Assignment 8

Fall 2014
CS595 Web Science
Dr. Michael Nelson

Mathew Chaney

November 29, 2014

Contents

Listings

1 Question 1

1.1 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.

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.

Create a histogram of how many pages each blog has (e.g., 30
blogs with just one page, 27 with two pages, 29 with 3 pages and
so on).
```

1.2 Answer

To complete this assignment, a blog word count dataset 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. Two default blogs, F-Measure and the Old Dominion Web Science and Digital Libraries blogs, were added to the list and then, using the seed URI provided (Listing 2), the remaining 98 URIs from random blogs within the blogger.com family were added. Then, using the `matrix.py` script, the page counts for each blog were extracted and saved to a file called `pagecounts`. The `matrix.py` script is a modified version of `generatefeedvectors.py` from the book *Programming Collective Intelligence* [1].

```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 1: main for get_uris.py

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

Listing 2: referenced variables in get_uris.py

The `get_uris main` function in Listing 1 was the driver that called the `get_atom` function (shown in Listing 3) to extract the atom [2] URIs from each blog and add them to the set of URIs with the `add_uri` function, shown in Listing 4.

```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 3: 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 4: add_uri function

The contents of each blog were downloaded and processed by the code shown in Listing 5 and the `get_titles`, `get_words` and `get_next` functions found in Listing 6. This code loops over the URIs that were downloaded with the `get_uris.py` script, parses each entry and extracts all the words in each entry's title. These words were then compiled into a master list for all 100 blogs, with the top 500 words that fit into the range bounded by the code in Listing 7 being used for the final word count.

```python
    with open('blog_uris') as infile:
        uris = [line.strip() for line in infile]
    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 5: 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 getwords(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 = getwords(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 6: processing each blog

```
        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)
```

Listing 7: bounding the terms

To build a histogram showing the blog page counts, the `pagecounts` file was parsed by the R script in Listing 8 and saved as a pdf, which is shown in Figure 2.

```
#! /usr/bin/Rscript

data <- read.table("pagecounts", sep="\t", header=TRUE, comment.char="")
counts <- table(data$pages)
pdf("hist.pdf")
barplot(counts, ylab="Number of Blogs", xlab="Page Count", main="Page Count per Blog")
dev.off()
```

Listing 8: building the histogram

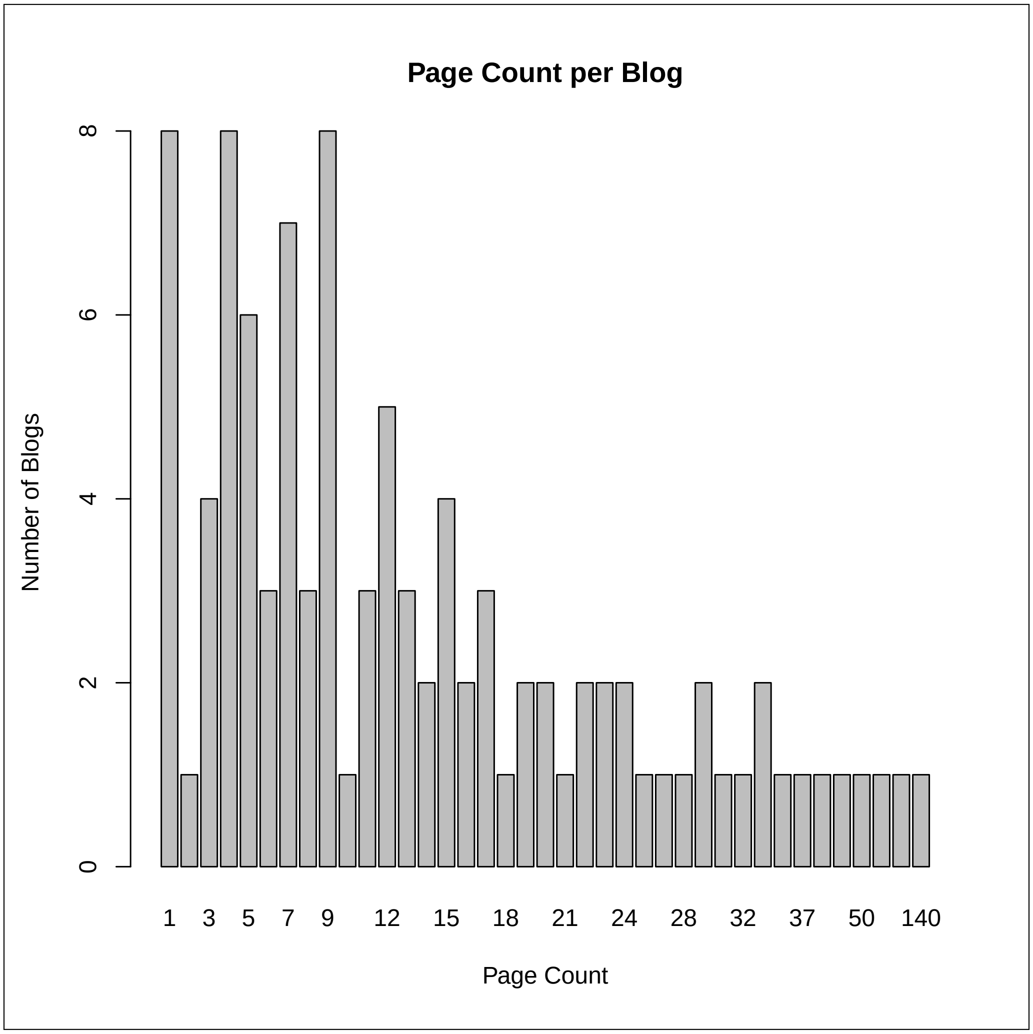

Figure 1: Page Count per Blog

2 Question 2

2.1 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).

2.2 Answer

The ascii and jpeg dendrograms were created using the method shown in Listing 9, which is modeled after the example from class.

```
    blognames, words, data = readfile('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 9: creating the dendrograms

This uses the `readfile` function shown in Listing 10 to read the data that was compiled from Question 1 into the script where it is then processed by the `hcluster` function found in Listing 11 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 10: 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 11: hcluster function

The `printclust` function from Listing 12 prints the ascii dendrogram of the cluster object parameter.

```
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 12: printclust function

The `drawdendrogram` function from Listing 13 creates a jpeg image of the cluster, which is shown in Figure ??.

```
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 13: drawdendrogram function

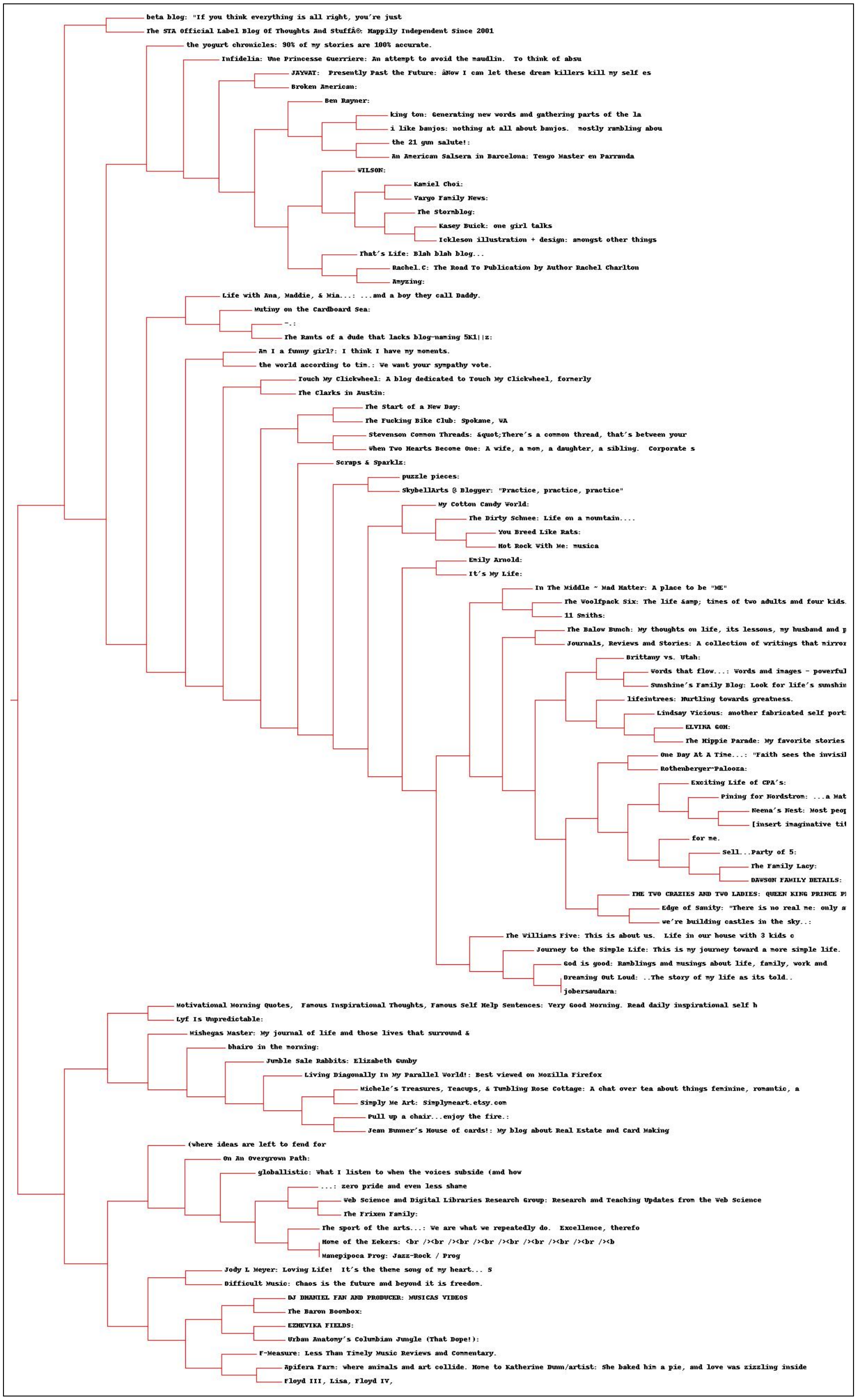
beta blog: "If you think everything is all right, you're just
The STA Official Label Blog Of Thoughts And StuffÂ®: Happily Independent Since 2001
the yogurt chronicles: 90% of my stories are 100% accurate.
Infidelia: Une Princesse Guerriere: An attempt to avoid the maudlin. To think of absu
JAYWAT: Presently 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:
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:
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
Scraps & Sparklz:
puzzle pieces:
SkybellArts @ Blogger: "Practice, practice, practice"
My Cotton Candy World:
The Dirty Schmee: Life on a mountain....
You Breed Like Rats:
Hot Rock With Me: musica
Emily Arnold:
It's My Life:
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:
The Balow Bunch: My thoughts on life, its lessons, my husband and p
Journals, Reviews and Stories: A collection of writings that mirror
Brittany vs. Utah:
Words that flow...: Words and images - powerful
Sunshine's Family Blog: Look for life's sunshin
lifeintrees: Hurtling towards greatness.
Lindsay Vicious: another fabricated self port:
ELVINA GOH:
The Hippie Parade: My favorite stories
One Day At A Time...: "Faith sees the invisil
Rothenberger~Palooza:
Exciting Life of CPA's:
Pining for Nordstrom: ...a Mat
Neena's Nest: Most peop
[insert imaginative tit
for me.
Sell...Party of 5:
The Family Lacy:
DAWSON FAMILY DETAILS:
THE TWO CRAZIES AND TWO LADIES: QUEEN KING PRINCE PI
Edge of Sanity: "There is no real me: only a
we're building castles in the sky..:
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.
God is good: Ramblings and musings about life, family, work and
Dreaming Out Loud: ..The story of my life as its told..
jobersaudara:
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:
Jumble Sale Rabbits: Elizabeth Gumby
Living Diagonally In My Parallel World!: Best viewed on Mozilla Firefox
Michele's Treasures, Teacups, & Tumbling Rose Cottage: A chat over tea about things feminine, romantic, a
Simply Me Art: Simplymeart.etsy.com
Pull up a chair...enjoy the fire.:
Jean Bunner's House of cards!: My blog about Real Estate and Card Making
(where ideas are left to fend for
On An Overgrown Path:
globallistic: What I listen to when the voices subside (and how
...: 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
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.
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 inside
Floyd III, Lisa, Floyd IV,

Figure 2: dendrogram

3 Question 3

3.1 Question

```
Cluster the blogs using K-Means, using k=5,10,20. (see slide 18).
How many interations were required for each value of k?
```

3.2 Answer

4 Question 4

4.1 Question

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

4.2 Answer

5 Question 5

5.1 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.

5.2 Answer

6 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 14: get_uris.py

```python
import feedparser
import futures
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 getwords(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 = getwords(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

if __name__ == '__main__':
    with open('blog_uris') as infile:
        uris = [line.strip() for line in infile]
    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 = {}
        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
        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)
        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')
        if len(sys.argv) == 2 and sys.argv[1] == 'wc':
            with open('blogdata1.txt', '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(wc[word]))
                        else: out.write('\t0')
                    out.write('\n')
```

Listing 15: matrix.py

7 References

[1] Toby Segaran. *Programming Collective Intelligence*. O'Reilly, first edition, 2007.

[2] Internet Engineering Task Force (IETF). RFC-4287 The Atom Syndication Format. https://tools.ietf.org/html/rfc4287, 2005.