

OLD DOMINION UNIVERSITY

CS 495: INTRODUCTION TO WEB SCIENCE
INSTRUCTOR: MICHAEL L. NELSON, PH.D
FALL 2014 4:20PM - 7:10PM R, ECSB 2120

Assignment # 9

GEORGE C. MICROS UIN: 00757376

Honor Pledge

I pledge to support the Honor System of Old Dominion University. I will refrain from any form of academic dishonesty or deception, such as cheating or plagiarism. I am aware that as a member of the academic community it is my responsibility to turn in all suspected violations of the Honor Code. I will report to a hearing if summoned.

Signed _____
December 4, 2014

George C. Micros

Written Assignment 9

Fall 2014

CS 495: Introduction to Web Science

Dr. Michael Nelson

December 4, 2014

Contents

1	Written Assignment 9	5
1.1	Question 1	6
1.1.1	The Question	6
1.1.2	The Answer	6
1.2	Question 2	11
1.2.1	The Question	11
1.2.2	The Answer	11
1.3	Question 3	13
1.3.1	The Question	13
1.3.2	The Answer	13
1.4	Question 4	15
1.4.1	The Question	15
1.4.2	The Answer	15
1.5	Extra Credit Question	17
1.5.1	The Question	17
1.5.2	The Answer	17
	References	21

Chapter 1
Written Assignment 9

1.1 Question 1

1.1.1 The 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).

1.1.2 The Answer

RSS and Atom feeds are an XML-based syndication format that are used to organize all blogs so that they can be easily be broken down into posts without the need for complicated parsing. The link to the pages containing the RSS or Atom feed can be typically found in the webpage of the blog itself. This is the link that contains all the blog posts and will be used to create the blog matrix data.

The code provided in the book [1] contains many useful python functions and a script that generates a feed vector based on a list of feed URLs. However, the links provided as input must be to the RSS/Atom feed itself, rather than the blog. In addition, paged blog feeds must be provided in separate links for each page. Therefore there is some preconditioning that must be done to obtain the data necessary to provide to this script.

The assignment specifies that of the 100 blog posts the following must be included:

<http://f-measure.blogspot.com/>
<http://ws-dl.blogspot.com/>

and 98 other blogs from blogspot. In practice more than 100, approximately 120, blogs were collected to prevent from falling under the required 100 if errors in extracting the feed occurred. The most common error presented was that certain blog required the reader to be logged in with their Google account, which made programmatically retrieving the feed more complicated. Such blogs occurred consistently at 2

However, the remaining blogs must be selected at random from blogspot. This is addressed by using the link

<https://www.blogger.com/next-blog?navBar=true&blogID=8145271598519191285>

that is requested when the “Next” button is pressed at the top of any blog. This automatically generates a new blog as a response to the request. Therefore, by making repeated requests to this URL the response received will always be different and random, for the most part.

The “Next” button allows us to collect the URLs of blogs, but the main objective is to collect RSS/Atom feed URL. The response html was parsed using BeautifulSoup and the link to the Atom feed extracted, the Atom feed seemed easier to find the paged link. Once the first link was extracted a request was made to that URL and from the response the next page link was extracted. This process was repeated until all the pages of the feed were collected. The links were all appended to a file that would be feed to the code provided by the book [1]. A comma separated file was created that contain the title and number of pages each blog contained in it. This is used later to produce the histogram of feed pages.

```

1 #! /usr/bin/python
2 import urllib2
3 from bs4 import BeautifulSoup as bs
4
5 # GET A REQUIRED NELSON BLOGS # RETURNS TITLE AND FEED PAGE LIST
6 def getBlogInfoInit():
7     Blogs = ["http://f-measure.blogspot.com/",
8             "http://ws-dl.blogspot.com/"]
9     ttl = []; feedLinks = []
10    for nextBlog in Blogs:

```

```

11     temp = []
12     try:
13         html = urllib2.urlopen(nextBlog).read()
14         soup = bs(html)
15         ttl.append(soup.title.string.encode('ascii'))
16         rss = soup.find('link', type='application/atom+xml')
17         if(rss != []):
18             rss = rss.get('href')
19             temp = allPages(rss)
20             temp.insert(0,rss)
21             feedLinks.append(temp)
22     except:
23         pass
24     return ttl, feedLinks
25 # GET A RANDOM BLOG, EXTRACT TITLE AND FEED PAGES # RETURNS TITLE AND FEED PAGE LIST
26 def getBlogInfo():
27     nextBlog = "https://www.blogger.com/next-blog?navBar=true&blogID=8145271598519191285"
28     feedLinks = []
29     try:
30         resp = urllib2.urlopen(nextBlog)
31         if (resp.code == 200):
32             html = resp.read()
33             soup = bs(html)
34             ttl = soup.title.string.encode('ascii')
35             rss = soup.find('link', type='application/atom+xml')
36             if(rss != []):
37                 rss = rss.get('href')
38                 feedLinks = allPages(rss)
39                 feedLinks.insert(0,rss)
40                 return ttl, feedLinks
41             else:
42                 return ttl, rss
43         else:
44             return [], []
45     except:
46         return [], []
47 # COLLECTS ALL FEED PAGES # RETURNS LIST OF FEED PAGES
48 def allPages(link):
49     pages = []
50     tmp = nextPage(link)
51     while(tmp != False):
52         pages.append(tmp)
53         tmp = nextPage(tmp)
54     return pages
55 # CHECK FOR NEXT FEED PAGE # RETURN NEXT FEED PAGE OR FALSE
56 def nextPage(link):
57     try:
58         html = urllib2.urlopen(link).read()
59         soup = bs(html)
60         nxt = soup.find('link', rel="next")
61         if(nxt != []):
62             nxt = nxt.get('href')
63             return nxt
64     except:
65         return False
66 def main():
67     f = open("feedlist.txt", "w")
68     g = open("feedstats.txt", "w")
69     g.write("Title , Pages\n")
70     titles = []; pageCnt = 0; links = []
71     ttl, lks = getBlogInfoInit()
72     for i in range(len(ttl)):
73         titles.append(ttl[i])
74         links.extend(lks[i])
75     print "\n"+ttl[i].rstrip().rstrip()+"\n", len(lks[i])
76     g.write("\n"+ttl[i].rstrip().rstrip()+"\n", "+str(len(lks[i]))+"\n");
77     while(len(set(titles)) < 120):
78         ttl, lks = getBlogInfo()
79         if(ttl != []):
80             titles.append(ttl)
81             print "\n"+ttl.lstrip().rstrip()+"\n", len(lks)
82             g.write("\n"+ttl.lstrip().rstrip()+"\n", "+str(len(lks))+"\n");
83             links.extend(lks)
84     for i in links:

```

```
85 f.write(i+"\n")
```

Listing 1.1: Python module to collect Atom feed links from blogs

The above Python script contains all the functions that are used to fetch a random blog, grab the Atom feed link and retrieve the paged feed links. The collection of Atom feed links, which link to each page of the feed, is saved in a file “feedlist.txt” that is used by the “generatefeevector.py” script to produce the blog matrix. In addition to this another file, “feedstats.txt”, is created that contains the title of each blog and the number of pages the Atom feed contains.

The code provided in the PCI book [1] takes “feedlist.txt” as input and generates a blog matrix where every column is a word from a blog which adheres to this restriction:

```
1 frac=float(bc)/len(feedlist)
  if frac>0.1 and frac<0.5:
```

Listing 1.2: Python module to collect Atom feed links from blogs

Despite naive this restriction is accepted but the number of terms is limited to the top 500. To achieve this a small modification was required. The words were collected with their respective frequency value and then sorted. From the resulting list the top 500 terms were used.

```
wordrank = []
2 for (w, bc) in apcount.items():
    frac = float(bc) / len(feedlist)
    if frac > 0.1 and frac < 0.5:
        wordrank.append((w, frac))
6
wordlist = []
8 # sort list based on frequency
wordrank.sort(key=lambda tup:tup[1], reverse=True)
10 for i in range(500):
    wordlist.append(wordrank[i][0].encode('ascii'))
```

Listing 1.3: Python module to collect Atom feed links from blogs

A histogram of the number of feed pages can be easily generate from the data in “feedstats.txt”. The data was stored in a comma seperated value format with all string enclosed in quotation marks to facilitate the file reading required by R.

The histogram function in R takes a vector of values and generates a frequency histogram with a default number of bins, usually 10. This is convenient, but produces bars for a range of integer values, e.g. 30 blogs have between 1 and 10 pages, 26 blogs have between 11 and 20 pages. However the desired results is one unit intervals, e.g., 30 blogs with just one page, 27 with two pages, 29 with 3 pages and so on. To achieve this we must specify the number of bins that we want to be produced in the histogram. This depends on the number of distinct values that occur. In the case that each blog has a different number of pages this value is equal to the number of blogs and therefore we can use that as an upper bound.

```
#!/usr/bin/Rscript
2
a = read.csv("feedstats.txt", sep=",", head=TRUE)
b = a[,2];
pdf("freqHist.pdf")
6 hist(b, breaks=length(b), main="Blog Histogram", xlab="# of pages", ylab="# of blogs")
axis(side=1, at=seq(0,110,10), labels=seq(0,110,10))
8 axis(side=2, at=seq(0,15,2), labels=seq(0,15,2)); dev.off()

10 pdf("densHist.pdf")
mean.b = mean(b)
12 var.b = var(b)
hist(b, breaks=length(b), main="Blog Histogram", xlab="# of pages", ylab="Density", freq=F)
14 axis(side=1, at=seq(0,110,10), labels=seq(0,110,10))
axis(side=2, at=seq(0,15,2), labels=seq(0,15,2))
16 curve(dgamma(x, shape = mean.b^2/var.b, scale = var.b/mean.b), add = T); dev.off()
pdf("densHist1.pdf")
18 hist(b, main="Blog Histogram", xlab="# of pages", ylab="Density", freq=F)
axis(side=1, at=seq(0,110,10), labels=seq(0,110,10))
20 axis(side=2, at=seq(0,15,2), labels=seq(0,15,2))
curve(dgamma(x, shape = mean.b^2/var.b, scale = var.b/mean.b), add = T); dev.off()
```

Listing 1.4: R script to produce histogram and density plot

The expected outcome from intuition is that the histogram will follow a power-law trend where there are many blogs that contain few pages, at least one, but there are very few that have many pages. This result seems

reasonable from previous assignments, but it is interesting if this simple “many-few, few-many” approach true characterizes the data, or is a useful approximation.

There is the possibility that there are very few bloggers that only have one page of feeds. Assuming someone was motivated to start blogging they would be very active in the beginning and write a substantial number of posts, enough for a few pages. Then the hype dies off and port become more infrequent and only the few committed bloggers stay active enough to collect many pages. But most people would write at least a few pages, meaning that the bulk of bloggers have some in between 1 or 2 pages and many pages. This is known as the Gamma distribution

The Gamma distribution describes a family of distributions, each with a different shape based on the parameter values. The known power-law distribution is a special case of the Gamma distribution. The Gamma distribution has the characteristic that it does not take negative values and is infinite in the positive direction. This is in contrast to the normal distribution that is defined for negative values. In this application a negative number of pages has no meaning and the Gamma distribution is the appropriate choice. Using the information from a useful blog post [2] we can create a histogram of the data and overlay the corresponding Gamma function fitted to the data’s statistical modes, i.e. the mean and variance. The following graphs show the histogram of the data as well as the fitted Gamma distribution.

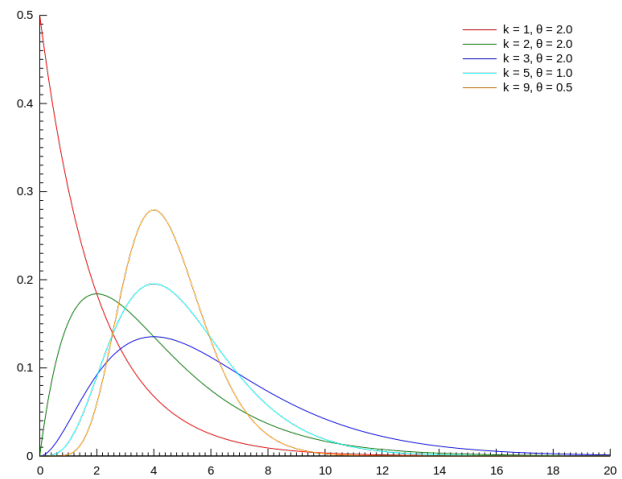


Fig. 1.1: Gamma distributions

Blog Histogram

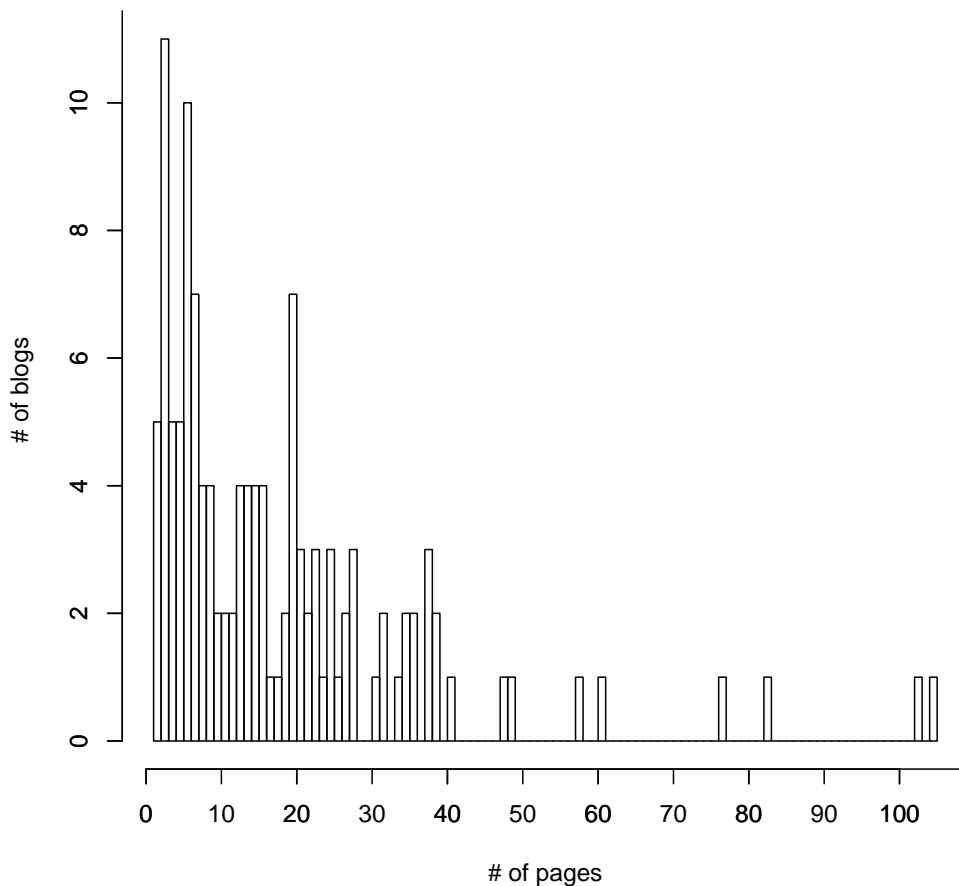


Fig. 1.2: Frequency Histogram (bins = no. blogs)

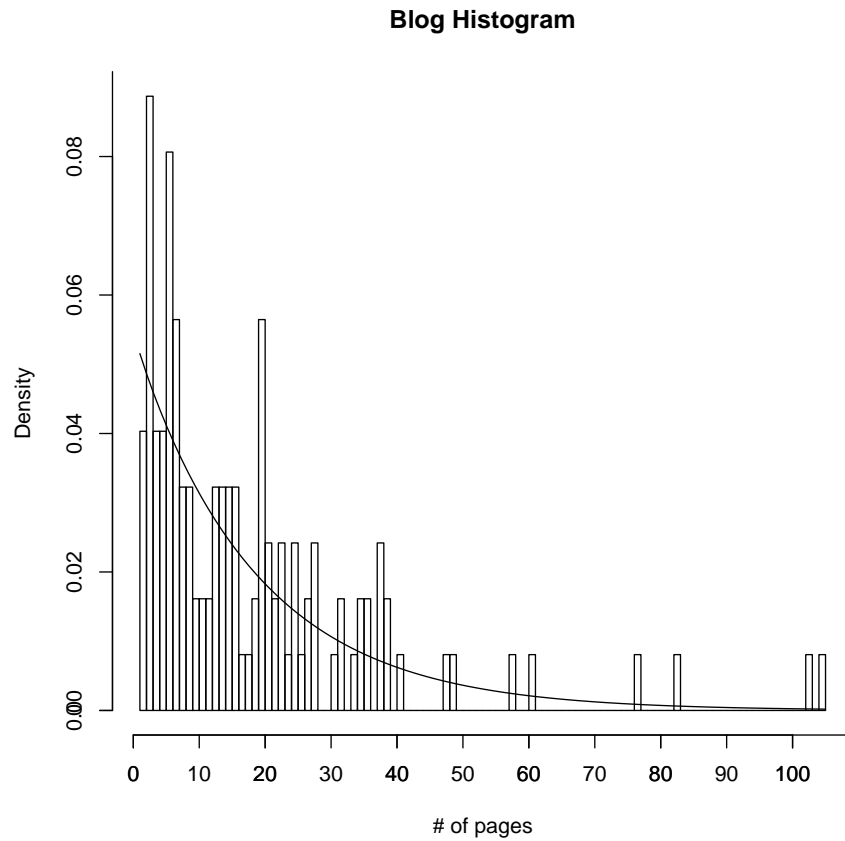


Fig. 1.3: Probability Density Histogram (bins = no. blogs) with fitted Gamma distribution

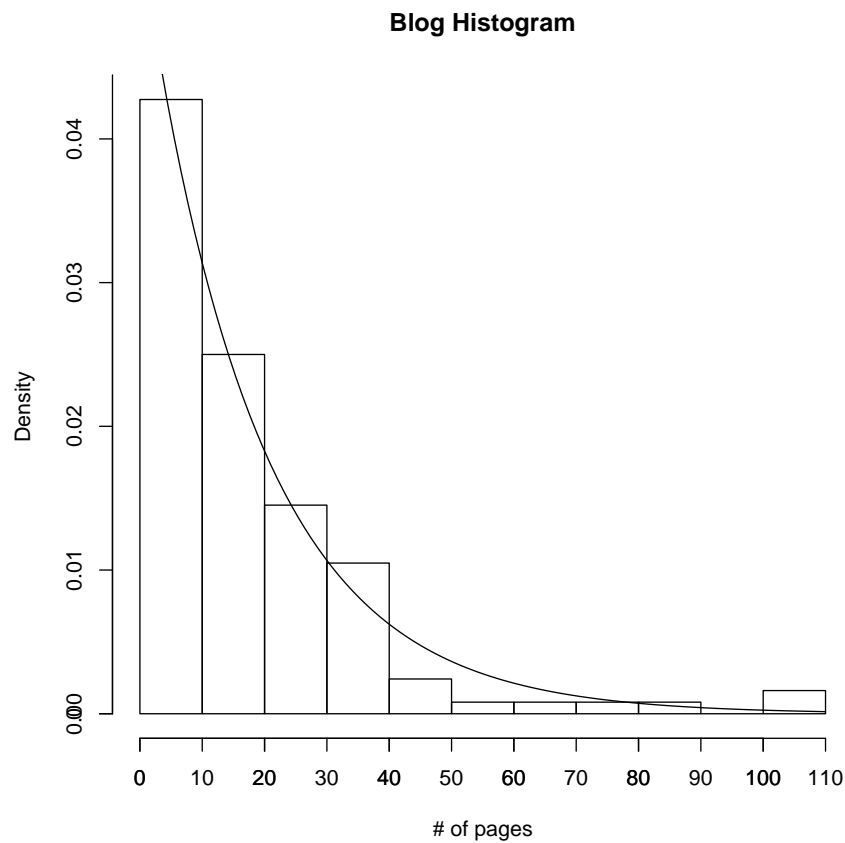


Fig. 1.4: Probability Density Histogram (bins = 10) with fitted Gamma distribution

1.2 Question 2

1.2.1 The 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).

1.2.2 The Answer

The Python function used to produce the ascii and jpeg dendrograms is provided in the book [1]. The output of the script was redirected to a text file “ascii.txt”.

```
1 #!/usr/bin/python
3 import clusters
  blognames, words, data=clusters.readFile('blogdata.txt')
5 clust=clusters.hcluster(data)
  clusters.printclust(clust, labels=blognames)
7
  clusters.drawdendrogram(clust, blognames, jpeg='blogdendro.jpg')
```

Listing 1.5: Python script to produce dendrograms

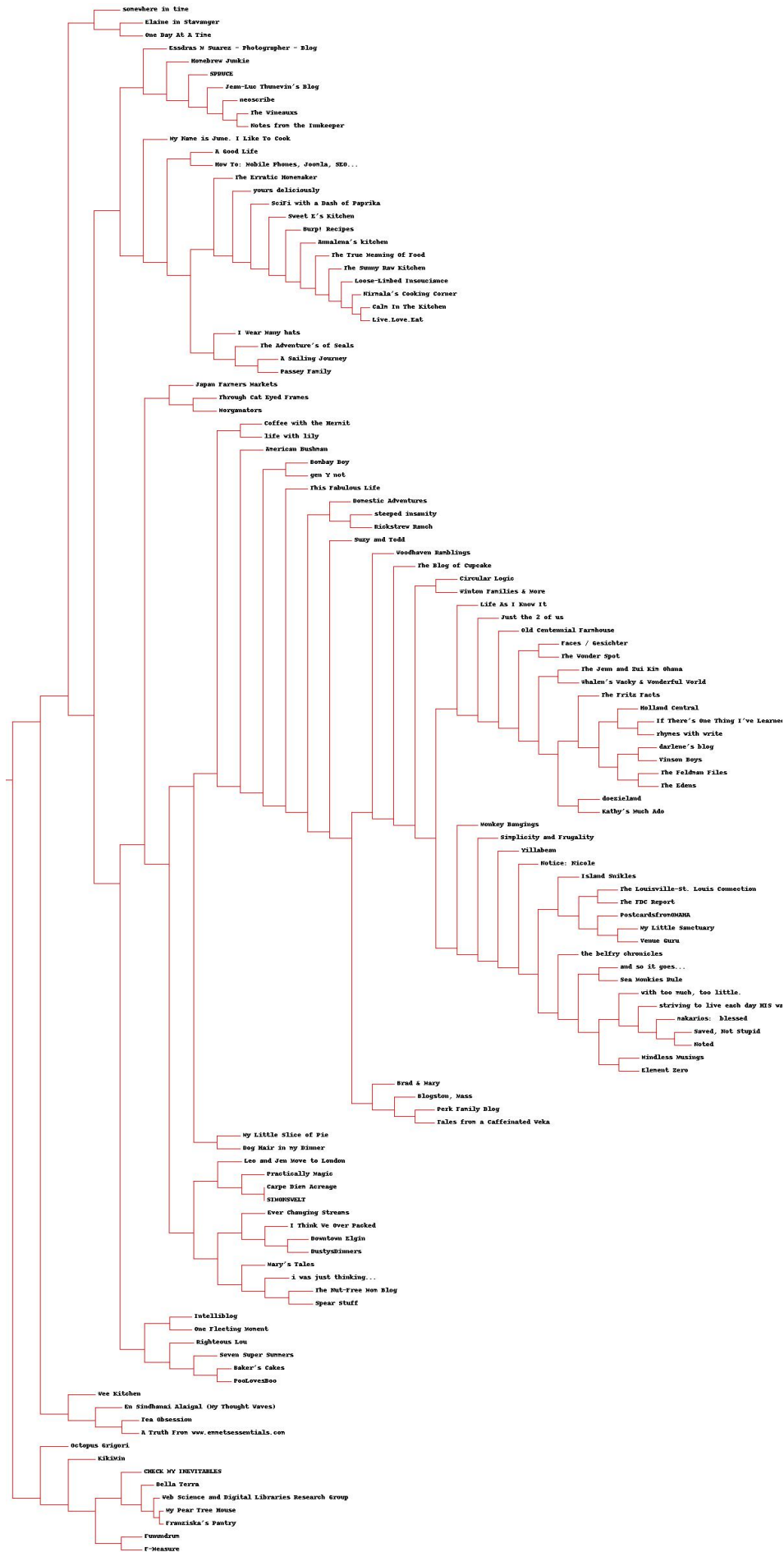


Fig. 1.5: JPEG dendrogram

1.3 Question 3

1.3.1 The Question

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

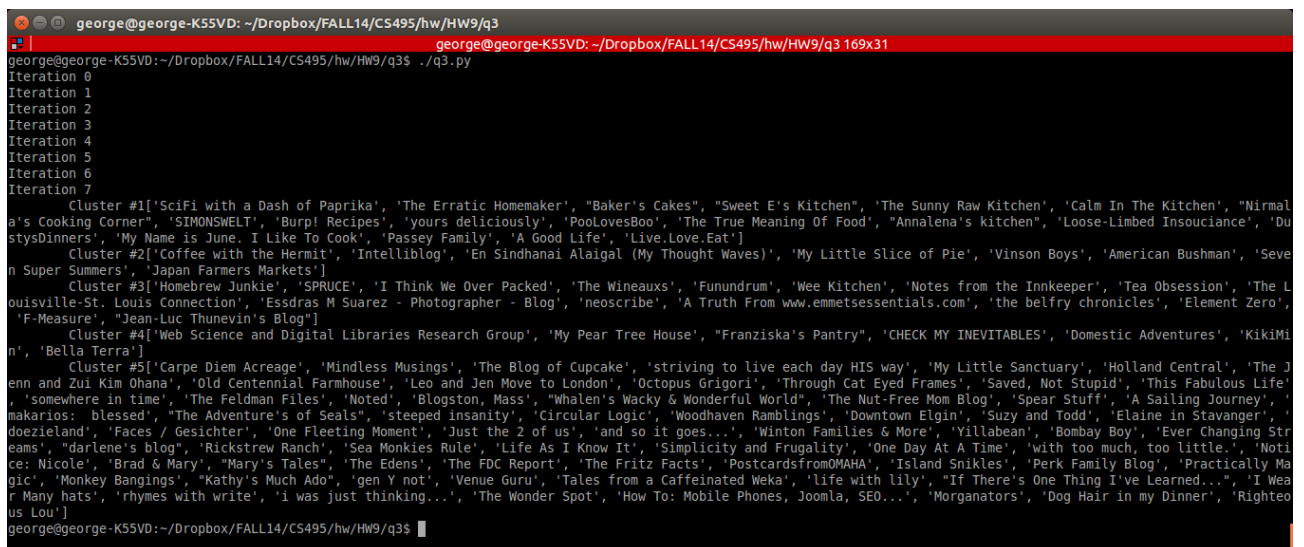
1.3.2 The Answer

The Python function used to produce the K-Means is provided in the book [1]. Clustering into 5 clusters required 7 iterations. Clustering into 10 clusters required 6 iterations. Clustering into 20 clusters required 6 iterations. The terminal output for the script is provided and displays that cluster members

```
#!/usr/bin/python
import clusters

no = 20
blognames, words, data = clusters.readfile('blogdata.txt')
kclust = clusters.kcluster(data, k=no)
for i in range(no):
    print "\tCluster #" + str(i+1) + str([blognames[r] for r in kclust[i]])
```

Listing 1.6: Python script to produce dendrograms



```
george@george-K55VD: ~/Dropbox/FALL14/CS495/hw/HW9/q3
george@george-K55VD: ~/Dropbox/FALL14/CS495/hw/HW9/q3$ ./q3.py
Iteration 0
Iteration 1
Iteration 2
Iteration 3
Iteration 4
Iteration 5
Iteration 6
Iteration 7
Cluster #1['SciFi with a Dash of Paprika', 'The Erratic Homemaker', 'Baker's Cakes', 'Sweet E's Kitchen', 'The Sunny Raw Kitchen', 'Calm In The Kitchen', 'Nirmal
a's Cooking Corner', 'SIMONSWELT', 'Burp! Recipes', 'yours deliciously', 'PooLovesBoo', 'The True Meaning Of Food', 'Annalena's kitchen', 'Loose-Limbed Insouciance', 'Du
stysDinners', 'My Name is June. I Like To Cook', 'Passey Family', 'A Good Life', 'Live.Love.Eat']
Cluster #2['Coffee with the Hermit', 'Intelliblog', 'En Sindhanai Alaigal (My Thought Waves)', 'My Little Slice of Pie', 'Vinson Boys', 'American Bushman', 'Seve
n Super Summers', 'Japan Farmers Markets']
Cluster #3['Homebrew Junkie', 'SPRUCE', 'I Think We Over Packed', 'The Wineauxs', 'Funundrum', 'Wee Kitchen', 'Notes from the Innkeeper', 'Tea Obsession', 'The L
ouisville-St. Louis Connection', 'Essdras M Suarez - Photographer - Blog', 'neoscribe', 'A Truth From www.emmetseentials.com', 'the belfry chronicles', 'Element Zero',
'F-Measure', 'Jean-Luc Thunevin's Blog']
Cluster #4['Web Science and Digital Libraries Research Group', 'My Pear Tree House', 'Franziska's Pantry', 'CHECK MY INEVITABLES', 'Domestic Adventures', 'KikiMi
n', 'Bella Terra']
Cluster #5['Carpe Diem Acreage', 'Mindless Musings', 'The Blog of Cupcake', 'striving to live each day HIS way', 'My Little Sanctuary', 'Holland Central', 'The J
enn and Zui Kim Ohana', 'Old Centennial Farmhouse', 'Leo and Jen Move to London', 'Octopus Grigori', 'Through Cat Eyed Frames', 'Saved, Not Stupid', 'This Fabulous Life',
'somewhere in time', 'The Feldman Files', 'Noted', 'Blogston, Mass', 'Whalen's Wacky & Wonderful World', 'The Nut-Free Mom Blog', 'Spear Stuff', 'A Sailing Journey', '
makarios: blessed', 'The Adventure's of Seals', 'steeped insanity', 'Circular Logic', 'Woodhaven Ramblings', 'Downtown Elgin', 'Suzy and Todd', 'Elaine in Stavanger', '
doezieland', 'Faces / Gesichter', 'One Fleeting Moment', 'Just the 2 of us', 'and so it goes...', 'Winton Families & More', 'Yillabean', 'Bombay Boy', 'Ever Changing Str
eams', 'darlene's blog', 'Rickstrew Ranch', 'Sea Monkies Rule', 'Life As I Know It', 'Simplicity and Frugality', 'One Day At A Time', 'with too much, too little.', 'Noti
ce: Nicole', 'Brad & Mary', 'Mary's Tales', 'The Edens', 'The FDC Report', 'The Fritz Facts', 'PostcardsfromOMAHA', 'Island Snikles', 'Perk Family Blog', 'Practically Ma
gic', 'Monkey Bangings', 'Kathy's Much Ado', 'gen Y not', 'Venue Guru', 'Tales from a Caffeinated Weka', 'life with lily', 'If There's One Thing I've Learned...', 'I Wea
r Many hats', 'rhymes with write', 'i was just thinking...', 'The Wonder Spot', 'How To: Mobile Phones, Joomla, SEO...', 'Morganators', 'Dog Hair in my Dinner', 'Righteo
us Lou']
george@george-K55VD: ~/Dropbox/FALL14/CS495/hw/HW9/q3$
```

Fig. 1.6: Terminal output from K-Means CLustering, $k = 5$

```

george@george-K55VD: ~/Dropbox/FALL14/CS495/hw/HW9/q3
george@george-K55VD: ~/Dropbox/FALL14/CS495/hw/HW9/q3 169x31
george@george-K55VD:~/Dropbox/FALL14/CS495/hw/HW9/q3$ ./q3.py
Iteration 0
Iteration 1
Iteration 2
Iteration 3
Iteration 4
Iteration 5
Iteration 6
Cluster #1['Carpe Diem Acreage', 'The Jenn and Zui Kim Ohana', 'Octopus Grigori', 'SIMONSWELT', 'doezieland', 'Faces / Gesichter', 'Just the 2 of us', 'Winton Families & More', 'Sea Monkeys Rule', 'Mary's Tales', 'Element Zero', 'i was just thinking...']
Cluster #2['The Blog of Cupcake', 'striving to live each day HIS way', 'Saved, Not Stupid', 'Noted', 'makarios: blessed', 'En Sindhanai Alaigal (My Thought Wave s)', 'with too much, too little.', 'Japan Farmers Markets']
Cluster #3['Holland Central', 'Through Cat Eyed Frames', 'The Feldman Files', 'Whalen's Wacky & Wonderful World', 'Spear Stuff', 'A Sailing Journey', 'Suzy and Todd', 'My Little Slice of Pie', 'Ever Changing Streams', 'darlene's blog', 'The Edens', 'Vinson Boys', 'The Fritz Facts', 'If There's One Thing I've Learned...', 'I Wear Many hats', 'rhymes with write', 'Seven Super Summers', 'The Wonder Spot', 'Morganators']
Cluster #4['My Little Sanctuary', 'I Think We Over Packed', 'somewhere in time', 'Blogston, Mass', 'Downtown Elgin', 'Essdras M Suarez - Photographer - Blog', 'One Fleeting Moment', 'Bombay Boy', 'One Day At A Time', 'Brad & Mary', 'American Bushman', 'Perk Family Blog', 'gen Y not', 'Venue Guru', 'Dog Hair in my Dinner']
Cluster #5['Web Science and Digital Libraries Research Group', 'My Pear Tree House', 'Franziska's Pantry', 'CHECK MY INEVITABLES', 'Domestic Adventures', 'KikiMin', 'Bella Terra']
Cluster #6['Mindless Musings', 'Old Centennial Farmhouse', 'Leo and Jen Move to London', 'Funundrum', 'This Fabulous Life', 'Circular Logic', 'The Louisville-St. Louis Connection', 'Yillabeau', 'Rickstrew Ranch', 'Life As I Know It', 'Simplicity and Frugality', 'Notice: Nicole', 'the belfry chronicles', 'The FDC Report', 'PostcardsfromOMAHA', 'Island Snikles', 'Practically Magic', 'Monkey Bangings', 'Tales from a Caffeinated Weka', 'F-Measure', 'How To: Mobile Phones, Joomla, SEO...', 'Righteous Lou']
Cluster #7['Tea Obsession', 'and so it goes...', 'A Truth From www.emmetssentials.com', 'Kathy's Much Ado']
Cluster #8['Homebrew Junkie', 'SPRUCE', 'Baker's Cakes', 'The Wineauxs', 'Wee Kitchen', 'Notes from the Innkeeper', 'The Nut-Free Mom Blog', 'Intelliblog', 'neoscribe', 'PoolovesBoo', 'My Name is June. I Like To Cook', 'Jean-Luc Thunevin's Blog']
Cluster #9['The Erratic Homemaker', 'Coffee with the Hermit', 'The Adventure's of Seals', 'steeped insanity', 'Woodhaven Ramblings', 'Elaine in Stavanger', 'Life with Lily']
Cluster #10['SciFi with a Dash of Paprika', 'Sweet E's Kitchen', 'The Sunny Raw Kitchen', 'Calm In The Kitchen', 'Nirmala's Cooking Corner', 'Burp! Recipes', 'yours deliciously', 'The True Meaning Of Food', 'Annalena's kitchen', 'Loose-Limbed Insouciance', 'DustysDinners', 'Passey Family', 'A Good Life', 'Live.Love.Eat']
george@george-K55VD:~/Dropbox/FALL14/CS495/hw/HW9/q3$

```

Fig. 1.7: Terminal output from K-Means CClustering, k = 10

```

george@george-K55VD: ~/Dropbox/FALL14/CS495/hw/HW9/q3
george@george-K55VD: ~/Dropbox/FALL14/CS495/hw/HW9/q3 169x38
george@george-K55VD:~/Dropbox/FALL14/CS495/hw/HW9/q3$ ./q3.py
Iteration 0
Iteration 1
Iteration 2
Iteration 3
Iteration 4
Iteration 5
Iteration 6
Cluster #1['Baker's Cakes', 'The Sunny Raw Kitchen', 'Calm In The Kitchen', 'Intelliblog', 'SIMONSWELT', 'The True Meaning Of Food', 'Annalena's kitchen', 'DustysDinners', 'A Good Life', 'Live.Love.Eat']
Cluster #2['Mindless Musings', 'Octopus Grigori', 'My Little Slice of Pie', 'Element Zero', 'Dog Hair in my Dinner']
Cluster #3['Carpe Diem Acreage', 'Homebrew Junkie', 'SPRUCE', 'Leo and Jen Move to London', 'The Wineauxs', 'Notes from the Innkeeper', 'The Nut-Free Mom Blog', 'Circular Logic', 'Downtown Elgin', 'Elaine in Stavanger', 'The Louisville-St. Louis Connection', 'Essdras M Suarez - Photographer - Blog', 'neoscribe', 'Sea Monkeys Rule', 'Simplicity and Frugality', 'Brad & Mary', 'the belfry chronicles', 'Mary's Tales', 'The FDC Report', 'PostcardsfromOMAHA', 'Island Snikles', 'Practically Magic', 'Venue Guru', 'Tales from a Caffeinated Weka', 'Jean-Luc Thunevin's Blog']
Cluster #4['Noted', 'Coffee with the Hermit', 'steeped insanity', 'Life with Lily']
Cluster #5[]
Cluster #6['My Name is June. I Like To Cook']
Cluster #7['I Think We Over Packed', 'makarios: blessed', 'Faces / Gesichter', 'Burp! Recipes', 'Winton Families & More', 'Notice: Nicole', 'Righteous Lou']
Cluster #8['My Little Sanctuary', 'The Jenn and Zui Kim Ohana', 'Old Centennial Farmhouse', 'Funundrum', 'Blogston, Mass', 'Whalen's Wacky & Wonderful World', 'doezieland', 'Just the 2 of us', 'and so it goes...', 'The Fritz Facts', 'Perk Family Blog', 'Kathy's Much Ado', 'gen Y not', 'rhymes with write', 'Seven Super Summers', 'i was just thinking...', 'The Wonder Spot']
Cluster #9['Through Cat Eyed Frames', 'This Fabulous Life', 'somewhere in time', 'The Feldman Files', 'KikiMin', 'Monkey Bangings', 'Japan Farmers Markets']
Cluster #10['with too much, too little']
Cluster #11['En Sindhanai Alaigal (My Thought Waves)', 'Bombay Boy', 'Ever Changing Streams', 'One Day At A Time', 'Domestic Adventures']
Cluster #12[]
Cluster #13['Spear Stuff', 'PoolovesBoo', 'American Bushman']
Cluster #14['Web Science and Digital Libraries Research Group', 'My Pear Tree House', 'CHECK MY INEVITABLES', 'Bella Terra']
Cluster #15['Tea Obsession', 'Woodhaven Ramblings', 'A Truth From www.emmetssentials.com', 'If There's One Thing I've Learned...']
Cluster #16['The Blog of Cupcake', 'striving to live each day HIS way', 'Holland Central', 'Saved, Not Stupid', 'A Sailing Journey', 'Suzy and Todd', 'One Fleeting Moment', 'darlene's blog', 'Rickstrew Ranch', 'Life As I Know It', 'The Edens', 'Vinson Boys', 'I Wear Many hats', 'Morganators']
Cluster #17['Franziska's Pantry']
Cluster #18['SciFi with a Dash of Paprika', 'The Erratic Homemaker', 'Sweet E's Kitchen', 'Nirmala's Cooking Corner', 'yours deliciously', 'Loose-Limbed Insouciance', 'Passey Family', 'F-Measure']
Cluster #19['Wee Kitchen', 'The Adventure's of Seals', 'Yillabeau', 'How To: Mobile Phones, Joomla, SEO...']
Cluster #20[]
george@george-K55VD:~/Dropbox/FALL14/CS495/hw/HW9/q3$

```

Fig. 1.8: Terminal output from K-Means CClustering, k = 20

1.4 Question 4

1.4.1 The Question

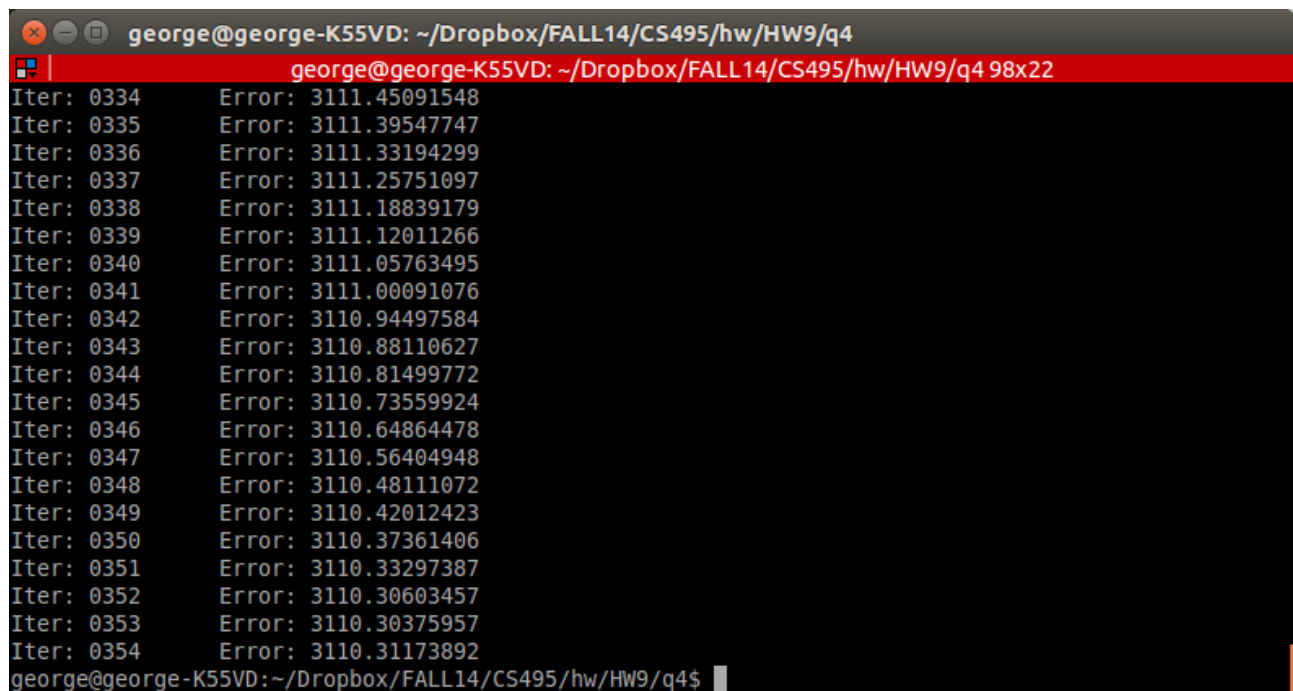
Use MDS to create a JPEG of the blogs similar to slide 29. How many iterations were required?

1.4.2 The Answer

The Python function used to produce the MDS plot is provided in the book [1]. The algorithm required 353 iterations to converge. The terminal output it attached.

```
1 #! /usr/bin/python
3 import clusters
  blognames, words, data=clusters.readfile('blogdata.txt')
5 coords=clusters.scaledown(data)
  clusters.draw2d(coords, blognames, jpeg='blogs2d.jpg')
```

Listing 1.7: Python script to produce dendrograms

A terminal window with a black background and white text. The title bar shows the user 'george' on a machine named 'george-K55VD' in the directory '~/Dropbox/FALL14/CS495/hw/HW9/q4'. The prompt is 'george@george-K55VD: ~/Dropbox/FALL14/CS495/hw/HW9/q4 98x22'. The output consists of 21 lines, each showing an iteration number followed by the word 'Error:' and a numerical value. The iteration numbers range from 0334 to 0354. The error values start at 3111.45091548 and decrease to 3110.31173892. The terminal window has standard window controls (close, maximize, minimize) in the top left corner.

```
george@george-K55VD: ~/Dropbox/FALL14/CS495/hw/HW9/q4
george@george-K55VD: ~/Dropbox/FALL14/CS495/hw/HW9/q4 98x22
Iter: 0334      Error: 3111.45091548
Iter: 0335      Error: 3111.39547747
Iter: 0336      Error: 3111.33194299
Iter: 0337      Error: 3111.25751097
Iter: 0338      Error: 3111.18839179
Iter: 0339      Error: 3111.12011266
Iter: 0340      Error: 3111.05763495
Iter: 0341      Error: 3111.00091076
Iter: 0342      Error: 3110.94497584
Iter: 0343      Error: 3110.88110627
Iter: 0344      Error: 3110.81499772
Iter: 0345      Error: 3110.73559924
Iter: 0346      Error: 3110.64864478
Iter: 0347      Error: 3110.56404948
Iter: 0348      Error: 3110.48111072
Iter: 0349      Error: 3110.42012423
Iter: 0350      Error: 3110.37361406
Iter: 0351      Error: 3110.33297387
Iter: 0352      Error: 3110.30603457
Iter: 0353      Error: 3110.30375957
Iter: 0354      Error: 3110.31173892
george@george-K55VD:~/Dropbox/FALL14/CS495/hw/HW9/q4$
```

Fig. 1.9: Terminal output from Multi-Dimensional Scaling



Fig. 1.10: Multi-Dimensional Scaling of the blogs

1.5 Extra Credit Question

1.5.1 The 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.

1.5.2 The Answer

The TFIDF calculation requires the computation of two numbers, the term frequency and the inverse document frequency. The *term frequency* (TF) is defined as the frequency of the term in the document normalized by the number of terms in the document.

$$TF = \frac{\text{number of term occurrences}}{\text{number of words in document}}$$

The *inverse document frequency* (IDF) of the query is defined as the log base 2 of the number of webpages in the corpus over the number of webpages containing the term.

$$IDF = \log_2\left(\frac{\text{number of webpages in corpus}}{\text{number of webpages containing the term}}\right)$$

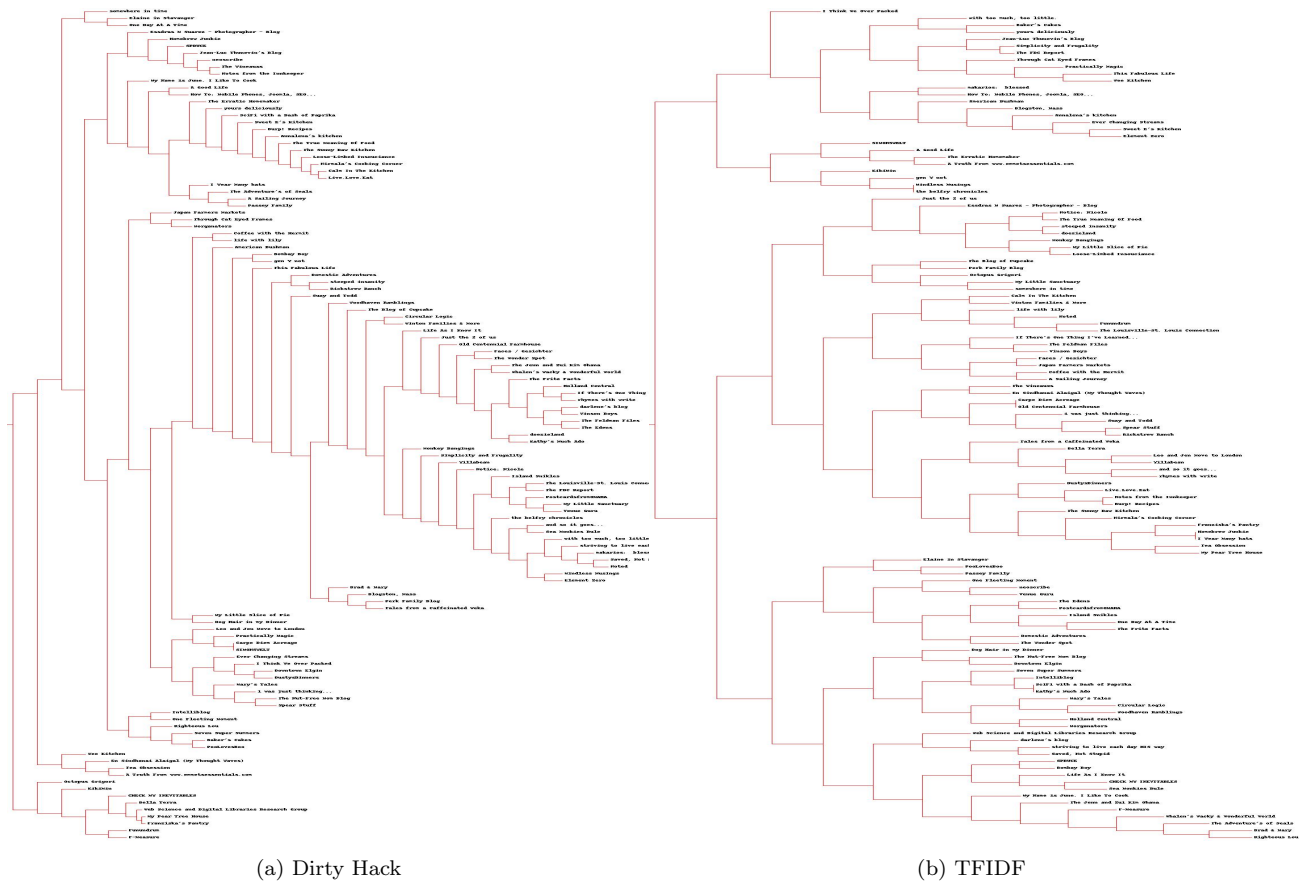
In the context of the blog matrix this can be visualized easily. Each row consists of a blog vector and each column consists of a term column vector. The elements of the matrix are the occurrence frequency of a given term in a specific blog. The code essentially scaled the frequency by the length of the blog feed, but that is constant across all terms. The term frequency of a term in a blog is the frequency divided by the number of terms in the blog. In the matrix that is the element value divided by the length of the blog vector. Then summing across all blogs gives the sum of the term frequencies of all blogs which results in a vector of length equal to the number of terms in the corpus. This is the TF value. The numerator of the IDF value is constant across all terms and therefore irrelevant. The denominator is the number of blogs that the term appeared in. In matrix view, this represents the number of non-zero elements in a column. Unfortunately the implementation the book used was not matrix based and therefore some magic had to take place to figure out how to compute these values. However the proper use of dictionaries made the implementation efficient and straight-forward.

```

tf = {}
2   blogs = wordcounts
   # go through every blog
4   for blog in blogs:
       # go through every term
6       for term in blogs[blog]:
           if term in tf:
8               tf[term] += blogs[blog][term]/float(len(blogs[blog]))
           else:
10              tf[term] = blogs[blog][term]/float(len(blogs[blog]))
   corp = float(len(blogs))
12   idf = {}
   for i in apcount:
14       idf[i] = log(corp/apcount[i])/log(2)
   tfidf = {}
16   for i in tf:
       tfidf[i] = tf[i]*idf[i]
18   new = []
   for i in tfidf:
20       new.append((i, tfidf[i]))
   new.sort(key=lambda tup:tup[1], reverse=True)
22   wordlist = []
   for i in range(500):
24       print new[i][0]
       wordlist.append(new[i][0].encode('ascii'))

```

Listing 1.8: Python script to produce dendrograms



The overall accuracy of the two methods is hard to evaluate in an absolute sense because the blogs are not labeled in a way that provides insight to an absolute form of clustering. An attempt to create clusters manually based on the content is not objective and can be easily misled by our perception of the blogs. Therefore there is no method to score the outcome can then compare the score of the two techniques.

The one thing that does seem to stand out is the distinct tree structure that the two methods display. This becomes more clear when they are placed alongside each other. The naive term frequency tree seems to randomly split at every iteration. Groupings split up in unequal groups at each step, i.e. a cluster will split up into two clusters, one containing a single blog and then other the rest. This shows that the groups are not very consistent and at the next split the outliers branch off into their own group. This process continues as each cluster loses a blog at each iteration and gets smaller, without any indication of a distinct separation into two groups. For example, the food cluster should split up into the desert and the entree clusters, but in this case the food cluster sheds a blog at a time.

The TFIDF dendrogram appears more structured and precise, very close to binary tree. Throughout the entire process the clusters are divided into subgroups. There are occurrences of the random “one-off” blog, but that is predictable. Also, the number of terms used is only 500, which many not be large enough to provide a proper result. The TFIDF method appears to find dominant characteristics of the cluster to partition on. While the naive method finds the most eccentric member and removes him.

One a more abstract, and slightly “hand-wavy”, interpretation blogs represent the people that write them. A population can be clustered into in a variety of ways, i.e. hobby, occupation, gender, age, but the clusters that provide the most information are the ones that partition the cluster into the subgroups that are bound by a characteristic. This type of clustering is equivalent to a useful demographic map, e.g. young male engineers, young female engineers, retired wood craftsman, etc. The group can be used to describe its members in a reasonable way. However, the population can be clustered based on who is a Harvard graduate that is also in an underground alt-rock band and not. There is lots of detail about one group, but the members of the other group are not held together by any common traits. Adolf Hitler and Edward Snowden would be in the same cluster and then the question becomes what do they have in common.

In conclusion, my opinion is that the TFIDF measure clusters in a way that produces distinct subgroups which are meaningful and consistent. On the other hand, the naive approach partitions based on the odd one out, but sometime the odd one isn't very odd and the non-odd ones don't have much to do with each other. This is the conclusion that I draw based on the structure of the divisions. Even if the groups don't seem related based on the title they have some common characteristic that TFIDF can determine, but the naive method does not.

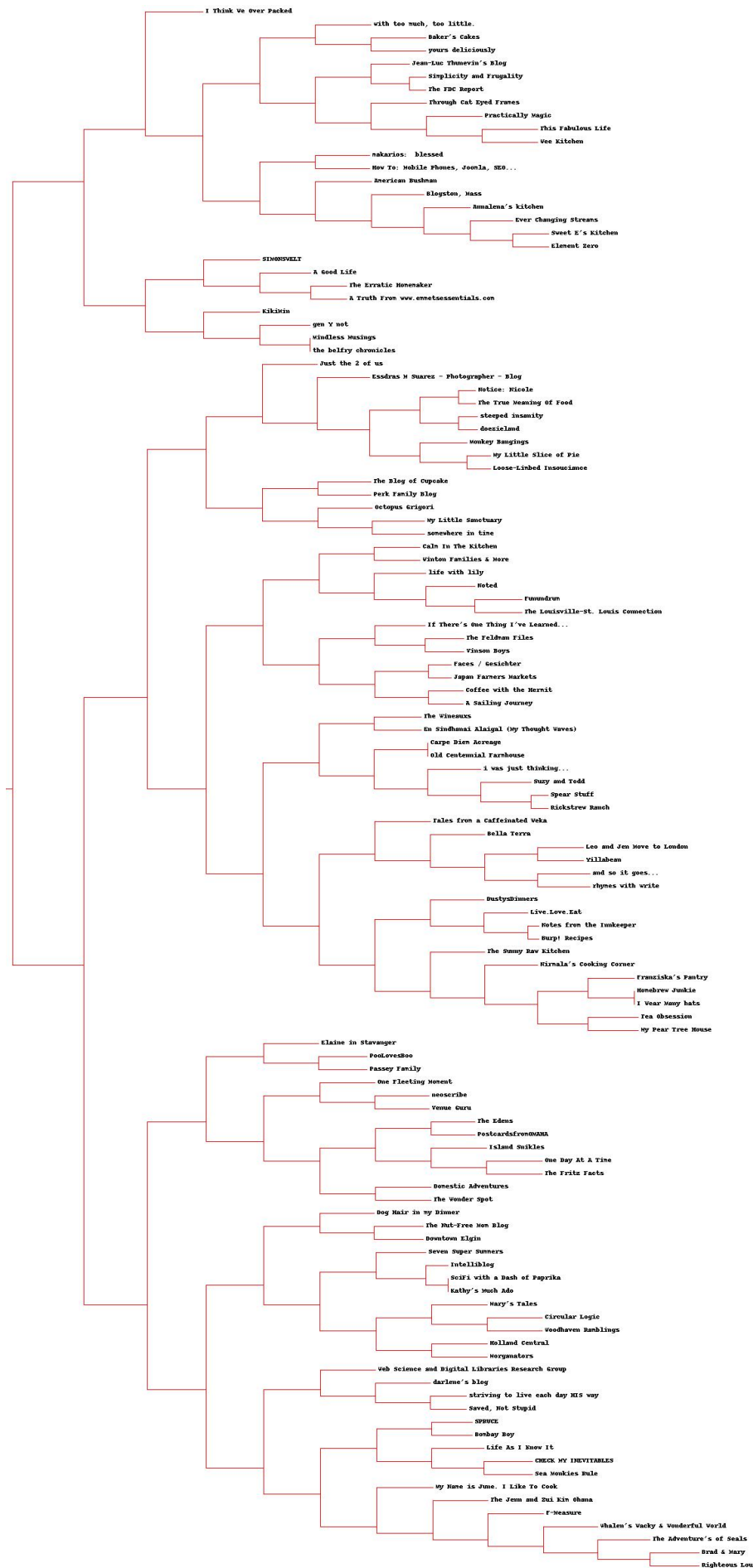


Fig. 1.12: TFIDF dendrogram

References

1. Toby Segaran. *Programming collective intelligence: building smart web 2.0 applications*. " O'Reilly Media, Inc.", 2007.
2. Eric Cai The Chemical Statistician. Exploratory data analysis: Combining histograms and density plots to examine the distribution of the ozone pollution data from new york in r. <http://www.r-bloggers.com/exploratory-data-analysis-combining-histograms-and-density-plots-to-examine-the-distribution-of-the-ozone-pollution-data-f>
3. Maja J Mataric. Designing emergent behaviors: From local interactions to collective intelligence. In *Proceedings of the Second International Conference on Simulation of Adaptive Behavior*, pages 432–441, 1993.