

OLD DOMINION UNIVERSITY

Assignment 7

David Bayard

April 8, 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:

```
1 for url in search( query="music", stop=170, domains = domains):
2
3 domains = { 'blogspot.com' }
4
5 splitURL = urlparse(url)
6
7 if list is not empty
8     if ((not listURLs) == False):
9
10     for domain in listURLs:
11         splitDomain = urlparse(domain)
12
13         listSplit.append(splitDomain.netloc.split('.')[0])
14
15     if (splitURL.netloc.split('.')[0] not in listSplit):
16         listURLs.append(url)
17
18 else:
19     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.

```
1 feed_urls = html.findAll("link", rel="alternate")
2
3 for f in feed_urls:
4     t = f.get("type", None)
5     if t:
6         if "rss" in t or "xml" in t:
7             href = f.get("href", None)
8             if href:
9                 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.

```
1 feed_urls = html.findAll("link", rel="alternate")
2
3 for f in feed_urls:
4     t = f.get("type", None)
5     if t:
6         if "rss" in t or "xml" in t:
7             href = f.get("href", None)
8             if href:
9                 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.

```
1 apcount = {}
2 wordcounts = {}
3 for feedurl in feedList:
4     try:
5         (title , wc) = getwordcounts(feedurl)
6         wordcounts[title] = wc
7         for (word, count) in wc.items():
8             apcount.setdefault(word, 0)
9
10            if count > 1:
11                apcount[word] += 1
12    except:
13        print ( 'Failed to parse feed %s' % feedurl)
```

Listing 4: Extract Words from Title

```
1 wordlist = []
2 for (w, bc) in apcount.items():
3     frac = float(bc) / len(feedList)
4
5     if(frac > .005 and frac < .5 and len(wordlist) < 1000):
6         wordlist.append(w)
```

Listing 5: Extract Highest Terms

```
1 dataFile.write('Blog')
2 for word in wordlist: dataFile.write('\t%s' % word)
3 dataFile.write('\n')
4 for (blog, wc) in wordcounts.items():
5     dataFile.write(blog)
6     print(blog)
7     for word in wordlist:
8         if word in wc:
9             dataFile.write('\t%d' % wc[word])
10        else:
11            dataFile.write('\t0')
12    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.

```
1 clust=clusters.hcluster(data)
2 asciiFile = open("ascii.txt", 'w')
3 orig_stdout = sys.stdout
4 sys.stdout = asciiFile
5
6 clusters.printclust(clust, labels=blognames)
7
8 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 `drawdendrogram` 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.

```
1
2 clusters.drawdendrogram(clust, blognames, jpeg='blogclust.jpg')
```

Listing 8: ASCII File Generation

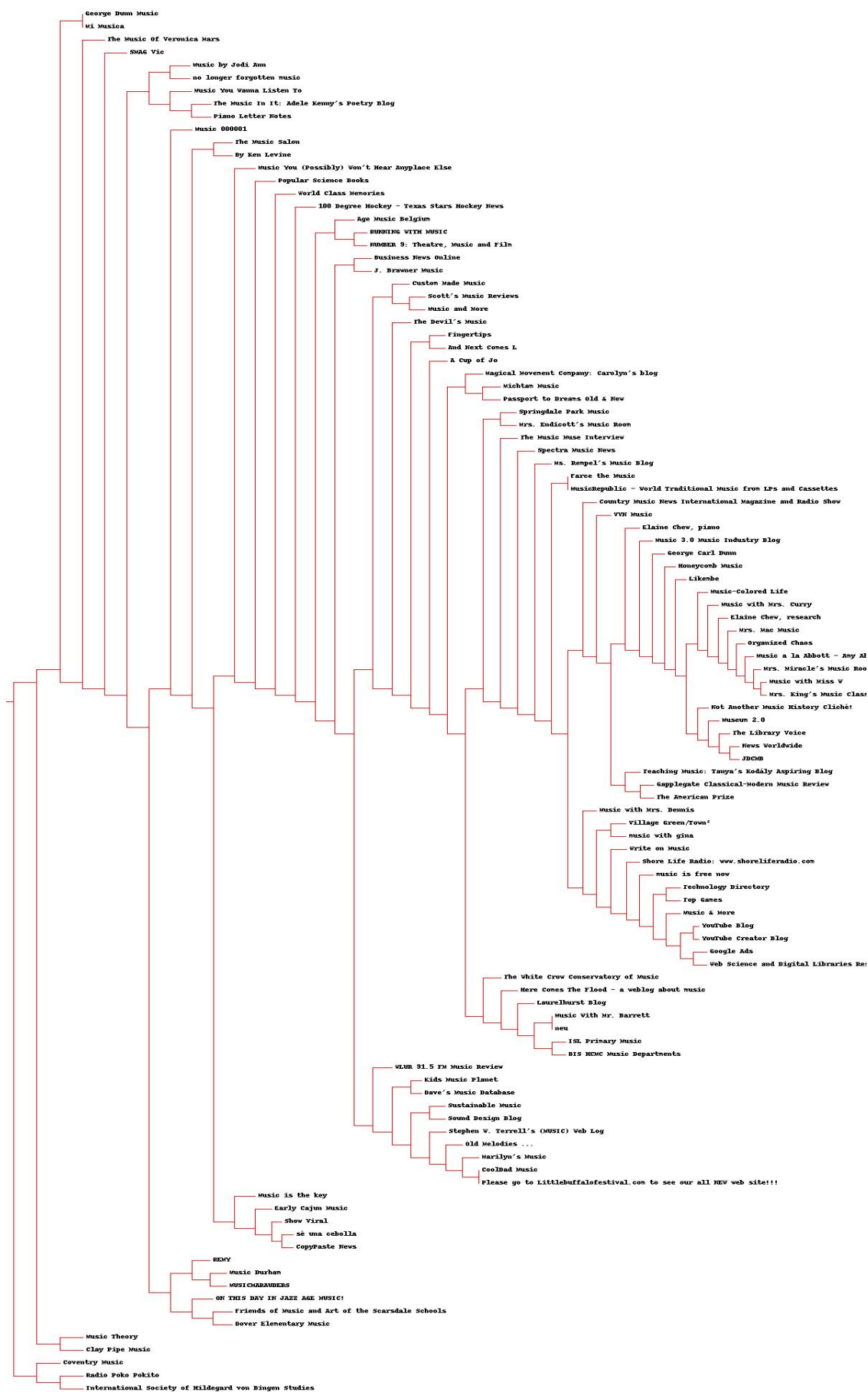


Figure 0.1: Dendrogram

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.

```
1 kclust=clusters.kcluster(data,k=5)
2 kclust=clusters.kcluster(data,k=10)
3 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.

```
1 for x in range(0,5):
2
3     for r in kclust[x]:
4         centralFile.write(blognames[r] + '\n')
5         centralFile.write('\n')
6     centralFile.write('\n')
7     centralFile.write("_____")
8     centralFile.write('\n')
```

Listing 10: Clustering into 5 groups

| Clustered Blogs in 5 Groups | | | | |
|--|---|--|---|--|
| Cluster 1 | Cluster 2 | Cluster 3 | Cluster 4 | Cluster 5 |
| Kids Music Planet Music Theory | Music With Mr. Barrett ISL Primary Music | CoolDad Music George Dunn Mu- sic | Farce the Music Technology Direc- tory | A Cup of Jo The Devil's Music |
| Music and More Age Music Belgium Mi Musica | Top Games Music with Miss W RUNNING WITH MU- SIC | Mrs. Mac Music News Worldwide Music Durham | VVN Music Clay Pipe Music The Music Salon | Michtam Music Custom Made Music Popular Science Books |

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

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 reduced. In this case, it took 290 iterations until the amount of error had stabilized, and then the MDS was drawn.

```
1 coords=clusters.scaledown(data)
2 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, “Web Science and Digital Libraries” appears near “The Library Voice”, 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, dendrogram, and ASCII files were generated in the same manner as the original URIs.

It is evident when comparing the dendrograms 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 dendrogram 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 dendrogram because there is a lack of URIs which relate to data science, therefore eliminating 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 surrounded by blogs such as "Googld Ads" and "Technology Directory". The same could be said for the 2dblog relating to the original URIs, but it is evident that, again, the new 2dblog 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.

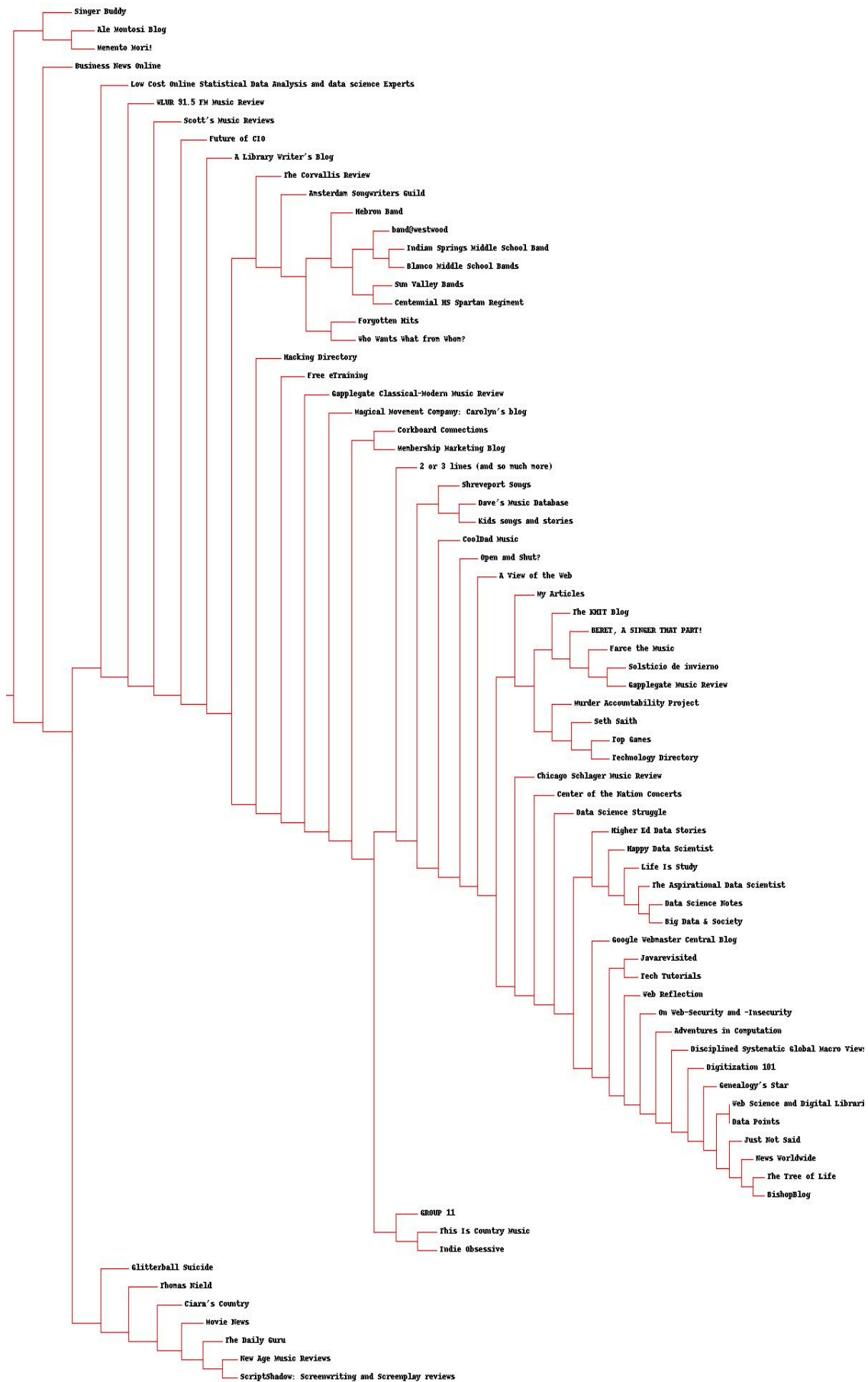


Figure 0.3: Extra Credit Dendrogram

