Assignment 3

CS 532: Introduction to Web Science

Spring 2017

Grant Atkins

Finished on February 23, 2017

1

Question

```
1.  Download the 1000 URIs from assignment #2.  "curl", "wget", or
"lynx" are all good candidate programs to use.  We want just the
raw HTML, not the images, stylesheets, etc.

from the command line:

% curl http://www.cnn.com/ > www.cnn.com
% wget -O www.cnn.com http://www.cnn.com/
% lynx -source http://www.cnn.com/ > www.cnn.com

"www.cnn.com" is just an example output file name, keep in mind
that the shell will not like some of the characters that can occur
in URIs (e.g., "?", "&").  You might want to hash the URIs, like:

% echo -n "http://www.cs.odu.edu/show_features.shtml?72" | md5
41d5f125d13b4bb554e6e31b6b591eeb

("md5sum" on some machines; note the "-n" in echo -- this removes
the trailing newline.)

Now use a tool to remove (most) of the HTML markup.  "lynx" will
do a fair job:

% lynx -dump -force_html www.cnn.com > www.cnn.com.processed

Use another (better) tool if you know of one.

A "better" approach is to use BeautifulSoup, see:

http://stackoverflow.com/questions/1936466/beautifulsoup-grab-visible-webpage-text

for some hints on how to start.  Note that none of these methods
are going to be perfect.

Keep both files for each URI (i.e., raw HTML and processed).
Upload both sets of files to your github account.
```

Answer

To handle the first part of this problem, downloading the 1000 URIs collected from Assignment #2, I decided to write a shell script as shown in Listing 1. The script first starts by creating directories if the directory is not found and a CSV file to store key pairs of URI and md5 hash value calculated later. It then iterates through each URI in my collection and is eventually saved to a folder containing all 1000 URIs html content.

For each URI in the collection it will:

1. Create an md5 hash for the URI
2. Add the URI and md5 hash to a CSV
3. Perform a curl HTTP get request to get the html content
4. Save the html content to file named by the md5 hash and ".html" extension

It should be noted that the curl HTTP get request used the User-Agent "Mozilla/5.0" along with the *-L* and *-m* arguments. I decided to add -L, which follows redirects, to this script because I noticed that some links have actually already changed locations and resulted in a 303 response, location change, when requested. I also added the -m argument, which sets the maximum time a connection can last, mainly due to the fact that some of these URIs were actually streaming data like live music or some radio station and it would continually retrieve data [8]. I set the maximum time to 3 seconds to retrieve the necessary information

```
1  #!/bin/bash
2
3  # create doirectories if not there
4  script_dir=$(dirname $0)
5
6  dir="$script_dir/output/html"
7  if [ ! -d "$dir" ]; then
8          mkdir $dir
9  fi
10
11 csv="$script_dir/output/md5Mapping.csv"
12 if [ ! -f "$csv" ]; then
13         touch $csv
14 fi
15
16
17 for uri in $(cat $script_dir/output/finalURIs.txt)
```

```
do
        # completed md5 on macbook
        hashedURI=$(echo -n $uri | md5)
        outputFile="$script_dir/output/html/$hashedURI.html"

        $(echo "$uri,$hashedURI" >> $csv)
        $(curl -L -m 3 -A "Mozilla/5.0" $uri > $outputFile)
done
```

Listing 1: Shell script for downloading 1000 URI html content

For the second part of this problem I decided to write a script in python 3.6 using the dependency BeautifulSoup for html parsing. The script starts by iterating through the files in the html directory created in part 1 of this problem. Using code provided from a Stackoverflow.com post, I created a list of lines that were derived from the encapsulated text in each html element []. I would then iterate through each of these lines and saved them to a new file with the same md5 hash name as the file it received this information from, this time saving it with the ".txt" extension. If the lines created were blank I ignored them and didn't add them to this new file. This script, processHtml.py, is shown in Listing 2.

As mentioned before some of these websites were actually streaming data which also resulted in the html content to not always be the same encoding type. A majority of the documents used UTF-8 encoding while some didn't. Therefore to compensate for this, I checked the exception that the text retrieved might not be UTF-8 and simply discarded it if it was not.

```
import os
from bs4 import BeautifulSoup
import re
import codecs


def visible(element):
    if element.parent.name in ['style', 'script', '[document]',
        'head', 'title']:
        return False
    elif re.match('<!--.*-->', str(element)):
        return False
    return True


def saveProcessed(filename,line):

    filename = "output/processed/"+filename+".txt"
    if not os.path.exists("output/processed"):
```

```
19          os.makedirs("output/processed")
20
21      # if not found, create
22      try:
23          with open(filename, 'a') as file:
24              file.write(line+"\n")
25      except (IOError,ValueError):
26          with open(filename, 'w') as file:
27              file.write(line+"\n")
28
29
30  def processHtml():
31      for filename in os.listdir("output/html"):
32          print(filename)
33          with codecs.open("output/html/"+filename, "r",encoding='
                utf-8', errors='surrogateescape') as fdata:
34
35              soup = BeautifulSoup(fdata, 'html.parser')
36
37              texts = soup.findAll(text=True)
38              visible_texts = list(filter(visible, texts))
39              for item in visible_texts:
40
41                  item = (item.strip())
42                  if len(item) != 0:
43                      try:
44                          print(item)
45                          md5name = filename[:-5]
46                          saveProcessed(md5name,item)
47                      except UnicodeEncodeError:
48                          # skip bad encodings
49                          print("skipped bad utf-8 encoding")
50
51
52  if __name__ == "__main__":
53      processHtml()
```

Listing 2: Python script for removing duplicates in data files

2

Question

2. Choose a query term (e.g., "shadow") that is not a stop word (see week 5 slides) and not HTML markup from step 1 (e.g., "http") that matches at least 10 documents (hint: use "grep" on the processed files). If the term is present in more than 10 documents, choose any 10 from your list. (If you do not end up with a list of 10 URIs, you've done something wrong).

As per the example in the week 5 slides, compute TFIDF values for the term in each of the 10 documents and create a table with the TF, IDF, and TFIDF values, as well as the corresponding URIs. The URIs will be ranked in decreasing order by TFIDF values. For example:

Table 1. 10 Hits for the term "shadow", ranked by TFIDF.

```
TFIDF TF IDF URI
-------------
0.150 0.014 10.680 http://foo.com/
0.044 0.008 10.680 http://bar.com/
```

You can use Google or Bing for the DF estimation. To count the number of words in the processed document (i.e., the deonminator for TF), you can use "wc":

```
% wc -w www.cnn.com.processed
    2370 www.cnn.com.processed
```

It won't be completely accurate, but it will be probably be consistently inaccurate across all files. You can use more accurate methods if you'd like, just explain how you did it.

Don't forget the log base 2 for IDF, and mind your significant digits!

https://en.wikipedia.org/wiki/Significant_figures#Rounding_and_decimal_places

Answer

The query term I chose for this problem was *California*. Since most of my URIs were retrieved with relevance to Jazz or Funk music, I wanted to see if the location California was a hot term in these pages.

I then wrote a shell script to find all the files containing the query term as shown in Listing 3. The script find's the first 10 files that that contains the query term in the processed text files shown on line 7. I observed that out of the 1000 files there were 28 files with the term *California*. It then saves the ten mdf5 hashes of the files found to a text file for reference and then also saves the headers "TFIDF, TF, IDF, URI" to a CSV. Using the ten files found it will then calculate the number of occurrences *California* using the the grep and wc commands, shown on lines 32-33. These would then be used to calculate the TF for this problem as shown below:

$$TF(California, doc) = \frac{termCount(California, doc)}{wordCount(doc)}$$

Then using the md5 hash name, I searched the csv created earlier to find the URI name for reference. I then used a variable IDF of 4.9412, derived using the following:

$$IDF(California, Corpus) = \log_2\left(\frac{51,000,000,000}{1,660,000,000}\right) = \log_2(30.72289) = 4.9412420$$

The value 51B was retrieved from Google's estimated index size on `worldwidewebsize.com` for the current month, February 2017 [7]. The value 1.66B was retrieved from a query of *California* in Google's search engine, shown in Figure 1.

After calculating the TF and IDF, the TFIDF could be calculated simply by multiplying the two values together as shown below. After everything was calculated I saved each of these values and their respective URIs to a CSV.

$$TFIDF(California, doc, Corpus) = TF(California, doc) * 4.9412$$

The results for the 10 URIs is shown in Table 1.

```
1 #!/bin/bash
2
3 # src directory
4 script_dir=$(dirname $0)
```

```
5
6  queryTerm="California"
7  grepList=$(grep "$queryTerm" $script_dir/output/processed -rli |
       head -n 10)
8
9  # save 10 found
10 tenFromQuery=$(echo $script_dir/output/tenFromQuery.txt)
11
12 if [ -f $tenFromQuery ]; then
13         echo "FILE EXISTS: $tenFromQuery"
14         rm "$tenFromQuery"
15 fi
16
17 $(echo "$grepList" | sed 's/\(.*\).\/output\/processed\//\1 /'
       >> "$script_dir/output/tenFromQuery.txt")
18
19 # csv setup for output
20 csv=$(echo $script_dir/output/tfidf.csv)
21
22 if [ -f $csv ]; then
23         echo "FILE EXISTS: $csv"
24         rm "$csv"
25 fi
26
27 $(echo "TFIDF, TF, IDF, URI" >> $csv)
28
29 # loop through 10 items
30 for item in $grepList
31 do
32         wordCount=$(grep -io "$queryTerm" $item | wc -l | bc)
33         totalWords=$(wc -w < $item | bc)
34
35         md5hash=$(echo $item | sed 's/\(.*\).\/output\/processed
               \//\1 /' | sed 's/\(.*\).txt/\1 /')
36         # commas not legal in filenames so just search for first
               comma delimiter
37         uri=$(grep $md5hash "$script_dir/output/md5Mapping.csv"
               | sed 's/,.*//')
38         echo $uri
39
40         echo "WordCount = $wordCount   totalWords= $totalWords"
41
42         # TF-IDF = TF    IDF
43         # = occurrence in doc / words in doc
44         # log2(total docs in corpus / docs with term)
45         # source http://www.worldwidewebsize.com/
46         # 51 billion pages indexed by google
47         # 1.66 billion results for 'California'
48
```

```
49          idf=4.9412
50          tf=$(echo "scale=5; $wordCount / $totalWords" | bc)
51          tfidf=$(echo "scale=5; $tf * $idf" | bc)
52
53          $(echo "$tfidf,$tf,$idf,$uri" >> $csv)
54  done
```

Listing 3: Shell script to compute tfidf

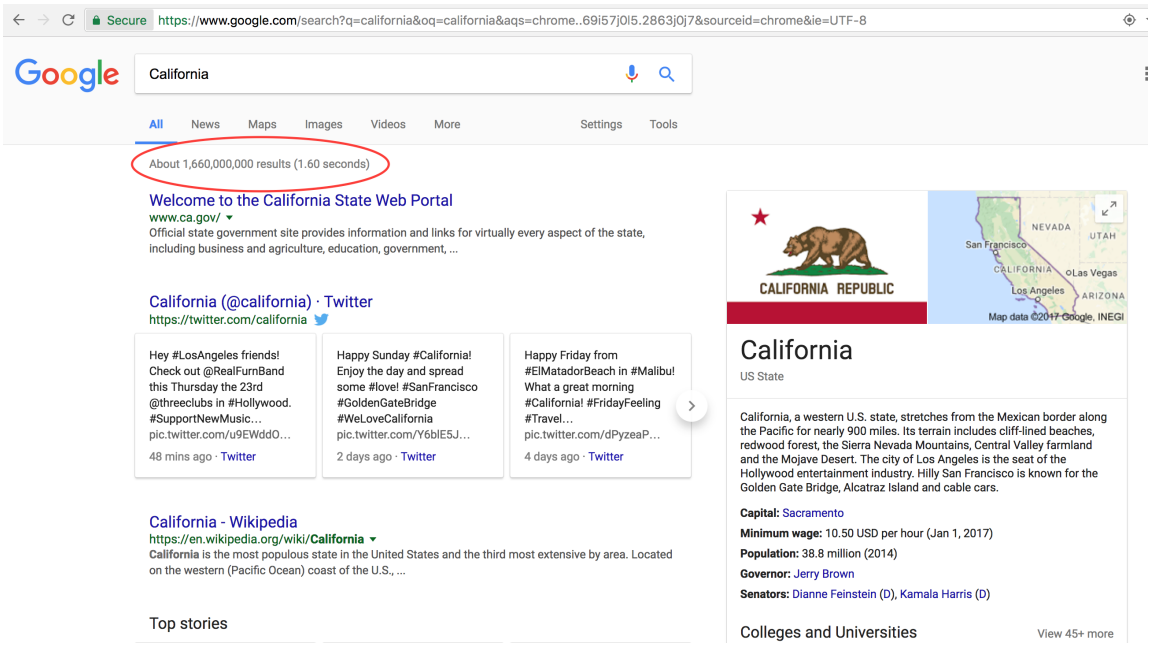

Figure 1: Results from Google query of *California*

TFIDF	TF	IDF	URI
0.00741	0.00150	4.9412	https://www.youtube.com/watch?v=_6mtcKN1o-E
0.00276	0.00056	4.9412	http://www.patheos.com/blogs/poptheology/2017/01/shuffled-selections-january-22-28/?utm_campaign=shareaholic&utm_medium=twitter&utm_source=socialnetwork
0.00380	0.00077	4.9412	http://www.lapetitemortgallery.com/nyc-artist-james-brown-drawing/
0.00143	0.00029	4.9412	http://www.newslocker.com/en-us/music/r-and-b-hip-hop-news/see-this-amazing-nj-teen-blend-hop-hop-jazz-and-rampb-on-youtube-njcom/
0.00642	0.00130	4.9412	https://www.youtube.com/watch?v=jbe_1sNGQ2o
0.00424	0.00086	4.9412	http://www.lctmag.com/operations/news/719804/empirecls-adds-mci-luxury-buses-with-3-point-seatbelts?utm_source=dlvr.it&utm_medium=twitter
0.00400	0.00081	4.9412	https://www.talkbass.com/threads/fender-marcus-miller-mij-jazz-with-j-east-preamp.1266114/?utm_source=dlvr.it&utm_medium=twitter
0.00168	0.00034	4.9412	https://www.theatlantic.com/entertainment/archive/2017/01/missy-elliott-im-better-jamiroquai-automaton-videos/514736/?utm_source=twb
0.00434	0.00088	4.9412	http://thesop.org/story/20170131/judyth-piazza-interviews-renowned-jazz-vocalist-jan-daley.html
0.00889	0.00180	4.9412	https://www.youtube.com/watch?v=C7uYx1J0hzU

Table 1: 10 URIs found containing *California*, with calculations TFIDF, TF and IDF

3

Question

3. Now rank the same 10 URIs from question #2, but this time by their PageRank. Use any of the free PR estimaters on the web, such as:

http://pr.eyedomain.com/
http://www.prchecker.info/check_page_rank.php
http://www.seocentro.com/tools/search-engines/pagerank.html
http://www.checkpagerank.net/

If you use these tools, you'll have to do so by hand (they have anti-bot captchas), but there are only 10 to do. Normalize the values they give you to be from 0 to 1.0. Use the same tool on all 10 (again, consistency is more important than accuracy). Also note that these tools typically report on the domain rather than the page, so it's not entirely accurate.

Create a table similar to Table 1:

Table 2. 10 hits for the term "shadow", ranked by PageRank.

PageRank	URI
0.9	http://bar.com/
0.5	http://foo.com/

Briefly compare and contrast the rankings produced in questions 2 and 3.

Answer

Using the `pr.eyedomain.com` website I checked the page rank of my 10 URIs obtained from question 2 [1]. This website only did domain searches on the URIs provided, making the 3 youtube URIs included the same value for each Page Rank. I attempted to use the seocentro page rank but it was prompting me to pay after 3 attempts so I stuck with `pr.eyeomdain.com`. I also attempted prchecker.info and checkerpagerank.net but was unable to get past their captchas. The end result is shown in Table 2.

When comparing the Page Rank to the TFIDF for each of the URIs, it's apparent that Page Rank is a much higher value for each of the URIs. What was surprising to me was the difference in TFIDF but also the values of the youtube URIs. The youtube URIs tend to correlate as the top 3 highest TFIDF values as well as the top Page Rank values. The terms and the TFIDF for majority of the values actually correlates very nicely with one another. However the biggest discrepancy was the Page Rank and TFIDF for theatlantic.com. The Page Rank of theatlantic.com was 0.8 but it also had the second lowest TFIDF. Upon visiting the website personally I searched for my query term *California* only to find it in a dropdown on the page showing that TFIDF isn't as reliable compared to Page Rank. This URI turned out to be a very high indexed news website, but TFIDF wasn't reliable enough to receive great values on this URI.

PR	TFIDF	URI
0.9	0.00741	https://www.youtube.com/watch?v=_6mtcKN1o-E
0.6	0.00276	http://www.patheos.com/blogs/poptheology/2017/01/shuffled-selections-january-22-28/?utm_campaign=shareaholic&utm_medium=twitter&utm_source=socialnetwork
0.4	0.00380	http://www.lapetitemortgallery.com/nyc-artist-james-brown-drawing/
0.2	0.00143	http://www.newslocker.com/en-us/music/r-and-b-hip-hop-news/see-this-amazing-nj-teen-blend-hop-hop-jazz-and-rampb-on-youtube-njcom/
0.9	0.00642	https://www.youtube.com/watch?v=jbe_1sNGQ2o
0.5	0.00424	http://www.lctmag.com/operations/news/719804/empirecls-adds-mci-luxury-buses-with-3-point-seatbelts?utm_source=dlvr.it&utm_medium=twitter
0.4	0.00400	https://www.talkbass.com/threads/fender-marcus-miller-mij-jazz-with-j-east-preamp.1266114/?utm_source=dlvr.it&utm_medium=twitter
0.8	0.00168	https://www.theatlantic.com/entertainment/archive/2017/01/missy-elliott-im-better-jamiroquai-automaton-videos/514736/?utm_source=twb
0.4	0.00434	http://thesop.org/story/20170131/judyth-piazza-interviews-renowned-jazz-vocalist-jan-daley.html
0.9	0.00889	https://www.youtube.com/watch?v=C7uYx1J0hzU

Table 2: Page Rank and TFIDF Comparison of 10 URIs with PR on a 0 to 1 scale

4

Question

```
==================================================
======Question 4 is for 3 points extra credit=======
==================================================

4.  Compute the Kendall Tau_b score for both lists (use "b" because
there will likely be tie values in the rankings).  Report both the
Tau value and the "p" value.

See:
http://stackoverflow.com/questions/2557863/measures-
of-association-in-r-kendalls-tau-b-and-tau-c
http://en.wikipedia.org/wiki/Kendall_tau_rank_correlation_coefficient#Tau-b
http://en.wikipedia.org/wiki/Correlation_and_dependence
```

Answer

Approaching this problem I had little knowledge of what the purpose of the Kendall Tau_b score. Although I still feel uneasy about a few parts of the equation, as shown below, I felt I got a general knowledge of what its result yields. The following Kendall Tau_b equation was taken from Wikipedia [3].

$$\tau_b = \frac{n_c - n_d}{\sqrt{(n_0 - n_1)(n_0 - n_2)}}$$

where

$$n_0 = n(n-1)/2$$

$$n_1 = \sum_i t_i(t_i - 1)/2$$

$$n_2 = \sum_j u_j(u_j - 1)/2$$

$$n_c = \text{Number of concordant pairs}$$

$$n_d = \text{Number of discordant pairs}$$

$$t_i = \text{Number of tied values in the } i^{th} \text{ group of ties for the first quantity}$$

$$u_j = \text{Number of tied values in the } j^{th} \text{ group of ties for the second quantity}$$

Basically this creates a correlation between the paired values. The *tau* value can have a coefficient between -1 to 1. Any value less that 1, meant there is a negative correlation and that page rank is the opposite of the TFIDF value. Any value greater than 1 means that there is a positive correlation showing that the correlation between page rank and TFIDF are similar, if its at 1 it represents a perfect match. If the value is at zero or very close to zero this means that there is no correlation between the page rank and TFIDF. I stated before that I thought they actually correlated fairly well, and Dr. Nelson would say, "That's an engineer's answer."

Therefore using the stackoverflow post mentioned in the question, I found an R library called Kendall which computes the Kendall Tau_b value [6]. Using the table from question 3, I made a CSV file to hold this data called **pageRankTfidf.csv**. I then imported this data into R to use. The code to generate this Kendall Tau_b value is below in Listing 4.

```
1 library(Kendall)
2 setwd(getwd())
3 # csv dataframe of PR, TFIDF, URI
4 csv <- read.table('output/pageRankTfidf.csv',header = TRUE,sep =
      ",")
5
6 ken <- Kendall(csv$PR,csv$TFIDF)
7 summary(ken)
```

Listing 4: R script to calculate Kendall Tau_b between page rank and TFIDF from previous 10 URIs

The summary produced:

```
Score =  19 , Var(Score) = 117.6667
denominator =  41.89272
tau = 0.454, 2-sided pvalue =0.097039
```

The tau was above 0 making it a positive correlation and also supporting my Engineer's hypothesis that it correlates.

5

Question

```
==================================================
======Question 7 is for 2 points extra credit=======
==================================================

7.  Build a simple (i.e., no positional information) inverted file
(in ASCII) for all the words from your 1000 URIs.  Upload the entire
file to github and discuss an interesting portion of the file in
your report.
```

Answer

To complete this question I decided not to create write my own code but base it off a sample code provided on Rosetta Code as shown in Listing 5, named **simpleInvertedIndex.py** [2]. It first goes through each processed file splitting the words and associates words with that file. After going through all 1000 processed files it would then sort each word found in any of those files and associate the word with all related the files in a key pair association, where the value was an array of the text files with the values.

This lead to some very interesting results. The keys that were most interesting were actually numbers. It was found that single character, single digit, were in the most files. This also seemed to be true for any single character in a document, such as: commas, backslashes, apostrophes, question marks, etc. As the key term was getting larger in total characters, it tended to have less and less file associations. Which to me makes sense as a majority of words in the english language don't have to cross six characters.

Years also seemed to be a great find in this inverted index. I found years starting from 2010 up to 2017, there could be more, that also had multiple occurrences in files. Older years seemed less prevalent in these documents. I found that each year after 2010 slightly went up in number of appearances in files, while 2017 at the end had the most number of occurrences.

Finally the emojis had game in these documents, but were fairly scarce but some managed to appear a in a few documents as shown in Figure 2.

```
1  """
2  TAKEN DIRECTLY FROM: http://rosettacode.org/wiki/Inverted_index#
       Simple_inverted_index
3  """
4  from pprint import pprint as pp
5  from glob import glob
6  try: reduce
7  except: from functools import reduce
8  try:    raw_input
9  except: raw_input = input
10
11
12 def parsetexts(fileglob='output/processed/*.txt'):
13     texts, words = {}, set()
14     for txtfile in glob(fileglob):
15         with open(txtfile, 'r') as f:
16             txt = f.read().split()
17             words |= set(txt)
18             texts[txtfile.split('\\')[-1]] = txt
19     return texts, words
20
```

```
21
22 def termsearch(terms): # Searches simple inverted index
23     return reduce(set.intersection,
24                   (invindex[term] for term in terms),
25                   set(texts.keys()))
26
27
28 texts, words = parsetexts()
29 print('\nTexts')
30 pp(texts)
31 print('\nWords')
32 pp(sorted(words))
33
34
35 invindex = {word:set(txt
36                          for txt, wrds in texts.items() if word
                                in wrds)
37             for word in words}
38 print('\nInverted Index')
39 pp({k:sorted(v) for k,v in invindex.items()})
```

Listing 5: Python script to create an inverted index

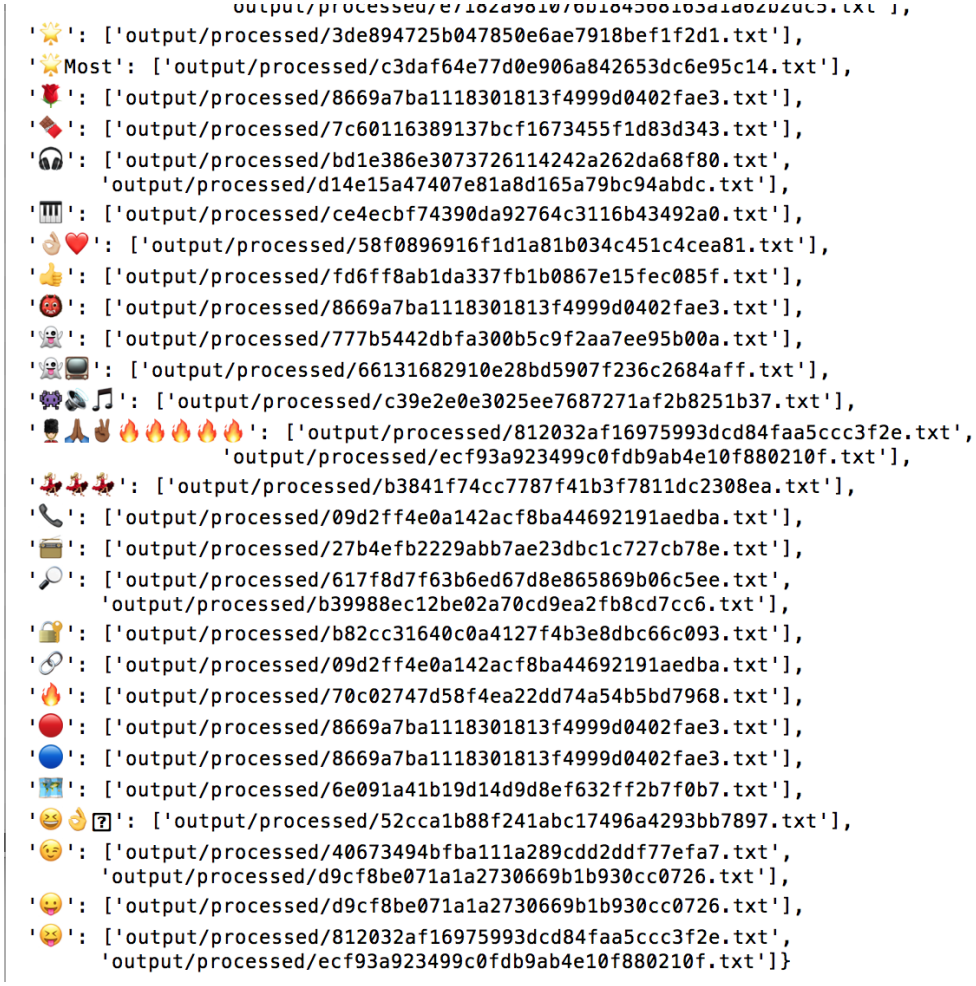

Figure 2: Inverted Index emojis found

References

[1] "Check last known Google PageRank." eyedomain. EyeDomain, n.d. Web. 22 Feb. 2017. http://pr.eyedomain.com/.

[2] "Inverted Index." Roseta Code Inverted index. N.p.,n.d. Web. 22 Feb. 2017. http://rosettacode.org/wiki/Inverted_index#Simple_inverted_index

[3] "Kendall rank correlation coefficient." Wikipedia - Kendall Rank Correlation coefficient. Wikipedia, n.d. Web. 22 Feb. 2017. https://en.wikipedia.org/wiki/Kendall_rank_correlation_coefficient#Tau-b

[4] Atkins, Grant. "finalURIs.txt - Twitter scraped URIs." cs532-s17 Github Repository. N.p., 09 Feb. 2017. Web. 22 Feb. 2017.https://github.com/grantat/cs532-s17/blob/master/assignments/A3/src/output/finalURIs.txt.

[5] Atkins, Grant. "Inverted Index results." cs532-s17 Github Repository. N.p., 09 Feb. 2017. Web. 22 Feb. 2017.https://github.com/grantat/cs532-s17/blob/master/assignments/A3/src/output/invertedIndex.txt.

[6] Fellows, Ian. "Measures of association in R ? Kendall's tau-b and tau-c." Stackoverflow. Stackoverflow, 10 April 2010. Web. 22 Feb. 2017. http://stackoverflow.com/questions/2557863/measures-of-association-in-r-kendalls-tau-b-and-tau-c.

[7] Kunder, Maurice. "The size of the Dutch World Wide Web" worldwidewebsize. N.p., n.d. Web. 22 Feb. 2017.http://www.worldwidewebsize.com/.

[8] Stenberg, Daniel. "Curl.1 the Man Page." Curl - How To Use. N.p., n.d. Web. 24 Jan. 2017. https://curl.haxx.se/docs/manpage.html.