INTRO. TO WEB SCIENCE: CS 532: A3

Due on Thursday, February 23, 2017

Dr. Nelson

Udochukwu Nweke

Contents

Problem 1

1. Download the 1000 URIs from assignment # 2. “curl”, “wget”, or “lynx” are all good candidate programs to use. We want just the raw raw HTML, not images, stylesheets, etc. Listing shows a Perl script.

Listing 1: Extract HTML

```python
import commands
import os, sys
from bs4 import BeautifulSoup

import re
import sys

reload(sys)
sys.setdefaultencoding('utf8')

#from nltk: https://github.com/nltk/nltk/commit/39a303e5ddc4cdb1a0b00a3be426239b1c24c8bb
def clean_html(html):

    # First we remove inline JavaScript/CSS:
    cleaned = re.sub(r"(?is)<(script|style).*?>.*?(</\1>)", "", html.strip())
    # Then we remove html comments. This has to be done before removing regular
    # tags since comments can contain '>' characters.
    cleaned = re.sub(r"(?s)<!--(.*?)-->[\n]?", "", cleaned)
    # Next we can remove the remaining tags:
    cleaned = re.sub(r"(?s)<.*?>", " ", cleaned)
    # Finally, we deal with whitespace
    cleaned = re.sub(r"&nbsp;", " ", cleaned)
    cleaned = re.sub(r"  ", " ", cleaned)
    cleaned = re.sub(r"  ", " ", cleaned)

    #my addition to remove blank lines
    cleaned = re.sub("\n\s*\n*", "\n", cleaned)

    return cleaned.strip()

def saveTextToFile(filename, text):
    try:
        outfile = open(filename, 'w')
        outfile.write(text)
        outfile.close()
    except:
        errorMessage()

def derefURL(url):

    try:
        co = 'curl -L --silent -m 20 "' + url + '"'
        data = commands.getoutput(co)
        return data
    except:
        errorMessage()
        return ''
```

Problem 1 continued on next page. . .

```python
def errorMessage():
    exc_type, exc_obj, exc_tb = sys.exc_info()
    fname = os.path.split(exc_tb.tb_frame.f_code.co_filename[1])
    print fname, str(exc_tb.tb_lineno), str(sys.exc_info())

def getUniqueURLs():

    infile = open('./1000UniqueURLs.txt', 'r')
    lines = infile.readlines()
    infile.close()

    return lines

def saveRawHTML(lines):

    for i in range(0, len(lines)):

        textOutfilename = './Text/' + str(i) + '.txt'
        #if( os.path.exists(textOutfilename) == True ):
        #    continue

        url = lines[i].strip()
        html = derefURL(url)

        if( len(html) != 0 ):

            try:
                #soup = BeautifulSoup(html, 'html.parser')
                #text = soup.get_text()
                text = clean_html(html)
                if( len(text) != 0 ):
                    saveTextToFile(textOutfilename, text)
                    print '\tsaved text', len(text)

            except:
                errorMessage()

            try:
                saveTextToFile('./RawHTML/' + str(i) + '.html', html)
                print '\tsaved html'
            except:
                errorMessage()

        print i

if __name__ == '__main__':

    lines = getUniqueURLs()
    saveRawHTML(lines)
```

Solution 1:

1. In order to get the raw HTML files for the 1000 files I downloaded from Assignment #2,I used a function called *derefURL()* to extract the HTML contents using curl -L option to follow redirects. The raw HTML

for the 1000 URIs is saved in RawHTML folder. This can be seen in listing 1 line 39
2. After downloading the HTML files, I used NLTK clean HTML function *(clean-html())* remove the HTML markup. This is seen in Listing 1, line 12.

Table 1: 10 Hits for the term "Russia", ranked by TFIDF. Complete URL list in file"10URLsWithRussia.txt"

TFIDF	TF	IDF	URI
0.0151	0.0044	3.4226	`http://insider.foxnews.com/...`
0.0041	0.0012	3.4226	`http://variety.com/...`
0.0038	0.0011	3.4266	`http://www.express.co.uk/...`
0.0034	0.0001	3.4266	`http://100percentfedup.com/...`
0.0027	0.0008	3.4266	`http://www.latimes.com/...`
0.0027	0.0008	3.4266	`http://www.politico.com/...`
0.0024	0.0007	3.4266	`https://www.buzzfeed.com/...`
0.0017	0.0005	3.4266	`https://www.washingtonpost.com/...`
0.0017	0.0005	3.4266	`https://www.theatlantic.com/...`
0.0014	0.0004	3.4266	`http://www.ocregister.com/...`

Problem 2

Choose a query term (e.g., "Shadow") that is not a stop word (see week 5 slides) and not the 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 10 documents, 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 corressponding URIs. The URIs will be ranked in decreasing order by TFIDF values.

Listing 2: TFIDF Code Snippet

```
from P1 import getUniqueURLs
from P1 import errorMessage
import os
import commands
import math

def searchForTermInFiles(lines, key):

    count = 0
    for i in range(0, len(lines)):

        print i
        url = lines[i].strip()
        filename = './Text/' + str(i) + '.txt'

        if( os.path.exists(filename) == False ):
            continue

        co = 'cat ' + filename + ' | grep -i -w ' + key
```

```
            output = commands.getoutput(co)
            #output = output.strip()

            if( len(output) != 0 ):
                print '\tFound'
                print '\t', url
                count += 1

            #if( count == 10 ):
            #    break

        print 'found count:', count
        print 'key:', key

def calculateTFIDFForIndex(lines, docsWithTerm, key, idfvalue):

        print 'TF-IDF for key:', key
        for docIndex in docsWithTerm:

            url = lines[docIndex].strip()
            filename = './Text/' + str(docIndex) + '.txt'

            co = 'cat ' + filename + ' | grep -icw ' + key

            try:
                output = commands.getoutput(co)
                wordCount = int(output)

                co = 'wc -w ' + filename
                totalWords = commands.getoutput(co)
                totalWords = int(totalWords.split(' ')[0].strip())

                tf = 0
                if( totalWords > 0 ):
                    tf = round(wordCount/float(totalWords), 4)

                print url
                #print filename
                print 'wordCount:', wordCount
                print 'totalWords:', totalWords
                print 'tf:', tf
                print 'idf:', IDF
                print 'tfidf:', round(tf * IDF, 4)


            except:
                errorMessage()

            print ''



lines = getUniqueURLs()
```

```
key = 'Russia'
#searchForTermInFiles(lines, key)

#IDF = 1000 docs / 93 documents with term Russia, IDF = log2(1000/93): 3.4266
IDF = 3.4266
docsWithTerm = [5, 6, 7, 11, 13, 19, 20, 25, 26, 68]
calculateTFIDFForIndex(lines, docsWithTerm, key, IDF)
```

Solution 2:
1. I chose a query term "Russia" and used grep command to search the number of times term "Russia" appeared in 10 different documents. grep-i gave me the flexibility of searching for both upper and lower case terms. grep -w is to match the whole word so as not to match sub strings. For example, don't match "Russian", but match "Russia". The function *searchForTermInFiles()* in line 7 of listing 2. is used to search for term"Russia" in the files.

2. After searching for term "Russia" and calculating the number of times it appeared in the document, I created a function to calculate the TFIDF. The function *calculateTFIDFForIndex()* is the fuction that calculates the TFIDF for all 10 documents with term "Russia".

TFIDF is the product of TF and IDF
TF is the Term Frequency and it counts the number of times term "Russia" occurs in the document. I got 93 documents with term "Russia" out of 1000 documents. IDF is the inverse Document Frequency and is calculated as follows:

$IDF = \frac{Numberofdocuments}{NumberofdocumentswithtermRussia} = \frac{1000}{93} = 3.4266$
Table 1 includes the TFIDF values for all 10 links.

Problem 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.

Solution 3:
1. Table 2 shows the Page Rank and normalized page rank for the 10 urls.

2. I used the `http://pr.eyedomain.com/`
to get the page ranking for the 10 urls.

3. Comparing the rankings produced in Table 1 and 2 shows that the higher the TFIDF, the higher the page ranks. urls with the same TFIDF have the same page rank.

Table 2: 10 Hits for the term "Russia", ranked by PageRank

PageRank Normalised	PageRank	URI
1.0	8	`http://insider.foxnews.com/...`
1.0	8	`http://www.latimes.com/...`
1.0	8	`https://www.buzzfeed.com/...`
1.0	8	`https://www.washingtonpost.com/...`
1.0	8	`https://www.theatlantic.com/...`
0.9	7	`http://variety.com/...`
0.9	7	`http://www.express.co.uk/...`
0.9	7	`http://www.politico.com/...`
0.9	6	`http://www.ocregister.com/...`
0.0	0	`http://100percentfedup.com/...`

Problem 4

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.
Solution 4:
1. Table 3 includes the ranks from TFIDF and Page Rank for the list of 10 URLs.

2. I computed the Kendall Tau_b and Pearson correlation coefficient through an R script (Listing 3).
The Kendell Tau_b value for TFIDF and Page Rank listings is -0.0809, and the Pearson Value is -0.03808, showing no significant correlations.

Listing 3: R Code Snippet For Kendell Tau_b and P values

```
TFIDF_rank <- c(1,2,3,4,5,5,7,8,8,10)
page_rank <- c(1,6,6,10,1,6,1,1,1,9)
alexa_rank <- c(3,8,7,10,4,5,2,1,6,9)

pearson_TFIDF_PR <- cor(TFIDF_rank, page_rank)
pearson_TFIDF_PR
kendallTauB_TFIDF_PR <- cor(TFIDF_rank, page_rank, method="kendall")
kendallTauB_TFIDF_PR


pearson_TFIDF_alexa <- cor(TFIDF_rank, alexa_rank)
pearson_TFIDF_alexa
kendallTauB_TFIDF_alexa <- cor(TFIDF_rank, alexa_rank, method="kendall")
kendallTauB_TFIDF_alexa


pearson_PR_alexa <- cor(page_rank, alexa_rank)
pearson_PR_alexa
kendallTauB_PR_alexa <- cor(page_rank, alexa_rank, method="kendall")
kendallTauB_PR_alexa
```

Table 3: 10 Hits for the term "Russia", ranked by PageRank

TFIDF_Rank	Page_Rank
1	1
2	6
3	6
4	10
5	1
5	6
7	1
8	1
8	1
10	9

Problem 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.
Solution 5:
1. In order to get the Alexa_Ranking (Table 4) for the 10 URLs I used Alexa's XML API `http://data.alexa.com/data?cli=10&url=` to manually get the Alexa_Rank for the 10 urls (Table **??**).

2. After getting Alexa ranks for the 10 URLs, I used the the values of TFIDF_Rank, Page_Rank and Alexa_Rank to compute the correlation and Kendell Tau_b values for all combinations through and R script (Listing 3.). Table 5 displays the Kendall's Tau_b and P values for all combinations of rankings.

Table 4: Alexa_Rank

TFIDF_Rank	Page_Rank	Alexa_Rank
1	1	s
2	6	8
3	6	7
4	10	10
5	1	4
5	6	5
7	1	2
8	1	1
8	1	6
10	9	9

Table 5: Correlation for all pairs of ranks

Combination	P-value	Kendall's Tau_b
TFIDF, PR	-0.03805454	-0.08087458
TFIDF, Alexa	-0.08203403	-0.1136657
PR, Alexa	0.8731077	0.7905694

Problem 6

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

Solution 6:
"Six Degrees of Separation" is the name of an academic play by John Guarre. This concept creates a place that would connect millions of people around the world. This concept explains how anyone in the planet can be connected to another person with an average of six jumps. Hence, the society we live in is a smallworld type of network characterised by short path lengths.

"Six degrees of Kavin Bacon" is a palour game for movie lovers. The Bacon number of a movie actor tries to connect a particular actor to Kavin Bacon. The number of jumps it takes to link a particular actor to a movie with Kavin Bacon directly or indirectly will give the actor's Bacon number. Directly means that the actor has appeared in the same movie with Kavin Bacon, while indirectly means the actor has appeared in a movie with some actor that has appeared in a movie with Kavin Bacon.

Just as "Six degrees of Kavin Bacon" is used to get an actors Bacon number, the Erdos number also gives the collaborative distance between mathematician Paul Erdos and some other person. The Erdos number can be measured by authorship of mathematical papers. Hence, Erdos number is the jumps it takes to link an author who co-authored with Paul Erdos directly or indirectly.

Erdos-Bacon number is simply the sum of Bacon number and Erdos number.

Kristen Stewarts Erdos-Bacon number will be the sum of the jumps it takes to link her to to a movie with Kevin Bacon directly or indirectly (Bacon number) and the jumps it takes to link her with a publication co-authored with Paul Erdon directly or indirectly (Erdos number). I used `https://oracleofbacon.org/` to calculate how Kristen Stewart is linked to kavin Bacon via movie appearances (Figure 1.) She has a Bacon number of 2.
In order to calculate Kristen Stewart's erdos number, I looked for a co-author that connected her to Paul Erdos indirectly. She published a paper on "Bringing Imression to Life with Neural Style Transfer in Come Swim" with Bautik J Josheni and David Shapiro. I searched for Bhautik's publications and found out that Bhautik co-authored most of his publications with Sebastein Ourselin and Sebastin Ourselin has an erdos number of 4 as shown in figure 2. This gives Kristen Stewart an Erdos number of 6.

The graph in figure 3 shows one of several links from Kristen Stewart to Paul Erdos based on paper publication.
However, there could be possible shorter links from Kristen Stewart, but this will require an exhaustive exploration of authorship graph. Figure 4 shows an imaginary graph with possible shorter links to Erdos.
Kristen Stewart's Erdos-Bacon number is 2+6 = 8.

Problem 6 continued on next page. . .

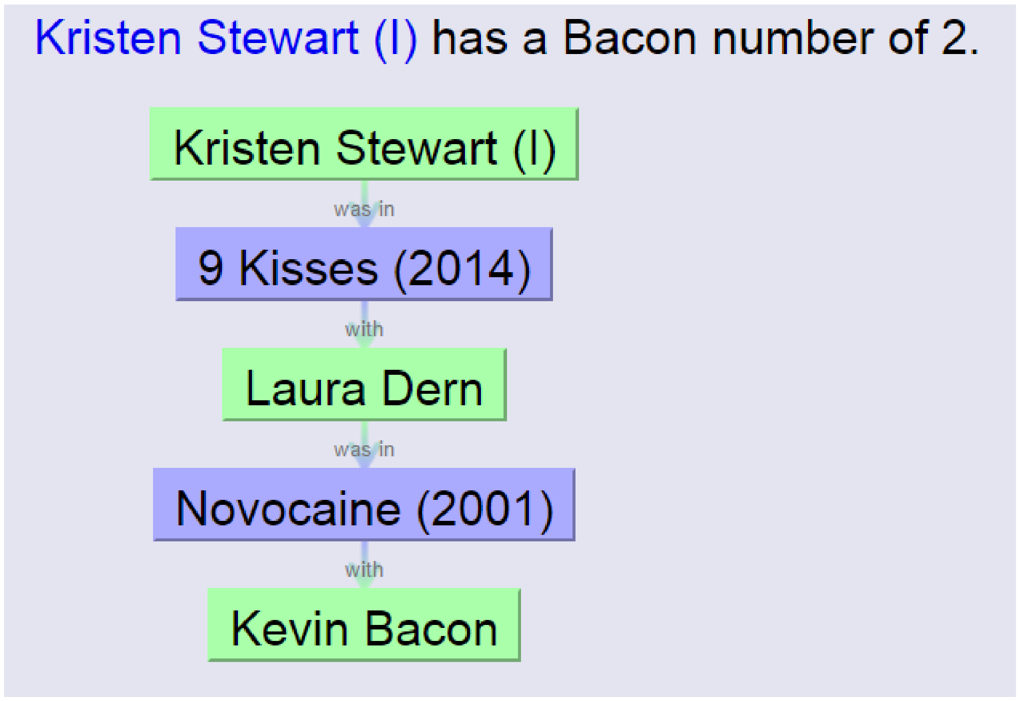
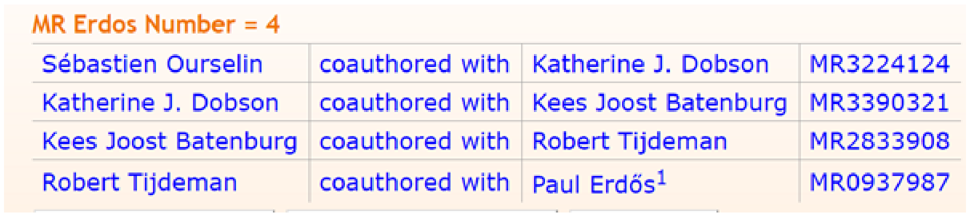


Figure 1: Bacon Number Result



MR Erdos Number = 4

Sébastien Ourselin	coauthored with	Katherine J. Dobson	MR3224124
Katherine J. Dobson	coauthored with	Kees Joost Batenburg	MR3390321
Kees Joost Batenburg	coauthored with	Robert Tijdeman	MR2833908
Robert Tijdeman	coauthored with	Paul Erdős[1]	MR0937987

Figure 2: Erdos Number Result

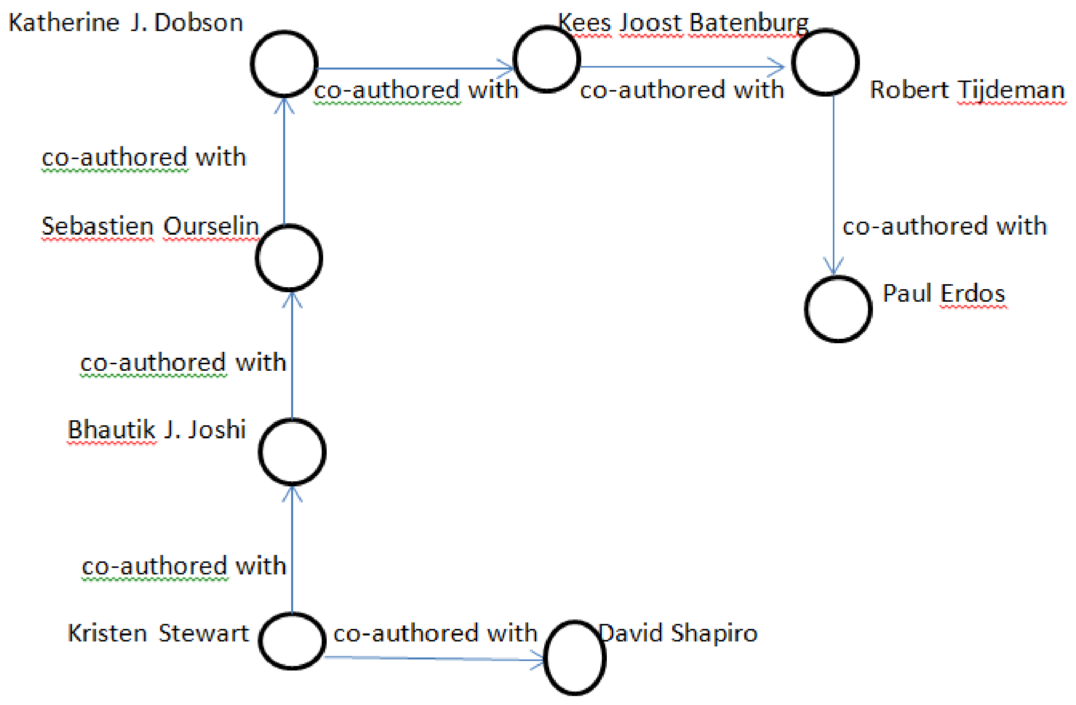


Figure 3: Kristen Stewart's Erdos Graph

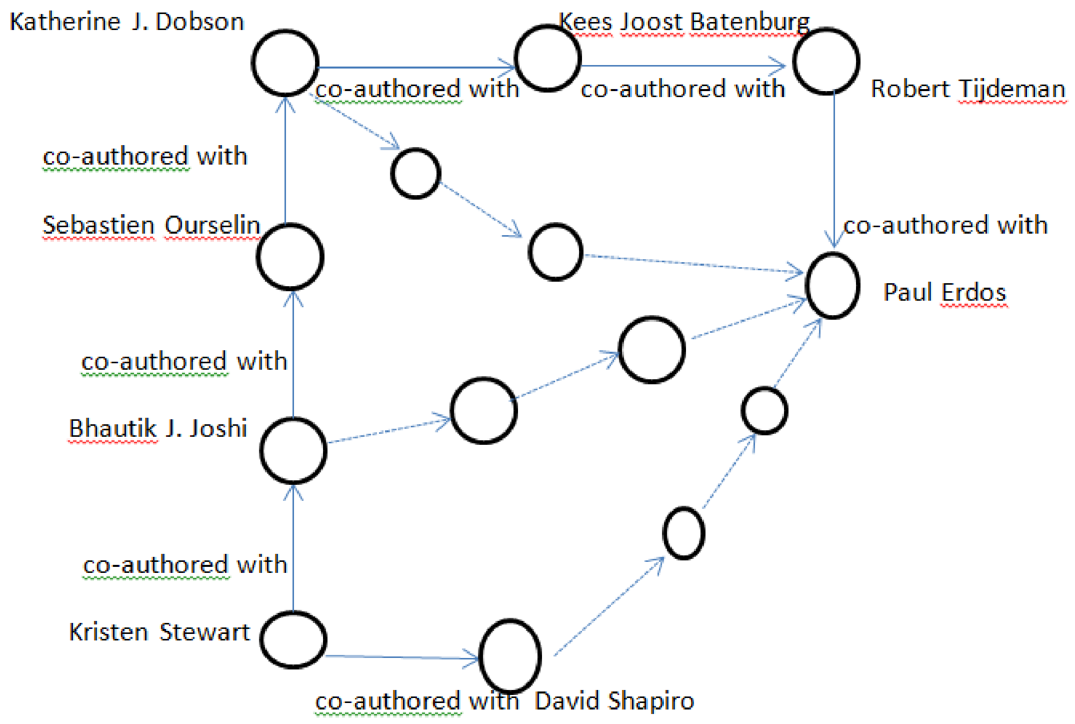


Figure 4: Kristen Stewart's Erdos Imaginary Graph

Problem 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.
Solution 7:

Listing 4: Inverted File Code

```
import os, sys
from subprocess import check_output


def errorMessage():
     exc_type, exc_obj, exc_tb = sys.exc_info()
     fname = os.path.split(exc_tb.tb_frame.f_code.co_filename[1])
     print(fname, str(exc_tb.tb_lineno), str(sys.exc_info()))

def getUniqueURLs():

     infile = open('./1000UniqueURLs.txt', 'r')
     lines = infile.readlines()
     infile.close()

     return lines
def getInvertedFile(lines, vocabulary, multipleVocabs):

     counter = 0
     for term, value in vocabulary.items():

          invertedLine = ''
          for i, docVocabDict in multipleVocabs.items():

               try:

                    print(i, 'of', len(multipleVocabs))
                    print('term:', term)
                    print('counter:', counter, 'of', len(vocabulary))
                    print('')

                    if( term in multipleVocabs[i] ):
                         invertedLine = invertedLine + ', ' + str(i)

               except:
                    errorMessage()

          if( len(invertedLine) != 0 ):
               if( invertedLine.strip()[0] == ',' ):
                    invertedLine = invertedLine[1:]

               invertedLine = term + ': ' + invertedLine
               try:
                    outfile = open('./invertedFile_good.txt', 'a')
                    outfile.write(invertedLine + '\n')
```

Problem 7 continued on next page. . .

```
                    outfile.close()
                except:
                    errorMessage()

        counter += 1

def getVocabulary(lines):

    vocabulary = {}
    multipleVocabs = {}
    for i in range(0, len(lines)):

        try:
            multipleVocabs[i] = {}
            print(i)
            url = lines[i].strip()
            filename = './Text/' + str(i) + '.txt'

            if( os.path.exists(filename) == False ):
                continue

            output = check_output(['cat', filename])
            output = output.decode('utf-8')
            output = output.lower().split(' ')

            for term in output:
                term = term.strip()

                if( len(term) != 0 ):
                    vocabulary[term] = True
                    multipleVocabs[i][term] = True
        except:
            errorMessage()


    return vocabulary, multipleVocabs


lines = getUniqueURLs()
vocabulary, multipleVocabs = getVocabulary(lines)

getInvertedFile(lines, vocabulary, multipleVocabs)
```

In order to generate an inverted file, I created a giant dictionary which consists of all terms in all the documents and I tokenised by space. I checked how many documents had each term in the vocabulary and wrote this statistic for each term.

References

[1] Calculating Kendalls Tau Rank Correlation Coefficient. http://stamash.org/calculating-kendalls-tau-rank-correlation-coefficient/. Accessed: 2014-10-01.

[2] R tutorial. http://www.r-tutor.com/gpu-computing/correlation/kendall-tau-b/. Accessed: 2014-10-01.

[3] Lada A Adamic. The small world web. In *International Conference on Theory and Practice of Digital Libraries*, pages 443–452. Springer, 1999.