

# Assignment 8

ROSS REELACHART

PROFESSOR MICHAEL L. NELSON

## Question 1

First I needed to gather 100 random blogs, including the two blogs that were given by the assignment. Starting from the given random blogger URI, I gathered the first 98 random blogs that the URI provided in the python file *get100Blogs.py*. This list of 100 blogs was stored in the file *100BlogsUris*. I ran this program a few times, and got different lists each time.

```
def get100Blogs():
    link = "http://www.blogger.com/next-blog?navBar=true&blogID=3471633091411211117"
    set = Set()

    while len(set) < 100:
        r = requests.get(link, allow_redirects=True)
        uri = r.url

        if len(uri) > 0:
            uri = uri.lower()

            parsedUrl = urlparse.urlparse(uri)
            parsedUrl = parsedUrl.scheme + '://' + parsedUrl.netloc + '/'

            set.add(parsedUrl)
            print len(set)
            print parsedUrl

    return set

def writeFile(data):
    file = open('100BlogUris', 'w')
    file.write('http://f-measure.blogspot.com/' + '\n')
    file.write('http://ws-dl.blogspot.com/' + '\n')

    for item in data:
        file.write(item + '\n')

data = get100Blogs()
writeFile(data)
```

Figure 1 - The code for *get100Blogs.py*. Note that the *'writeFile'* function appends the given two blogs to the existing list of 100.

Second I needed to gather the atoms from each blog. To set that up, I made the small python program *getAtoms.py* that took the 100 blog URIs and appended "feeds/posts/default?max-results=1000" to each URI. This updated list of URIs were put into the file *atomsFromBlogs*. (Yes, there is a persistent typo in "Froms".)

The next step involved downloading the *generateFeedVector.py* program from the Programming Collective Intelligence book, which also necessitated I download *feedparser.py* since *generateFeedVector* used that library. (This and *clusters.py* were found on github.<sup>1</sup>)

Now, this code did not account for blogs having multiple pages. I added a function that used BeautifulSoup to search for "rel="next"," which indicated that more pages existed. All of the URIs of the pages associated with a blog were stored in the variable *blogPageList*, which was then used to find all the words and word counts from that entire blog. All of these functions were placed into the program

*getBlogTermMatrix.py*, which generated the final *blogTermMatrix.txt* file which held all of the blogs and the 1000 most frequently-used terms across all of them. Also, *stopWordList.txt* was used to keep the program from counting frequent but unimportant words from clogging up my 1000 terms.

```
def getNextBlogPage(url, blogPageList=[]):
    req = requests.get(url)
    soup = BeautifulSoup(req.text)
    nextLink = soup.find('link', rel='next', href = True)
    if nextLink is not None:
        nextLink = nextLink['href']
        blogPageList.append(nextLink)
        getNextBlogPage(nextLink, blogPageList)
    return blogPageList
```

Figure 2 - The selection of code from *getBlogTermMatrix.py* that accounted for blogs with multiple pages.

This process initially took over two hours, and didn't even complete. Then I re-ran the program, and it only took 45 minutes to complete. I attribute this speed-up to my decision to use a modified list of 100 blogs, taken from the original lists I generated, that left out most of the gigantic music-related blogs that were holding up my program.

(Continued on next page)

## Question 2

Generating the ASCII and JPEG dendrogram were generated by using the code provided from week 12 class slides 12 and 13<sup>2</sup>. In order to get those to work, however, I needed to download *clusters.py* from the PCI book. The ASCII dendrogram was generated and redirected into *AsciiDendrogram.txt* for storage and viewing. The dendrograms were generated from the *blogTermMatrix.txt* file. The full dendrograms can be viewed on the github<sup>3</sup>.

```
1 -
2
3 -
4     Me fala uma música boa aí
5     -
6         Green Eggs and Ham Mondays 8-10am
7         -
8             A H T A P O T
9             IoTube      :)
10    -
11    -
12    -
13        Desolation Row Records
14    -
15        @65 Sounding Booth
16        ΜΕΕΑ ΣΤΗ ΒΡΩΜΙΑ
17    -
18        Abu Everyday
19    -
20        Chemical Robert!
21    -
22        Who needs a TV?
23        Parish Radio
24    -
25    -
26        *Sixeyes: by Alan Williamson
27    -
28        The Jeopardy of Contentment
29    -
30        Stereo Pills
31        The Stark Online
32    -
```

Figure 3 - A sample of the ASCII dendrogram.

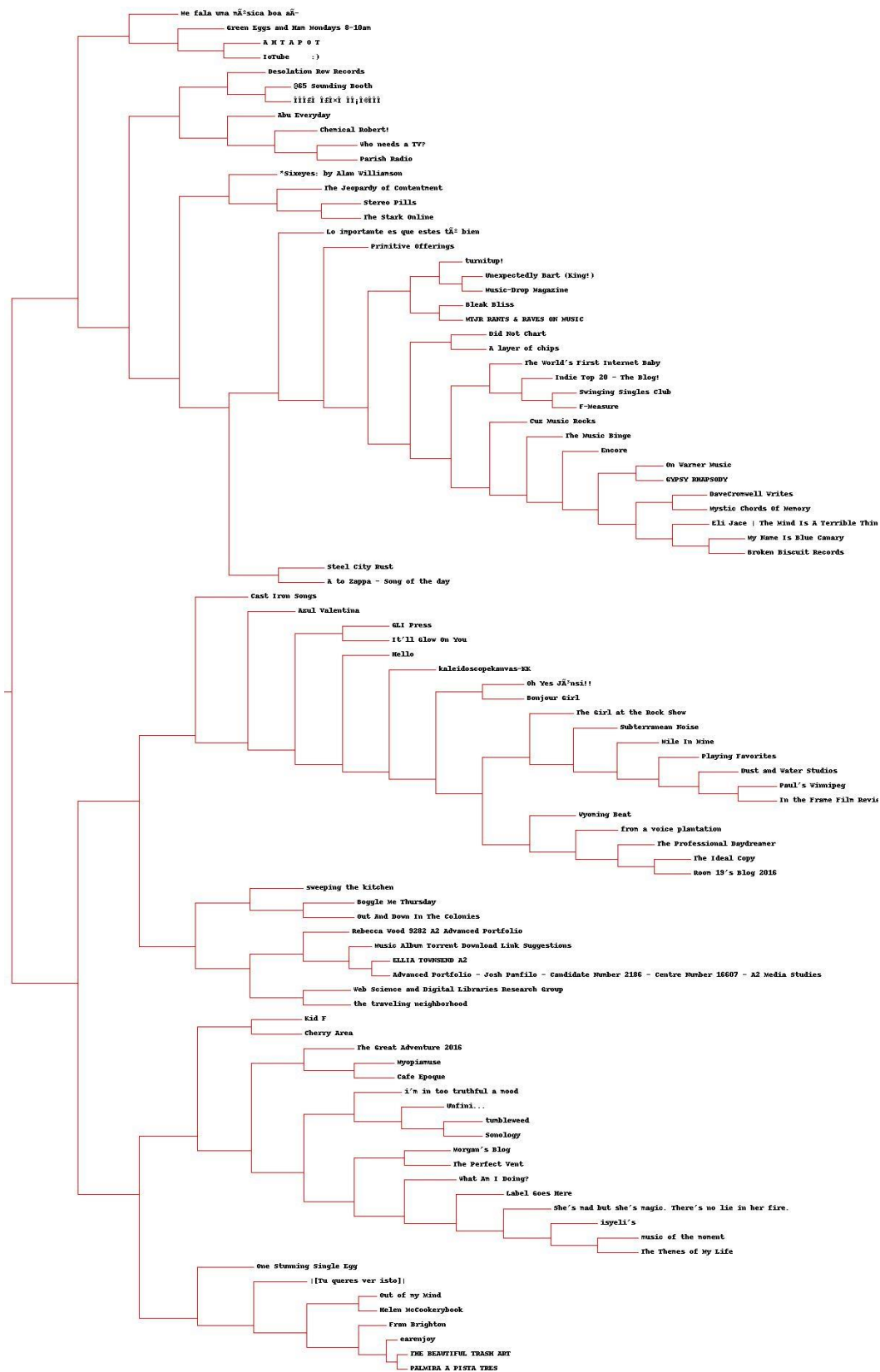


Figure 4 - The complete JPEG dendrogram.

### Question 3

Again, *clusters.py* was used in code mostly taken from week 12 class slide 18<sup>2</sup>, modified with print functions and then redirected into the *KMeansAndIterations.txt* file. For centroid values of 5, 10, and 20; 6, 4, and 5 iterations were needed respectively. The K-Means clusters were generated from *blogTermMatrix.txt*.

```
Centroid Value: 5
Iteration 0
Iteration 1
Iteration 2
Iteration 3
Iteration 4
Iteration 5
Iteration 6
['Steel City Rust', 'Cast Iron Songs', '||[Tu quieres ver isto]||', 'The Great A
Blog 2016', 'Broken Biscuit Records', 'The Professional Daydreamer', 'Azul Va
['Bleak Bliss', 'MTJR RANTS & RAVES ON MUSIC', 'Unexpectedly Bart (King!)', '
'F-Measure']
['Cuz Music Rocks', 'On Warmer Music', "Paul's Winnipeg", 'Oh Yes J\%c3%b3ns
Magazine', 'Eli Jace | The Mind Is A Terrible Thing To Paste', 'The Jeopardy
'Dust and Water Studios', 'Encore', 'In the Frame Film Reviews', "i'm in too
['Music Album Torrent Download Link Suggestions', 'ELLIA TOWNSEND A2', 'Web S
9282 A2 Advanced Portfolio', 'Myopiamuse', 'the traveling neighborhood', 'tum
'Parish Radio', 'earenjoy', 'Cherry Area', 'Advanced Portfolio - Josh Pamfilo
fire.". "isaveli's". 'Label Goes Here'. 'PALMIRA A PISTA TRES']
```

Figure 5 - A sample of the *KMeansAndIterations.txt* file.

## Question 4


Once more, *clusters.py* was used in the generated the Multidimensional Scaling visualization of the blogs. Using code from slide 28<sup>2</sup> and the *blogTermMatrix.txt* file, the JPEG was created. Using redirection, I stored the readout of all the iterations values in *mdsIterations.txt*. Starting from an error value of about 4341, the program reduced the error to about 2833.826 after **153** iterations. The full visualization can be viewed on the github<sup>3</sup>.



Figure 6 - The complete MDS visualization.

## Question 5

Using the formula for TFIDF used in assignment 3<sup>4</sup>, except actually written into the code in a loop for *atomsFromBlogs*, I found the TFIDF score for each term found in the 100 blogs.



```
for word in wordlist:
    if word in wc:
        #Calculate the TF for a particular word
        termFrequency = wc[word]/float(len(wc))

        #Calculate the Inverse Document Frequency
        inverseDocumentFrequency = logBase2(iteration/float(apcount[word]))

        #Calculate the TFIDF
        tfIdf = termFrequency*inverseDocumentFrequency

        out.write('\t%f' % tfIdf)
    else:
        out.write('\t0')
out.write('\n')
```

Figure 7 - The coded formula for TFIDF in *getTFIDF.py*

The results of the modified *getBlogTermMatrix.py* program, which is now called *getTFIDF.py*, were stored in *blogTermMatrixTFIDF.txt*. That output was then run through the same JPEG generation program from question 2. The full dendrogram can be viewed on the [github](#)<sup>3</sup>.

A lot changed when the dendrogram was arranged by TFIDF instead of raw term frequency. It's difficult to see the differences with so many different blogs changing places, sometimes going from the top of the dendrogram to the bottom or vice versa. I did, however, notice a larger change when I layered the two dendrograms on top of each other and reduced the opacity on one of them. I noticed that the majority of large changes occurred in the top half of the dendrogram, with smaller changes happening in the lower half.

(Continued on next page.)



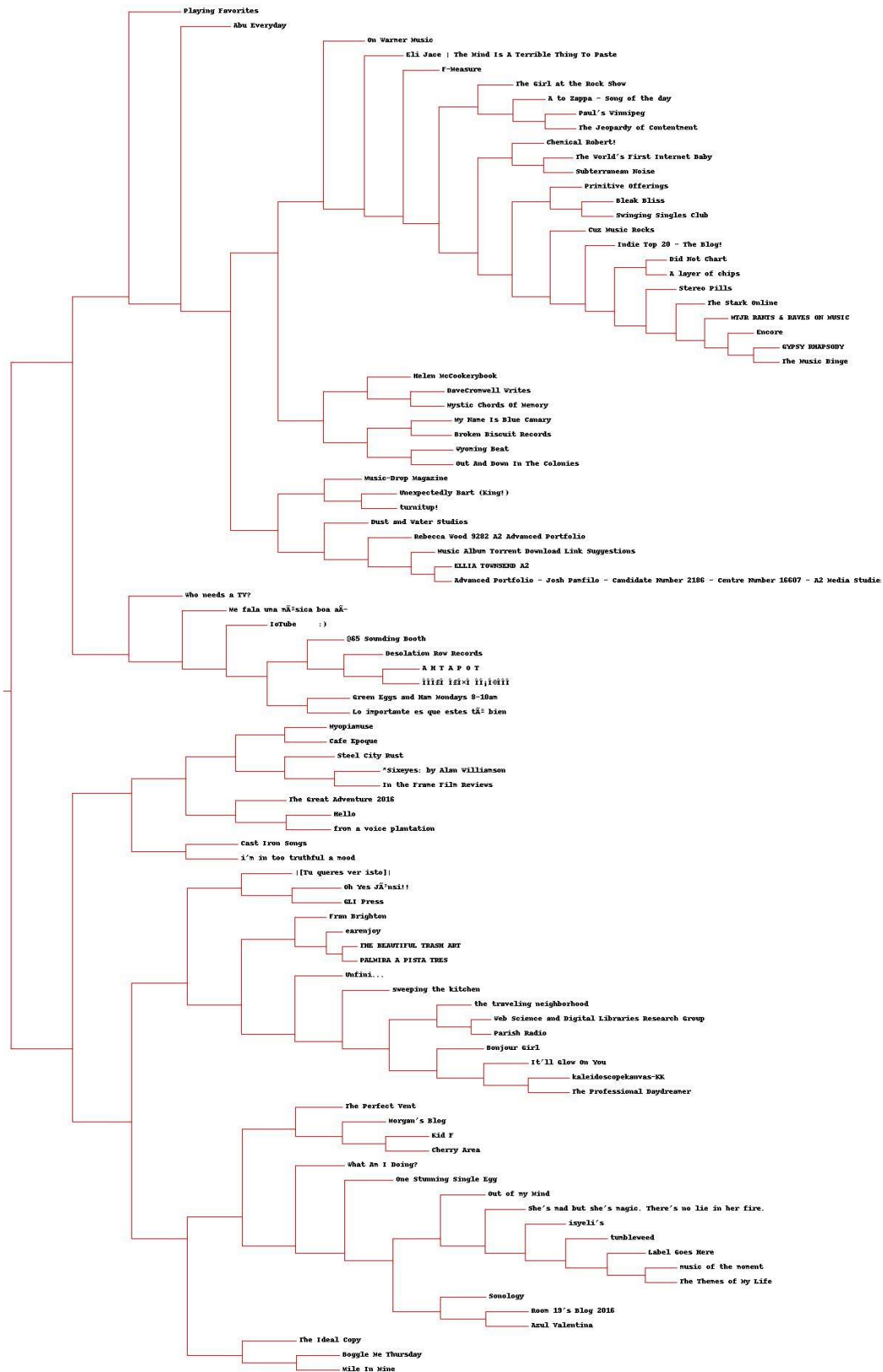


Figure 8 - The JPEG dendrogram resulting from using TFIDF instead of term frequency



## References

- 1.) <https://github.com/uolter/PCI/tree/master/chapter3>
- 2.) <https://github.com/phonedude/cs532-s17/blob/master/slides/week-12-clustering.ppt>
- 3.) <https://github.com/rreelachart/cs532-s17/tree/master/submissions/assignment%208/Images%20and%20Output>
- 4.) <https://github.com/rreelachart/cs532-s17/tree/master/submissions/assignment%203>