a hierarchy layout
  for i in range(n): print ' ',
  if clust.id<0:
    # negative id means that this is branch
    print '-'
  else:
    # positive id means that this is an endpoint
    if labels==None: print clust.id
    else: print labels[clust.id]

  # now print the right and left branches
  if clust.left!=None: printclust(clust.left,labels=labels,n=n+1)
  if clust.right!=None: printclust(clust.right,labels=labels,n=n+1)

def getheight(clust):
  # Is this an endpoint? Then the height is just 1
  if clust.left==None and clust.right==None: return 1

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

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

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


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

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

  # Create a new image with a white background
  img=Image.new('RGB',(w,h),(255,255,255))
  draw=ImageDraw.Draw(img)

  draw.line((0,h/2,10,h/2),fill=(255,0,0))
  # Draw the first node
  drawnode(draw,clust,10,(h/2),scaling,labels)
  img.save(jpeg,'JPEG')

def drawnode(draw,clust,x,y,scaling,labels):
  if clust.id<0:
    h1=getheight(clust.left)*20
    h2=getheight(clust.right)*20
    top=y-(h1+h2)/2
    bottom=y+(h1+h2)/2
    # Line length
    ll=clust.distance*scaling
    # Vertical line from this cluster to children
    draw.line((x,top+h1/2,x,bottom-h2/2),fill=(255,0,0))

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

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

    # Call the function to draw the left and right nodes
    drawnode(draw,clust.left,x+ll,top+h1/2,scaling,labels)
    drawnode(draw,clust.right,x+ll,bottom-h2/2,scaling,labels)
  else:
    # If this is an endpoint, draw the item label
    draw.text((x+5,y-7),labels[clust.id],(0,0,0))

def rotatematrix(data):
  newdata=[]
  for i in range(len(data[0])):
    newrow=[data[j][i] for j in range(len(data))]
    newdata.append(newrow)
  return newdata

import random

def kcluster(rows,distance=pearson,k=4):
  # Determine the minimum and maximum values for each point
  ranges=[(min([row[i] for row in rows]),max([row[i] for row in rows]))
  for i in range(len(rows[0]))]

  # Create k randomly placed centroids
  clusters=[[random.random()*(ranges[i][1]-ranges[i][0])+ranges[i][0]
  for i in range(len(rows[0]))] for j in range(k)]

  lastmatches=None
  for t in range(100):
    print 'Iteration %d' % t
    bestmatches=[[] for i in range(k)]

    # Find which centroid is the closest for each row
    for j in range(len(rows)):
      row=rows[j]
      bestmatch=0
      for i in range(k):
        d=distance(clusters[i],row)
        if d<distance(clusters[bestmatch],row): bestmatch=i
      bestmatches[bestmatch].append(j)

    # If the results are the same as last time, this is complete
    if bestmatches==lastmatches: break
    lastmatches=bestmatches

    # Move the centroids to the average of their members
    for i in range(k):
      avgs=[0.0]*len(rows[0])
      if len(bestmatches[i])>0:
        for rowid in bestmatches[i]:
          for m in range(len(rows[rowid])):
            avgs[m]+=rows[rowid][m]
        for j in range(len(avgs)):
          avgs[j]/=len(bestmatches[i])
        clusters[i]=avgs

  return bestmatches

def tanamoto(v1,v2):
  c1,c2,shr=0,0,0

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

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

def scaledown(data,distance=pearson,rate=0.01):
  n=len(data)

  # The real distances between every pair of items
  realdist=[[distance(data[i],data[j]) for j in range(n)]
             for i in range(0,n)]

  # Randomly initialize the starting points of the locations in 2D
  loc=[[random.random(),random.random()] for i in range(n)]
  fakedist=[[0.0 for j in range(n)] for i in range(n)]

  lasterror=None
  for m in range(0,1000):
    # Find projected distances
    for i in range(n):
      for j in range(n):
        fakedist[i][j]=sqrt(sum([pow(loc[i][x]-loc[j][x],2)
                                 for x in range(len(loc[i]))]))

    # Move points
    grad=[[0.0,0.0] for i in range(n)]

    totalerror=0
    for k in range(n):
      for j in range(n):
        if j==k: continue
        # The error is percent difference between the distances
        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 19: 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.