Assignment 3

CS 532: Introduction to Web Science

Spring 2018

Hrishi Gadkari

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 for all 1000 HTML documents.
"python-boilerpipe" will do a fair job see
(http://ws-dl.blogspot.com/2017/03/2017-03-20-survey-of-5-boilerplate.html):

from boilerpipe.extract import Extractor
extractor = Extractor(extractor='ArticleExtractor', html=html)
extractor.getText()

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

Answer

For the above problem to get the raw html from the 1000 URIs which we collected from Assignment 2, I wrote a program in Python 3.5 as shown in Listing 1. The following dependencies were used for the program:

- import requests
- import os

The program makes a get request using requests.get with User-Agent as Mozilla/5.0 and then downloads the context of the page using req.text [1]. While writing the output to the file, I was getting UnicodeDecodeError which was solved by giving a parameter as encoding="utf-8' [2] to the open file function. All the raw html files are stored in **raw_html** folder.

```
import requests
import os



count = 1
#creating directory using os package
os.mkdir("raw_html")

with open('1000ulinks.txt') as fp:

        for line in fp:
        #getting the raw html for each of the 1000 URIs using
            requests
                try:
                        req = requests.get(line.strip(), timeout
                            =6, stream=True, headers={'User-Agent
                            ': 'Mozilla/5.0'})
                        string = req.text


        #writng the output as encoding 'utf-8'
                        with open('raw_html/%s.html' % count, 'w
                            ', encoding='utf-8') as outfile1:
                                outfile1.write(string)
                                outfile1.close()

                        count = count + 1

                except:
                        pass
```

Listing 1: Python program for downloading 1000 URI html content

Now to get the processed content for each of the collected raw html files, I wrote a program in Python 3.5 as shown in Listing 2. First I went through the references [5] [7] given in the question and got to know that

Python-boilerpipe is a good library for boilerplate removal and text extraction method from HTML pages. I went through the github documentation of boilerpipe [7] in order to understand how it works and implemented in my program. I installed boilerpipe using following command:

```
pip install boilerpipe3
```

The following dependencies were used for the program:

- from boilerpipe.extract import Extractor
- import os

The program iterates through each of the raw html files by reading html contents and passes to the html argument as html content, which then extracts only text from the document using getText() [7]. Although this library did not completely remove all the html content from few of the files but was sufficient for my rest of the tasks. The processed documents are stored in **processed** folder.

```
import os
from boilerpipe.extract import Extractor

# creating directory using os library
os.mkdir("processed")


count = 1

while (count < 1001):
        with open('raw_html/%s.html' % count, 'r+', encoding='utf-8') as fp:
                #reading the collected html files from previous step
                extractor = Extractor(extractor='ArticleExtractor', html=fp.read())
                #extracting non-html content
                processed = extractor.getText()
                with open('processed/%s.txt' % count, 'w', encoding='utf-8') as outfile1:
                        outfile1.write(processed)


        count = count + 1
```

Listing 2: Python program for processing all the raw html 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

For the above problem to get the documents for a query, I wrote a program in Python 3.5 as shown in Listing 3 and ran the query *football* over them. The following dependency was used:

- import re

The program iterates through each of the 1000 processed documents and reads the content of the file. It then searches through the content for the query using re.search() [8]. It then retrieved around 20 documents containing that query while chose 10 amongst them. The files are stored in **queried_docs** folder.

```
import re

count = 1

while (count < 1001):
        with open('processed/%s.txt' % count, 'r+', encoding='
            utf-8') as fp:
        #reading the conent from processed folder containing
            processed documents
                content=fp.read()
        #searching the documents for query football
                if re.search('football', content):
        #storing the documents containing the query in
            queried_docs folder
                        with open('queried_docs/%s.txt' % count,
                             'w', encoding='utf-8') as fp1:
                                fp1.write(content)


        count = count + 1
```

Listing 3: Python program to get the documents with query as football

To get the TF and IDF formulae, I went through the week 5 slides as mentioned in the question and noted the following:

$$TF = \frac{queryoccurrenceindoc}{totalnumberofindoc}$$

$$IDF = \log_2\left(\frac{Totalnumberofdocsincorpus}{docswithterm}\right)$$

I wrote a program in Python 3.5 as shown in Listing **??** to calculate the TF-IDF, TF and IDF values for the query and documents. The following dependencies were used:

- import os
- import math
- import csv
- import linecache

The program first iterates through each of the 10 documents and reads the content. Then it stores each of the words in the document in a list using *obj*.split() [9]. For calculating TF , it then uses *obj*.count(football) [9] to count the number of occurrences in the list and divides it by the total number of words in the list using len(*obj*) [9]. To get the total number of documents in the corpus.

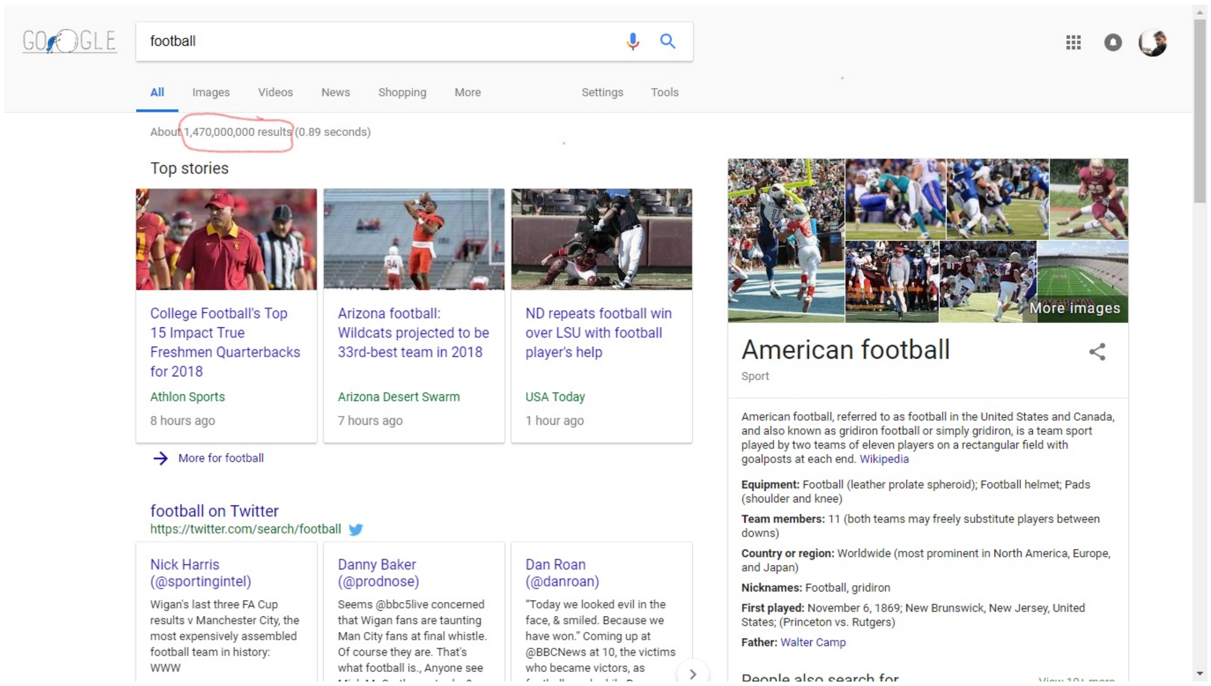
```
import math
import os
import csv
import linecache

# getting the curent directory path and appending to the
    queried_docs folder
path = os.getcwd() + '/queried_docs'
counts = 1
tuple = []
#Reading the files one by one from the directory containing 10
    URIs
for filename in os.listdir(path):
        with open('queried_docs/%s' % filename, 'r', encoding='
            utf-8') as fp:
                wordstring = fp.read()
                #creating list of words
                wordlist = wordstring.split()
                #calculating TF, IDF, TFIDF values for each
                    document
                TF = (wordlist.count('football'))/(len(wordlist)
                    )
                DF = 47000000000/1470000000
                IDF = math.log(DF,2)
                TFIDF = TF*IDF
                list = os.path.splitext(os.path.basename(
                    filename))
                #getting the URI from the links files based on
                    the line number and document number
                    linecount = int(list[0])
                    r = linecache.getline('1000ulinks.txt',
                        linecount)
                    URI = r.strip()
                    tuple = tuple + [(TFIDF, TF, IDF, URI),]
                    fp.close()
#sorting the tuple based on TFIDF value in descending order
tuple = sorted(tuple, key=lambda tuples:tuples[0], reverse=True
    )

#writing the output in csv file
with open('rank.csv', 'a+', newline='') as csvfile:
          spamwriter = csv.writer(csvfile, delimiter=',',
              quotechar='|')
          spamwriter.writerow(['TF-IDF', 'TF', 'IDF', 'URI'])
          for a,b,c,d in tuple:


                    spamwriter.writerow(['%.3f' % a,'%.3f' % b,'%.3f
                        ' % c, '%s' % d])


          csvfile.close()
```

Listing 4: Python program to compute tfidf

I visited the `worldwidewebsize.com` to the number of pages indexed by Google which was 47B, while to get the number of documents in google with the query, I got the count as 1.47B sown in Figure 1. Then log to the base 2 was calculated using math.log [10]. The values were then sorted as per TF-IDF and stored in **rank.csv** file. The values are displayed in Table 1

TFIDF	TF	IDF	URI
0.083	0.017	4.999	https://www.azcentral.com/story/sports/high-school/2018/02/08/business-group-trying-keep-valley-junior-college-football-alive-beyond-2018/319131002/?hootPostID=fb32e68c95bcdc943b909439c5c31ba3
0.074	0.015	4.999	http://www.fox26houston.com/news/california-lawmakers-introduce-safe-youth-football-act-by-setting-minimum-age-for-tackle-football?utm_source=dlvr.it&utm_medium=twitter
0.071	0.014	4.999	http://abc7.com/sports/california-bill-would-ban-tackle-football-before-high-school/3058813//
0.065	0.013	4.999	http://www.chicagotribune.com/suburbs/daily-southtown/sports/ct-sta-pat-disabato-column-st-0211-20180209-story.html
0.035	0.007	4.999	http://www.latimes.com/politics/essential/la-pol-ca-essential-politics-updates-california-would-bar-organized-tackle-1518130990-htmlstory.html
0.023	0.005	4.999	http://www.telegram.com/news/20180209/five-st-johns-football-players-among-crop-of-area-athletes-finalizing-college-plans?platform=hootsuite
0.022	0.004	4.999	http://lompocrecord.com/sports/high-school/football/another-taua-to-head-to-reno---this-time/article_a7473c8e-13a8-5f79-82ac-8747900922df.html
0.021	0.039	4.999	http://www.dailymail.co.uk/sport/football/article-5374081/Man-United-star-Mata-fears-football-losing-soul.html?ITO=1490&ns_mchannel=rss&ns_campaign=1490
0.009	0.002	4.999	http://www.bbc.co.uk/sport/football/43010230
0.006	0.001	4.999	http://www.football-chairman.com

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



About 1,470,000,000 results (0.89 seconds)

Top stories

College Football's Top 15 Impact True Freshmen Quarterbacks for 2018
Athlon Sports
8 hours ago

Arizona football: Wildcats projected to be 33rd-best team in 2018
Arizona Desert Swarm
7 hours ago

ND repeats football win over LSU with football player's help
USA Today
1 hour ago

More for football

football on Twitter
https://twitter.com/search/football

Nick Harris (@sportingintel)
Wigan's last three FA Cup results v Manchester City, the most expensively assembled football team in history: WWW

Danny Baker (@prodnose)
Seems @bbc5live concerned that Wigan fans are taunting Man City fans at final whistle. Of course they are. That's what football is., Anyone see

Dan Roan (@danroan)
"Today we looked evil in the face, & smiled. Because we have won." Coming up at @BBCNews at 10, the victims who became victors, as

More images

American football
Sport

American football, referred to as football in the United States and Canada, and also known as gridiron football or simply gridiron, is a team sport played by two teams of eleven players on a rectangular field with goalposts at each end. Wikipedia

Equipment: Football (leather prolate spheroid); Football helmet; Pads (shoulder and knee)

Team members: 11 (both teams may freely substitute players between downs)

Country or region: Worldwide (most prominent in North America, Europe, and Japan)

Nicknames: Football, gridiron

First played: November 6, 1869; New Brunswick, New Jersey, United States; (Princeton vs. Rutgers)

Father: Walter Camp

Figure 1: Results from Google query of *football*

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

To calculate the Page Rank for each of the 10 URIs from the previous question I used the `http://www.checkpagerank.net/` uri as mentioned in the question. I have stored the values in **ptdf.csv**. File. Also the values are displayed in the Table 2. First I thought that URIs with high TFIDF values will have higher Page Rank, but this is not the case. From the Table 2 we can see that the TFIDF values varies with Page Ranks. Few higher paged rank URIs have lower TFIDF values while a couple of high TFIDF values have lower Page ranks. I have sorted the URIS considering first the Page Rank and then the respective higher TFIDF values since few URIs have the same Page Rank.

Page Rank	**TFIDF**	**URI**
0.9	0.0351	`http://www.latimes.com/politics/essential/la-pol-ca-essential-politics-updates-california-would-bar-organized-tackle-1518130990-htmlstory.html`
0.9	0.021	`http://www.dailymail.co.uk/sport/football/article-5374081/Man-United-star-Mata-fears-football-losing-soul.html?ITO=1490&ns_mchannel=rss&ns_campaign=1490`
0.9	0.009	`http://www.bbc.co.uk/sport/football/43010230`
0.8	0.083	`https://www.azcentral.com/story/sports/high-school/2018/02/08/business-group-trying-keep-valley-junior-college-football-alive-beyond-2018/319131002/?hootPostID=fb32e68c95bcdc943b909439c5c31ba3`
0.8	0.065	`http://www.chicagotribune.com/suburbs/daily-southtown/sports/ct-sta-pat-disabato-column-st-0211-20180209-story.html`
0.7	0.071	`http://abc7.com/sports/california-bill-would-ban-tackle-football-before-high-school/3058813/`
0.7	0.023	`http://www.telegram.com/news/20180209/five-st-johns-football-players-among-crop-of-area-athletes-finalizing-college-plans?platform=hootsuite`
0.6	0.074	`http://www.fox26houston.com/news/california-lawmakers-introduce-safe-youth-football-act-by-setting-minimum-age-for-tackle-football?utm_source=dlvr.it&utm_medium=twitter`
0.6	0.022	`http://lompocrecord.com/sports/high-school/football/another-taua-to-head-to-reno---this-time/article_a7473c8e-13a8-5f79-82ac-8747900922df.html`
0.5	0.006	`http://www.football-chairman.com`

Table 2: Page Rank and TFIDF Comparison of 10 URIs

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

For the above problem I first went through `http://en.wikipedia.org/wiki/Correlation_and_dependence` and got to know that it is basically calculating correlaton between pairs of values. The Tau value can have a coefficient between -1 to 1, which means that if the value is 1 or greater it has positive correlation between TFIDF and Page Rank values, meaning they are similar. If the value is zero or close, there is no correlation between them, its a tie. And if the value is closer to -1 or less, there is negative correlation, meaning the Page Rank is the opposite of the TFIDF value. Then I went through these references [3] and [4] and got to know that there is a Kendall package for R language, which gives a summary of values as output. The program reads the TFIDF and Page Rank lists from **ptdf.csv** file as shown in Listing 5.

```
#loading library Kendall
library(Kendall)
setwd(getwd())
#reading contents from csv file
csv <- read.table('ptdf.csv',header = TRUE,sep = ",")
#producing summary of results
summary(Kendall(csv$PageRank,csv$TFIDF))
```

Listing 5: R script to produce summary of Kendall's results

The summary produced:

```
Score =  0 , Var(Score) = 117.4667
denominator =  41.42463
tau = 0, 2-sided pvalue =1
```

From the above summary we can see that the tau value is zero meaning there is no correlation between the two pairs of values.

5

Question

```
====================================================
======Question 5 is for 3 points extra credit=======
====================================================

5.  Compute a ranking for the 10 URIs from Q2 using Alexa information
(see week 4 slides).  Compute the correlation (as per Q4) for all
pairs of combinations for TFIDF, PR, and Alexa.
```

Answer

For the above problem I calculated the Alexa rank from `https://www.alexa.com/siteinfo` for each of the 10 URIs which we got in question 2. The values are stored in **alexa_rank.csv** file as shown in Table 3.

I calculated the correlation for each of the TFIDF, PR and Alexa values as per Kendall Tau_b score from question 4. The program is written in R as shown in Listing 6. The values are rad from **alex.r csv** file It produced some interesting results.

```
##loading library Kendall
library(Kendall)
setwd(getwd())
#reading contents from csv
csv <- read.table('alexa.csv',header = TRUE, sep = ",")
#summary for PR-TFIDF
summary(Kendall(csv$PageRank,csv$TFIDF))
#summary for TFIDF-Alexa
summary(Kendall(csv$TFIDF,csv$Alexa))
#summary for PR-Alexa
summary(Kendall(csv$PageRank,csv$Alexa))
```

Listing 6: R program to produe summary of Kendall results

Summary produced for TFIDF and Alexa lists:

```
Score =  2 , Var(Score) = 124
denominator =  44.49719
tau = 0.0449, 2-sided pvalue =0.92844
```

From the above results we can see that the tau value is positive meaning there is some correlation between them, although I thought it would be zero.

Summary produced for Page Rank and Alexa lists:

```
Score =  -39 , Var(Score) = 118.3333
denominator =  41.89272
tau = -0.931, 2-sided pvalue =0.00047717
```

From seeing the above results, I was amazed to see that the Tau_b value is negative and closer to -1 meaning there is negative correlation between the values, although I thought it would be positive. But then I realized that Alexa scores the URIs opposite to that of the Page Rank, which means that there is correlation and similarity between them.

The correlation between TFIDF and Page Rank has already been tested in question 4.

Alexa Rank	**URI**
101	`http://www.bbc.co.uk/sport/football/43010230`
174	`http://www.dailymail.co.uk/sport/football/article-5374081/Man-United-star-Mata-fears-football-losing-soul.html?ITO=1490&ns_mchannel=rss&ns_campaign=1490`
830	`http://www.latimes.com/politics/essential/la-pol-ca-essential-politics-updates-california-would-bar-organized-tackle-1518130990-htmlstory.htm`
2172	`https://www.azcentral.com/story/sports/high-school/2018/02/08/business-group-trying-keep-valley-junior-college-football-alive-beyond-2018/319131002/?hootPostID=fb32e68c95bcdc943b909439c5c31ba3`
7094	`http://www.chicagotribune.com/suburbs/daily-southtown/sports/ct-sta-pat-disabato-column-st-0211-20180209-story.html`
8716	`http://abc7.com/sports/california-bill-would-ban-tackle-football-before-high-school/3058813/`
49281	`http://www.telegram.com/news/20180209/five-st-johns-football-players-among-crop-of-area-athletes-finalizing-college-plans?platform=hootsuite`
125065	`http://www.fox26houston.com/news/california-lawmakers-introduce-safe-youth-football-act-by-setting-minimum-age-for-tackle-football?utm_source=dlvr.it&utm_medium=twitter`
399889	`http://lompocrecord.com/sports/high-school/football/another-taua-to-head-to-reno---this-time/article_a7473c8e-13a8-5f79-82ac-8747900922df.html`
6633445	`http://www.football-chairman.com`

Table 3: Alexa Rank for 10 URIs

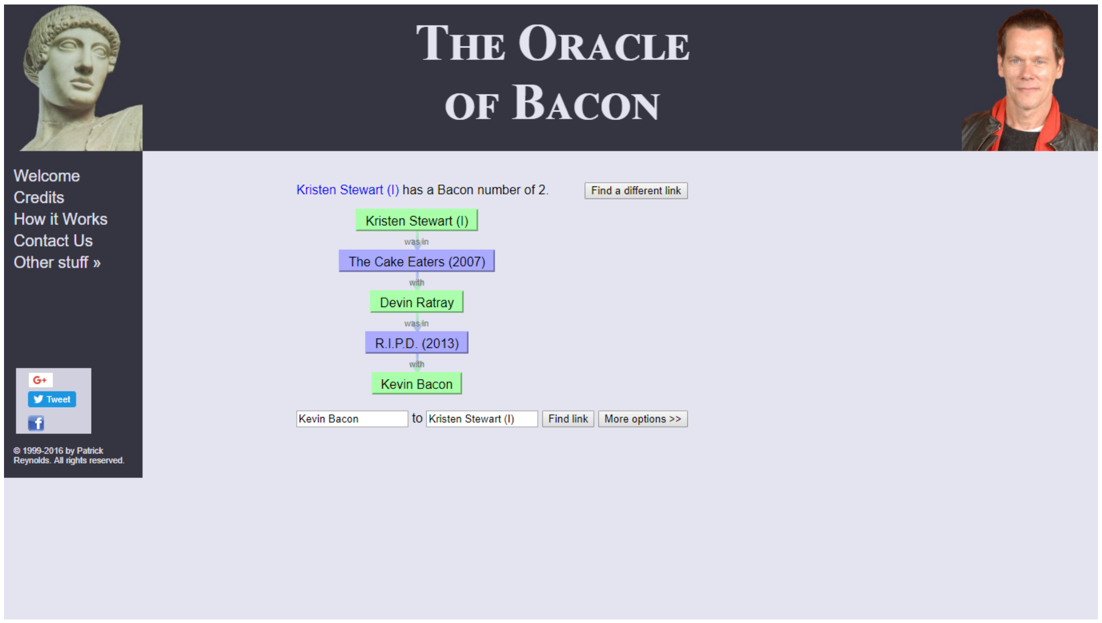
6

Question

```
====================================================
======Question 6 is for 2 points extra credit=======
====================================================

6.  Give an in-depth analysis, complete with examples,
graphs, and all other pertinent argumentation for
Kristen Stewart's (of "Twilight" fame) Erdos-Bacon number.
```

Answer

The Erdos-Bacon number is nothing but the sum of Erdos number and Bacon number. To get the Bacon number for Kristen Stewart I visited the `https://oracleofbacon.org/movielinks.php` uri which gave me a Bacon number of 2 for Kristen as shown in Figure 2.



Kristen Stewart (I) has a Bacon number of 2.

Kristen Stewart (I)
was in
The Cake Eaters (2007)
with
Devin Ratray
was in
R.I.P.D. (2013)
with
Kevin Bacon

Kevin Bacon to Kristen Stewart (I)

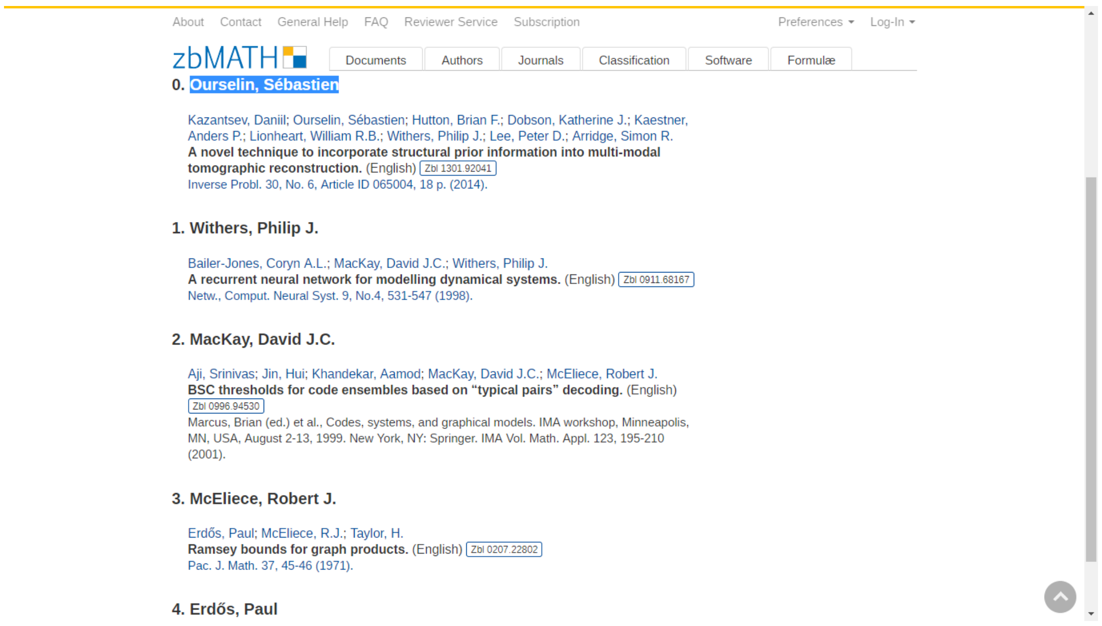
© 1999-2016 by Patrick Reynolds. All rights reserved.

Figure 2: Bacon number for Kristen Stewart

To find out the Erdos number for Kristen Stewart I searched through google to know the name of the paper she co-authored. I got to know that she published a paper on machine learning: `https://arxiv.org/pdf/1701.04928v1.pdf` along with Bhautik Joshi and David Shapiro. Then on further googling, I encountered a link`https://news.ycombinator.com/item?id=13446478` where I came to know that Bhautik Joshi had co-authored in *A quantitative assessment of approaches to mesh generation for surgical simulation* along with Sebastein Ourselin who has a Erdos number of 4 calculated from `https://zbmath.org/collaboration-distance/?a=ourselin.sebastien&b=erdos.paul` as shown in Figure3.

The following graph was produced using `Webgraphviz.com`:

From the above graph we get the Erdos number for Kristen Stewart as 6. Thus the Erdos-Bacon number for Kristen Stewart is $6 + 2 = 8$.



About Contact General Help FAQ Reviewer Service Subscription Preferences Log-In

zbMATH Documents Authors Journals Classification Software Formulæ

0. Ourselin, Sébastien

Kazantsev, Daniil; Ourselin, Sébastien; Hutton, Brian F.; Dobson, Katherine J.; Kaestner, Anders P.; Lionheart, William R.B.; Withers, Philip J.; Lee, Peter D.; Arridge, Simon R.
A novel technique to incorporate structural prior information into multi-modal tomographic reconstruction. (English) Zbl 1301.92041
Inverse Probl. 30, No. 6, Article ID 065004, 18 p. (2014).

1. Withers, Philip J.

Bailer-Jones, Coryn A.L.; MacKay, David J.C.; Withers, Philip J.
A recurrent neural network for modelling dynamical systems. (English) Zbl 0911.68167
Netw., Comput. Neural Syst. 9, No.4, 531-547 (1998).

2. MacKay, David J.C.

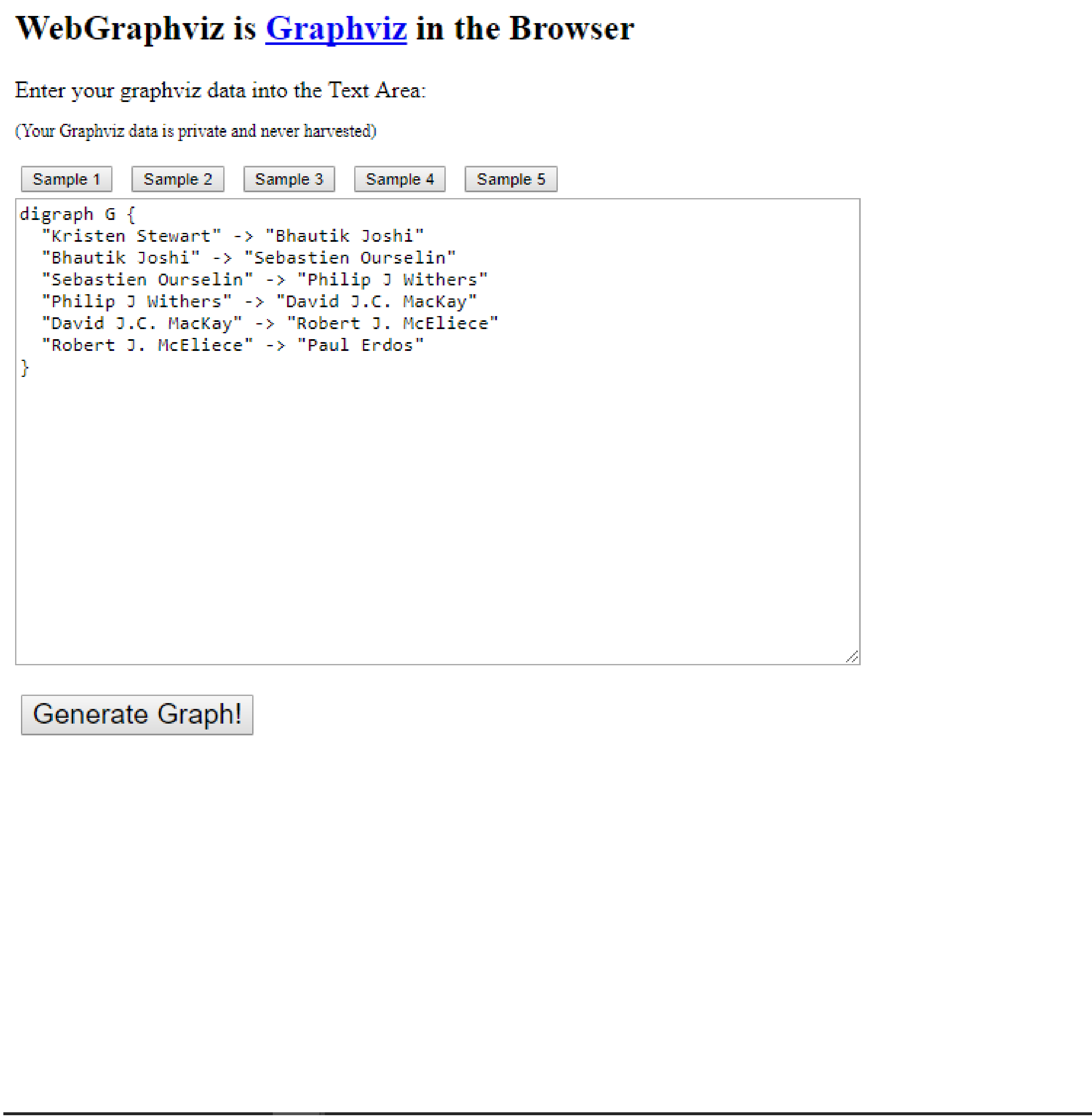
Aji, Srinivas; Jin, Hui; Khandekar, Aamod; MacKay, David J.C.; McEliece, Robert J.
BSC thresholds for code ensembles based on "typical pairs" decoding. (English) Zbl 0996.94530
Marcus, Brian (ed.) et al., Codes, systems, and graphical models. IMA workshop, Minneapolis, MN, USA, August 2-13, 1999. New York, NY: Springer. IMA Vol. Math. Appl. 123, 195-210 (2001).

3. McEliece, Robert J.

Erdős, Paul; McEliece, R.J.; Taylor, H.
Ramsey bounds for graph products. (English) Zbl 0207.22802
Pac. J. Math. 37, 45-46 (1971).

4. Erdős, Paul

Figure 3: Erdos number for Sebastein Ourselin



WebGraphviz is Graphviz in the Browser

Enter your graphviz data into the Text Area:

(Your Graphviz data is private and never harvested)

```
digraph G {
  "Kristen Stewart" -> "Bhautik Joshi"
  "Bhautik Joshi" -> "Sebastien Ourselin"
  "Sebastien Ourselin" -> "Philip J Withers"
  "Philip J Withers" -> "David J.C. MacKay"
  "David J.C. MacKay" -> "Robert J. McEliece"
  "Robert J. McEliece" -> "Paul Erdos"
}
```

Figure 4: Step1: Erdos graph for Kristen Stewart

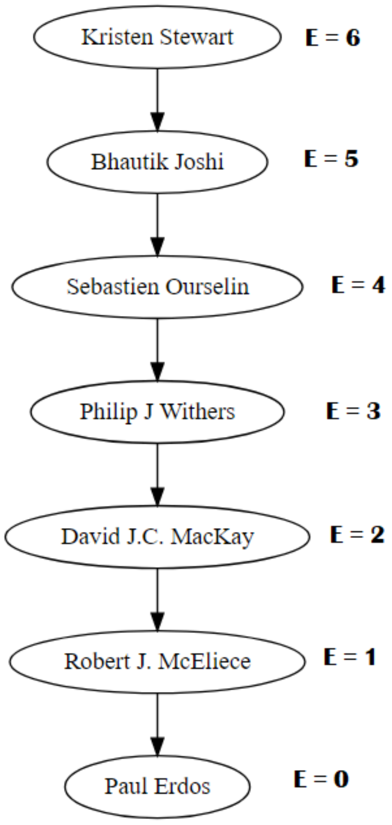


Figure 5: Step2: Erdos graph for Kristen Stewart

7

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

For the above problem which I found very interesting, wrote a program in Python 3.5 as shown in Listing 7. First I tested on few of the processed documents from question 1 and created wordlist for each of the document using *obj*.split() [9]. I observed that few of the documents had html content which was appending non-linguistic words to the list. I realized that I had to use a different boilerplate removal library, so I went through the [5] and figured out JusText [6] is perfect for this. The following dependencies were used:

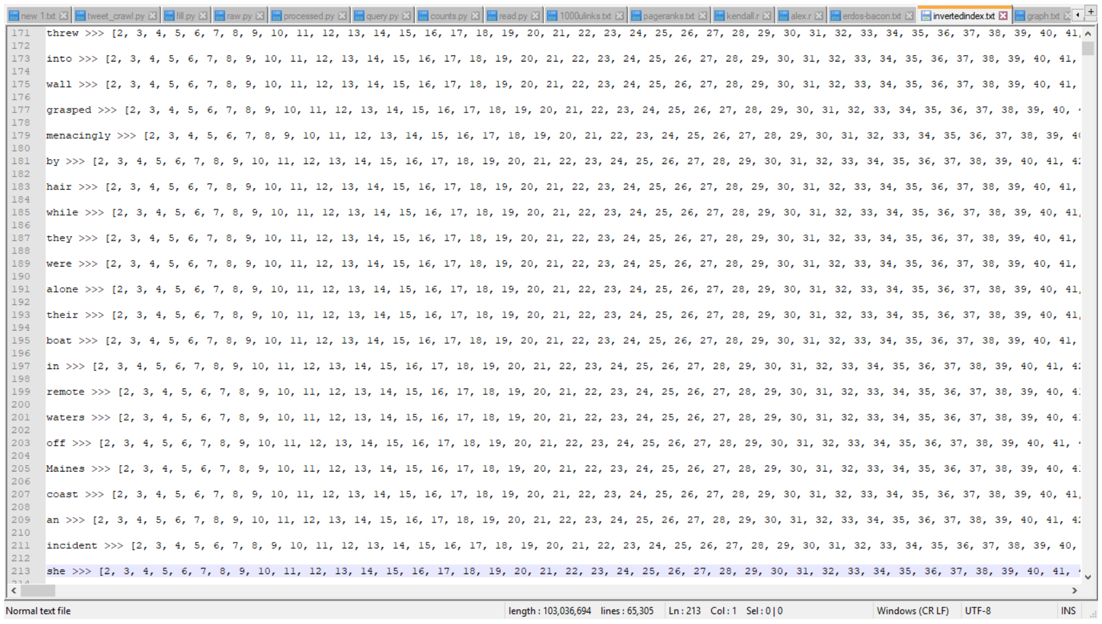
- import justext
- import re

```
import justext
import re

para = ''
d = {}
inv = []
count = 1

while (count < 1001):
        with open('raw_html/%s.html' % count, 'r', encoding='utf-8') as fp:
                #reading the files for raw html content
                wordstring = fp.read()
                #extracting linguistic sentences from the content by excluding the boilerplate content
                paragraphs = justext.justext(wordstring, justext.get_stoplist("English"))
                for paragraph in paragraphs:

                        if not paragraph.is_boilerplate:
                                para = para + ' ' + (paragraph.text)
                #creating a list of words
                wordlist = para.split()

                for w in wordlist:
                #excluding non-alphanumeric characters
                        w = re.sub(r'\W+', '', w)

                #checkig for duplicates
                        if w not in inv:
                        # check if string not empty
```

```
                    if w:
                        inv.append(w)

            for w1 in inv:
                #appending values to keys in dictionary
                try:
                    d[w1].append(count)
                except KeyError:
                    d[w1] = [count]
                    pass
        count = count + 1
        #writing the result in invertedindex.txt file in ascii format
with open('invertedindex.txt', 'w', encoding='ascii') as pp:
            for k, v in d.items():
                pp.write(str(k) + ' >>> '+ str(v) + '\n\n')
```

Listing 7: Python script to create an inverted index

The program first reads the html documents one by one from the raw_html folder which we got from question 1 and reads the contents into justext.justext() [6] function. It then extracts words using paragraph.text [6]. Then it filters all non-alphanumeric characters from the list using re.sub() [8]. Then appending the words one by one as key to a dictionary with values as the document number it writes the output in **invertedindex.txt** file. A sample output is shown below. What we can see that it starts from document 2 because document 1 had no content.



```
threw >>> [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41,

into >>> [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41,

wall >>> [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41,

grasped >>> [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40,

menacingly >>> [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39,

by >>> [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41,

hair >>> [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41,

while >>> [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41,

they >>> [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41,

were >>> [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41,

alone >>> [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41,

their >>> [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41,

boat >>> [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41,

in >>> [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41,

remote >>> [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40,

waters >>> [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40,

off >>> [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41,

Maines >>> [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40,

coast >>> [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41,

an >>> [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41,

incident >>> [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40,

she >>> [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41,
```

Figure 6: Inverted Index Output

References

[1] "Requests: HTTP for Humans." equests: HTTP for Humans - Requests 2.18.4 documentation, n.d. Web. February 20, 2018. http://docs.python-requests.org/en/master/.

[2] "Stack Overflow."Switching to Python 3 causing UnicodeDecodeError. N.p.,n.d. Web. February 20, 2018. https://stackoverflow.com/questions/23917729/switching-to-python-3-causing-unicodedecodeerror

[3] "Kendall rank correlation coefficient." Wikipedia - Kendall Rank Correlation coefficient. Wikipedia, n.d. Web. February 20, 2018. https://en.wikipedia.org/wiki/Kendall_rank_correlation_coefficient#Tau-b

[4] "CRAN - Package Kendall." Kendall Rank Correlation. n.d. Web. February 20, 2018. https://cran.r-project.org/web/packages/Kendall/Kendall.pdf

[5] Alexander Nwala. "Web Science and Digital Libraries Research ." 2017-03-20: A survey of 5 boilerplate removal methods. N.p., Web. February 20, 2018. http://ws-dl.blogspot.com/2017/03/2017-03-20-survey-of-5-boilerplate.html.

[6] Miso-belica. "miso-belica/jusText .", GitHub, March 05, 2017, n.d., Web. February 20, 2018. https://github.com/miso-belica/jusText.

[7] Misja. "misja/python-boilerpipe." Github Repository. N.p., October 03, 2017. Web. February 20, 2018.https://github.com/misja/python-boilerpipe.

[8] "6.2. re - Regular expression operations." 6.2. re - Regular expression operations - Python 3.6.4 documentation. n.d., Web. February 20, 2018. https://docs.python.org/3/library/re.html.

[9] William Turkel-Adam Crymble. "Counting Word Frequencies with Python" Programming Historian. N.p., n.d. Web. February 20, 2018.https://programminghistorian.org/lessons/counting-frequencies.

[10] "9.2. math - Mathematical functions." 9.2. math - Mathematical functions - Python 2.7.14 documentation. N.p., n.d. Web. February 20, 2018. https://docs.python.org/2/library/math.html.