OLD DOMINION UNIVERSITY

Assignment 7

David Bayard

April 25, 2019

QUESTION 1.

Create a blog-term matrix. Start by grabbing 100 blogs hosted on https://www.blogger.com.

Solution:

In order to grab 100 blogs, the google-search API was used. This API provides a function which scrapes URIs from Google, accepting a domain name and query term as parameters, as shown below:

```
for url in search( query="music", stop=170, domains = domains):

domains = {'blogspot.com'}

splitURL = urlparse(url)

  if list is not empty
    if((not listURLs) == False):

      for domain in listURLs:
        splitDomain = urlparse(domain)

        listSplit.append(splitDomain.netloc.split('.')[0])

      if(splitURL.netloc.split('.')[0] not in listSplit):
        listURLs.append(url)

  else:
    listURLs.append(url)
```

Listing 1: Scraping URIs

In the code above, the query term “music” is used to extract URIs under the domain “blogspot.com”. These URIs are then fed to a function which extracts all of the href tags from each URI.

```

feed_urls = html.findAll("link", rel="alternate")
    for f in feed_urls:
        t = f.get("type",None)
        if t:
            if "rss" in t or "xml" in t:
                href = f.get("href",None)
                if href:
                    possible_feeds.append(href)
```

Listing 2: Grab href

All of these href tags are stored in the possible feeds list. This list is then filtered, removing any href tags that are not RSS or Atom feeds.

```
feed_urls = html.findAll("link", rel="alternate")
    for f in feed_urls:
        t = f.get("type",None)
        if t:
            if "rss" in t or "xml" in t:
                href = f.get("href",None)
                if href:
                    possible_feeds.append(href)
```

Listing 3: Scraping URIs

After extracting a list of RSS feeds from the URIs, a matrix was created, where the terms are the title of each blog, and the values are the frequency of occurrence of each term in that blog. This file is generated by using code collected from the Programming Collective Intelligence book.

```
apcount = {}
wordcounts = {}
for feedurl in feedList:
  try:
    (title, wc) = getwordcounts(feedurl)
    wordcounts[title] = wc
    for (word, count) in wc.items():
      apcount.setdefault(word, 0)

      if count > 1:
        apcount[word] += 1
  except:
    print ('Failed to parse feed %s' % feedurl)
```

Listing 4: Extract Words from Title

```
wordlist = []
for (w, bc) in apcount.items():
  frac = float(bc) / len(feedList)

  if(frac > .005 and frac < .5 and len(wordlist) < 1000):
    wordlist.append(w)
```

Listing 5: Extract Highest Terms

```
dataFile.write('Blog')
for word in wordlist: dataFile.write('\t%s' % word)
dataFile.write('\n')
for (blog, wc) in wordcounts.items():
    dataFile.write(blog)
    print(blog)
    for word in wordlist:
        if word in wc:
            dataFile.write('\t%d' % wc[word])
        else:
            dataFile.write('\t0')
    dataFile.write('\n')
```

Listing 6: Write File

The three snippets of code above are responsible for getting each word in the RSS or Atom Title tag, counting the frequency of each term, extracting the terms with the highest frequency (and a limit of 1000), and writing the blog words, blog titles, and word counts to the file.

QUESTION 2

Create an ASCII and JPEG dendrogram that clusters (i.e., HAC) the most similar blogs

Solution:

In both tasks above, functions provided by the Programming Collective Intelligence book are used, providing the data file created in the above code. First the file is read using the readfile function, which returns a list containing each blog title and the corresponding word count. This list is passed to the hcluster which calculates the distance between two blogs, finds the most similar two blogs, and merges these blogs until only one blog remains.

```
clust=clusters.hcluster(data)
asciiFile = open("ascii.txt", 'w')
orig_stdout = sys.stdout
sys.stdout = asciiFile

clusters.printclust(clust,labels=blognames)

sys.stdout = orig_stdout
```

Listing 7: ASCII File Generation

The code snippet above makes use of the printclusters function, which iterates through every branch in the hierarchy and prints the blog title. This is then printed to the output stream, which is rerouted to a file in order to capture the output.

The code below takes advantage of the drawdendogram function provided by the Programming Collective Intelligence book. This function takes the heirarchy structure created by the earlier call to the readfile function, and maps each node using ImageDraw API.

```

clusters.drawdendrogram(clust,blognames,jpeg='blogclust.jpg')
```

Listing 8: ASCII File Generation

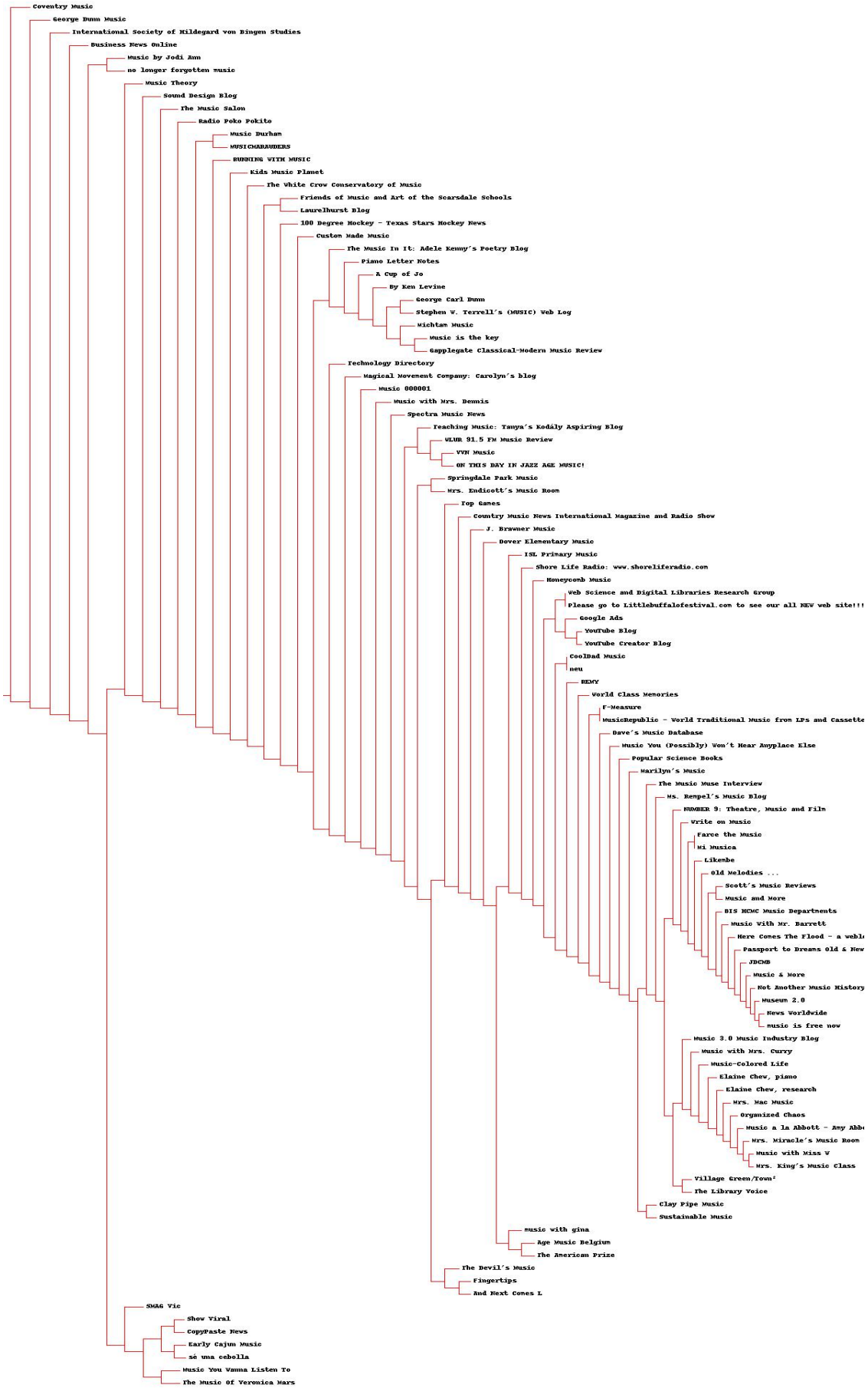

Figure 0.1: Dendogram

QUESTION 3

Cluster the blogs using K-Means, using k=5,10,20.

Solution:

Using the kcluster function provided by the Programming Collective Intelligence book, K-Means of 5, 10, and 20 were calculated and recorded in the clusteredBlogs.txt file. In each case, a table is provided, displaying the clustered blogs in k groups.

```
kclust=clusters.kcluster(data,k=5)
kclust=clusters.kcluster(data,k=10)
kclust=clusters.kcluster(data,k=20)
```

Listing 9: Clustering into 5 groups

The code above returns a list indexed with a number of clusters, depending of the k value passed in. Using this function, the blogs are split into groups of five, ten, and twenty, and the data is accessed by the index of each group number as shown below.

```
for x in range(0,5):

  for r in kclust[x]:
    centralFile.write(blognames[r] + '\n')
    centralFile.write('\n')
  centralFile.write('\n')
  centralFile.write("----------------------------------------------------")
  centralFile.write('\n')
```

Listing 10: Clustering into 5 groups

Clustered Blogs in 5 Groups				
Cluster 1	Cluster 2	Cluster 3	Cluster 4	Cluster 5
F-Measure	Music Durham	Music You Wanna	George Dunn Music	Top Games
CoolDad Music	VVN Music	Coventry Music	The Music In It	100 Degree Hockey
Farce the Music	The Music Salon	Custom Made Music	Piano Letter Notes	Fingertips
HoneyComb Music	Music with Miss W	On This Day		Kids Music Planet
Music is the key	Organized Chaos	Music 00		Music Theory

The table above contains blogs which were split into five clusters. The exact same procedure was followed to produce clusters of ten and twenty, which are contained within the "clusteredBlogs.txt" file.

The number of iterations changes each time the program is run because the random k points may be located in different positions, and may thus be assigned different blogs. In this instance, the number of iterations required were 3 for k = 5, 5 for k = 10, and 6 for k = 20

It is interesting to note that one of the clusters only received 3 blogs. This indicates that k must have been located in a location further away from the central data, preventing it from being assignmed blogs.

QUESTION 4

Use MDS to create a JPEG of the blogs similar to slide 29 of the week 11 lecture. How many iterations were required?

Solution:

By passing the data list to the multidimensional scaling function provided by the Programming Collective Intelligence book, an image representing a two dimensional view of the distance between each blog is produced. This function attempts to move each node according to the combination of all the other nodes pushing and pulling on it, until the total amount of error cannot be reducted. In this case, it took 290 iterations until the amount of error had stabilized, and then the MDS was drawn.

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

Listing 11: Drawing 2d from Programming Collective Intelligence

The image on the next page displays the 2d image created from the reduced nodes coordinates. These values appear to align with the general expectations. For example, “YouTube Creator Blog” appears near “YouTube Blog”, displaying a sense of relation between the two blogs.

Figure 0.2: 2d Drawing

QUESTION 6

Re-run questions 1-4, but this time instead of using the 98 "random" blogs, use 98 blogs that should be "similar" to: http://f-measure.blogspot.com/ http://ws-dl.blogspot.com/

98 other URIs were extracted by using Google and querying terms such as music, data science, data, and songs, among other terms. Using these URIs, the 2d figure, dendogram, and ASCII files were generated in the same manner as the original URIs.

It is evident when comparing the dendograms and 2d files generated by the two data sets, that there are major differences. First and foremost, the “similar” dataset has increased structure and relations. For example, looking at the dendogram on page 10, it is evident that there is almost an entirely new branch with segments into data science. This branch does not exist in the original dendogram because there is a lack of URIs which relate to data science, therefore eleminating any possibility of the branch occurring.

Moving on to the blogs2dExtraCredit, on page 11, it appears as though the clustering of related data has increased. For example, the “Web Science and Digital Research Group” is surronded by blogs such as “Tech Tutorials” and “Web Reflection”. The same could be said for the new dendogram relating to the original URIs, but it is evident that, again, the new dendogram file is more structured. Overall, the introduction of two distinct subjects exponentiates the differences in the terms, leading to more a more defined hierarchy, and thus a more defined structure.

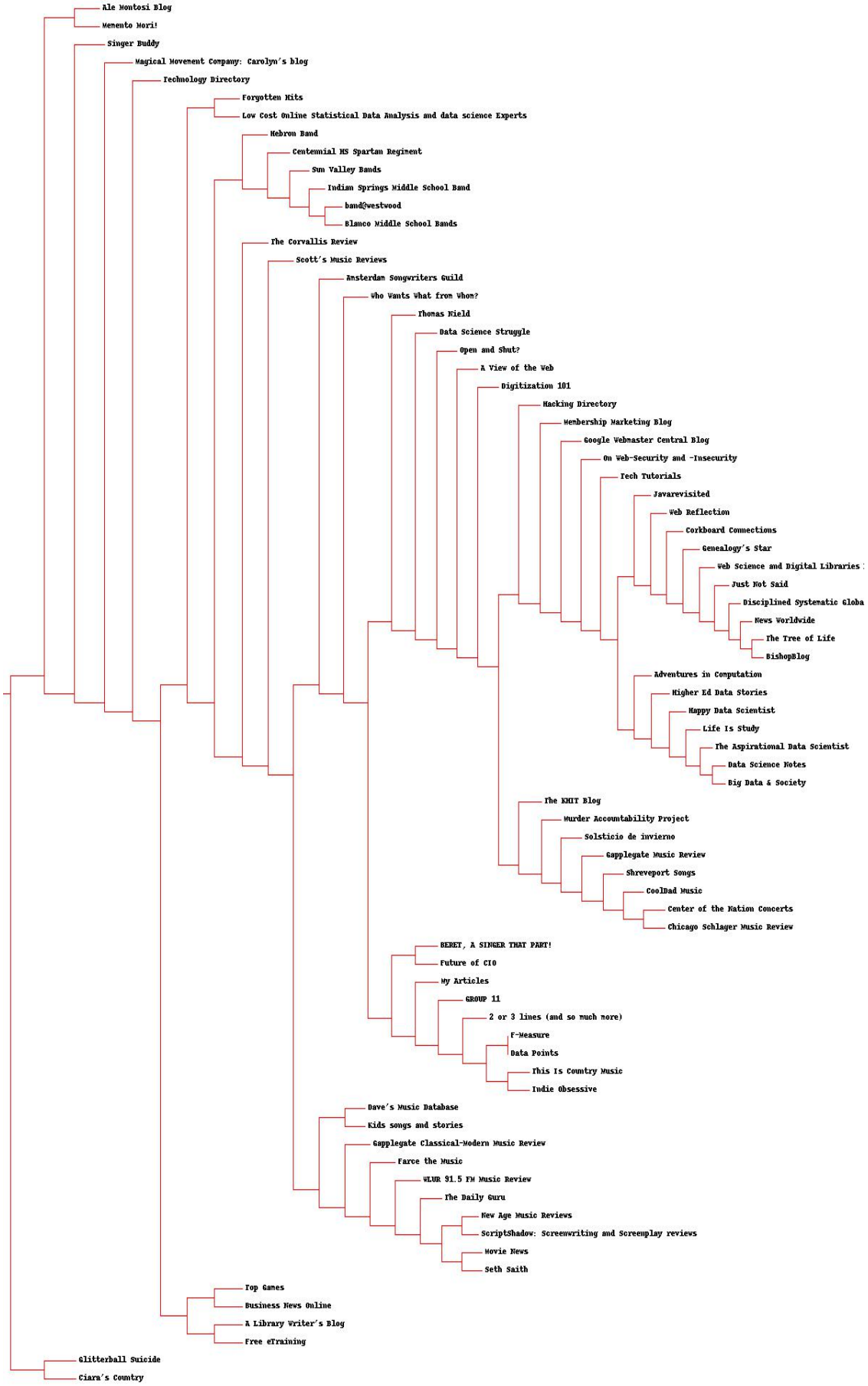

Figure 0.3: Extra Credit Dendogram

Figure 0.4: Extra Credit 2d Drawing