INTRO. TO INFO RETRIEVAL: CS 734: A2

Due on Saturday, October 14, 2017

Dr. Nelson

Udochukwu Nweke

Contents

Problem 1

5.8 Write a program that can build a simple inverted index of a set of text documents. Each inverted list will contain the file names of the documents that contain that word. Suppose the file A contains the text “the quick brown fox”, and file B contains “the slow blue fox”. The output of your program would be:

```
% ./your-program A B
blue B
brown A
fox A B
quick A
slow B
the A B
```

Solution 1:

Listing 1: Inverted file code snippet

```python
#!/usr/bin/env python

import sys

def genInverted(fileDict):

    vocab = set()
    for filename, content in fileDict.items():

        f = content
        f = f.lower()
        f = f.split()
        f = set(f)

        fileDict[filename] = f
        vocab = vocab.union(f)

    invertedFileDict = {}
    for word in vocab:

        invertedFileDict[word] = []
        for filename, content in fileDict.items():

            if( word in fileDict[filename] ):
                invertedFileDict[word].append(filename)

    return invertedFileDict

def main(filenames):

    fileDict = {}

    try:
        for f in filenames:
            f = f.strip()

            infile = open(f, 'r')
            fileDict[f] = infile.read()
            infile.close()

    except Exception as e:
        print('File error:', e)

    invertedFileDict = genInverted(fileDict)

    print('Inverted index of files:', filenames, '\n')
    for vocab, files in invertedFileDict.items():
        print(vocab, files)
    print()

if __name__ == "__main__":

    if( len(sys.argv) > 1 ):
        main( sys.argv[1:] )
    else:
        print('Invalid use: no parameters given')
```

In order to build a simple inverted index, I took the following steps:

1. I read text from three or more different text files namely, A, B, and C.

2. I generated a vocabulary by taking the union of all the terms in all the documents

3. I stored the vocabulary into a dictionary with values as terms and keys as the files in which the terms occour in. I generated an inverted index from the dictionary. This is demonstrated in listing 1, line 2-17.

Problem 2

4.8. Find the 10 Wikipedia documents with the most inlinks. Show the collection of anchor text for those pages.

Solution 2:

Listing 2: Extract inlinks from wikismall

```python
import glob

from common import genericErrorInfo
from common import derefURL
from common import readTextFromFile
from common import dumpJsonToFile
from common import getDictFromFile

from bs4 import BeautifulSoup
```

Problem 2 continued on next page. . .

```
from urllib.parse import unquote

def getHTMLPaths():

    pathnames = []

    try:
        infile = open('wiki-small-html-files.txt', 'r')
        pathnames = infile.readlines()
        infile.close()
    except:
        genericErrorInfo()

    return pathnames

def getHTMLFilename(wiki):

    wiki = wiki.strip()
    wiki = wiki.split('.html')[0]
    wiki = wiki.split('/')[-1].strip().lower()

    return wiki

def getHTMLFilenames(pathnames):

    filenames = []
    for path in pathnames:
        path = getHTMLFilename(path)
        filenames.append(path)

    return filenames

def storeHTMLFiles_single_use():

    outfile = open('wiki-small-html-files.txt', 'w')

    counter = 0
    for filename in glob.iglob('en/**/*.html', recursive=True):
        outfile.write(filename + '\n')
        counter += 1

    outfile.close()


def getWikiOutlinks(sourcewiki, html, outlinksDict):

    if( len(html) == 0 ):
        return outlinksDict

    try:
        soup = BeautifulSoup(html, 'html.parser')
        anchorTags = soup.find_all('a')
```

Problem 2 continued on next page. . .

```
        for tag in anchorTags:

            if( tag.has_attr('href') == False ):
                continue

            if( tag['href'].find('wikipedia') == -1 ):
                continue

            link = tag['href']
            link = unquote(link)

            outlinksDict.setdefault(link, 0)
            outlinksDict[link] += 1

    except:
        genericErrorInfo()

    return outlinksDict


def getTopKPages(pathnames, filenames):

    if( len(pathnames) == 0 ):
        return []

    outlinksDict = {}

    for i in range(len(pathnames)):
        wiki = pathnames[i]
        wiki = wiki.strip()
        html = readTextFromFile(wiki)

        if( i % 100 == 0 ):
            print(i, 'of', len(pathnames), 'wiki file:', wiki)
            print('\tlen:', len(outlinksDict))

        sourcewiki = getHTMLFilename(wiki)
        getWikiOutlinks(sourcewiki, html, outlinksDict)

        #if( i == 3 ):
        #    break

    dumpJsonToFile('./outlinksDict.json', outlinksDict)

def getTopKFromDict(k):

    print('\n'*2)
    print('top', k)
    if( k<1 ):
        return

    outlinksDict = getDictFromFile('./outlinksDict.json')
    result = sorted(outlinksDict.items(), key=lambda x: x[1], reverse=True)
```

Problem 2 continued on next page. . .

```
    for i in range(len(result)):
        print(i+1, result[i])

        if( i == k-1 ):
            break


def main():
    #pathnames = getHTMLPaths()
    ##filenames = getHTMLFilenames(pathnames)
    #getTopKPages(pathnames, filenames)

    getTopKFromDict(30)

if __name__ == '__main__':
    main()
```

1. I read the HTML files that were downloaded from the wiki small corpus from `http://www.searchengines-book.com`

2. I parsed the HTML files from step 1 to *BeautifulSoup* to extract outlinks. This is demonstrated in lisitng 2, *getWikiOutlinks()*

3. I stored all the links extracted into a dictionary with key as link and value as the frequency of occurence of the link.

4. I sorted the dictionary by value in reverse order, as demonstrated in listing 2 line 115.

5. I picked the top 10 links that have the highest number of inlinks

Problem 3

4.1. Plot rank-frequency curves (using a log-log graph) for words and bigrams in the Wikipedia collection available through the book website (`http://www.searchengines-book.com`). Plot a curve for the combination of the two.What are the best values for the parameter c for each curve?

Solution 3:

Listing 3: n-gram Vocabulary Code

```
from P2 import getHTMLPaths
from common import readTextFromFile
from common import getTextFromHTML
from common import dumpJsonToFile
from common import getDictFromFile

from sklearn.feature_extraction.text import CountVectorizer

def genCSVFile():

    outfile = open('sorted-1-2-gram.org.csv', 'w')
    outfile.write('Rank,Term,Freq,C\n')

    gramsDict = getDictFromFile('1-2-gram.json')
    sortedDict = sorted(gramsDict.items(), key=lambda x: x[1], reverse=True)

    total = 770552
    rank = 1
    for tup in sortedDict:
        term, freq = tup
        c = (freq/total) * rank
        outfile.write( str(rank) + ', ' + term + ', ' + str(freq) + ', '
        + str(round(c, 5)) + '\n' )
        rank += 1

    outfile.close()


def getVocabFreqDict(filenames):

    vocabDict = {}

    for i in range( len(filenames) ):
        f = filenames[i].strip()
        html = readTextFromFile(f)
        text = getTextFromHTML(html)

        if( len(text) == 0 ):
            continue

        countVectorizer = CountVectorizer( min_df=1, stop_words='english',
        ngram_range=(1,2) )
        termFreqMat = countVectorizer.fit_transform([text])

        for term in list(countVectorizer.vocabulary_.keys()):
            vocabDict.setdefault(term, 0)
            vocabDict[term] += 1

        if( i % 100 == 0 ):
            print( i, 'of', len(filenames) )

    dumpJsonToFile( '1-2-gram.json', vocabDict )


#get ngram (1, 2) dictionary
#filenames = getHTMLPaths()
#getVocabFreqDict(filenames)

genCSVFile()
```

Listing 4: Extract link from wikismall

```
"respiratory syndrome": 1,
    "important genre": 1,
    "arranged rift": 1,
    "ended 1993": 2,
    "placing contracts": 1,
    "constructed 1950": 1,
    "rescue optimus": 1,
    "saint christina": 1,
    "libre told": 1,
    "treason act": 1,
    "magazines cost": 1,
    "stopped broadcasting": 1,
    "state important": 2,
    "season characters": 1,
    "council constantinople": 1,
    "children minutes": 1,
    "use verb": 1,
    "greenhouse concentrated": 1,
    "history track": 1,
    "asia lesser": 1,
    "nbc ratings": 1,
    "2005 cuatro": 1,
    "fuzziness rubbing": 1,
    "shin power": 1,
    "mayor city": 8,
    "contending contained": 1,
    "shinanah qassim": 1,
```

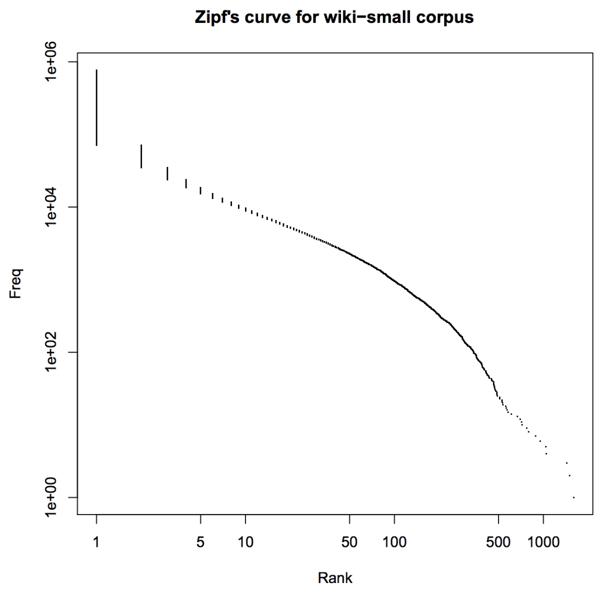

Figure 1: Zipf's curve for n-grams

In order to plot rank-frequency for words and bigrams in in the wikipedia collection, I took the following steps:

1. I read the text files to get HTML text, this is demonstrated in line 35 of listing 3

2. I used the *boilerplate* removal library (Justext) to extract plain text.

3. I used sklearn *CountVectorizer* to get 1 and 2-grams vocabulary. This is demonstrated in listing 3, line 41-47.

4. I stored the vocabulary (1-2 grams) into a CSV file with. This is can be seen in listing 3.

5. I plotted the *1-2-grams.csv* with an Rscript and the graph is shown in Figure 1.

6. The best c values are 0.1 and 0.15. The graph in Figure 1 shows that the frequently occuring c value lies where the graph goes on the horizontal line and that occurs half way between 0 and 0.2 which can be 0.1 and 0.15.

Problem 4

4.2. Plot vocabulary growth for the Wikipedia collection and estimate the parameters for Heaps law. Should the order in which the documents are processed make any difference?

Solution 4:

Listing 5: Vocabulary growth

```
from P2 import getHTMLPaths

from common import readTextFromFile
from common import getTextFromHTML

from sklearn.feature_extraction.text import CountVectorizer

def getHeapsData(filenames):


    outfile = open('vocabWordCount.csv', 'w')
    outfile.write('Vocab,WordCount\n')

    totalVocab = set()
    wordCount = 0
    for i in range( len(filenames) ):
        f = filenames[i].strip()
        html = readTextFromFile(f)
        text = getTextFromHTML(html)

        if( len(text) == 0 ):
            continue

        countVectorizer = CountVectorizer( min_df=1, stop_words='english',
        ngram_range=(1,2) )
        termFreqMat = countVectorizer.fit_transform([text])

        totalVocab = totalVocab.union( set(countVectorizer.vocabulary_.keys()) )
        wordCount += termFreqMat.todense().sum()
```

Problem 4 continued on next page. . .

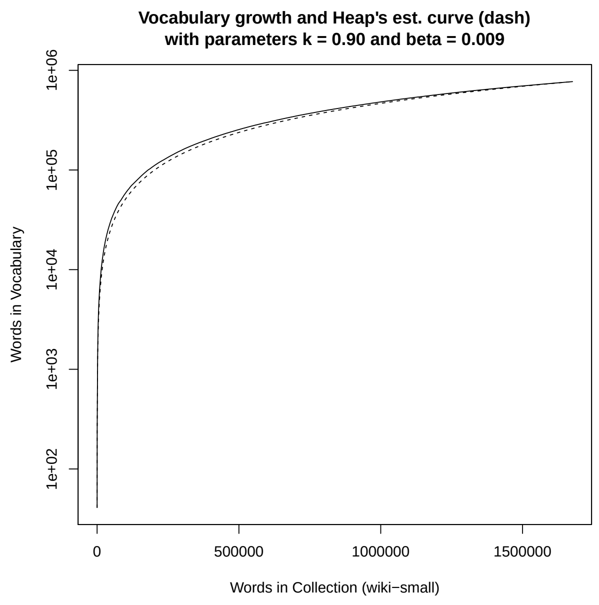

Figure 2: Vocabulary Growth Plot

```
            outfile.write( str(len(totalVocab)) + ', ' + str(wordCount) + '\n' )

            if( i % 100 == 0 ):
                print( i, 'of', len(filenames) )

        outfile.close()

filenames = getHTMLPaths()
getHeapsData( filenames )
```

1. The *CountVectorizer* library is used to get the vocabulary which is a set of unique word, and the term frequency matrix. This is demonstrated in listing 5, line 24.

2. I kept accumulating the vocabulary by taking the union of the new vocabulary and the previous vocabulary from step 1. This is demonstrated in line 27, listing 5.

3. I used term frequency matrix to get the terms and the various frequencies and added it to get the word count as demonstrated in line 28, listing 5.

The order of documents processed does not make any difference because the order did not affect the size of the corpus. " Heaps law predicts that the number of new words will increase very rapidly when the corpus is small and will continue to increase indefinitely, but at a slower rate for larger corpora". It is size of the vocabulary that will have an effect .

The Heap's k and beta parameters shows that Heap's law is a good fit.
Estimated Heaps law parameters : k = 0.90 and beta = 0.009.

The vocabulary growth plot for Heap's law in Figure 2 shows that Heap's is a good fit with the estimated k

Problem 4 continued on next page. . .

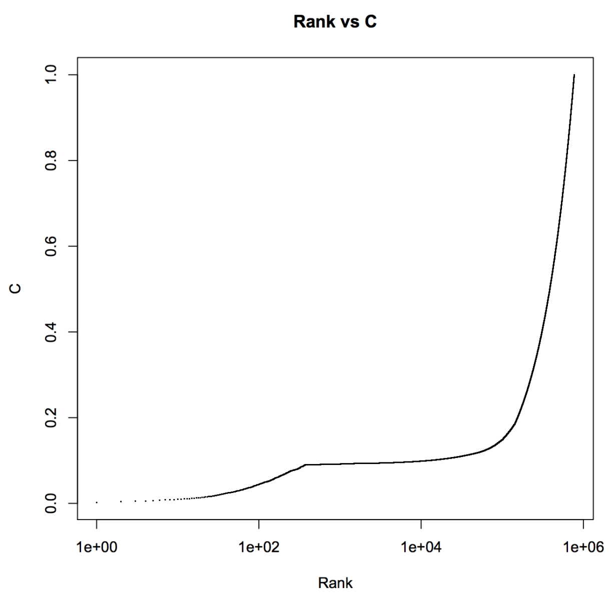

Figure 3: Heap's curve

and beta parameters

Problem 5

4.6. Process five Wikipedia documents using the Porter stemmer and the Krovetz stemmer. Compare the number of stems produced and find 10 examples of differences in the stemming that could have an impact on ranking

Solution 5:

Listing 6: Port and Krovetz stemming

```
from Porter import PorterStemmer
from krovetzstemmer import Stemmer

from common import readTextFromFile
from common import getTextFromHTML

krov = Stemmer()


f =  'en/articles/d/i/v/Divining_rod.html'
text = getTextFromHTML( readTextFromFile(f) )

print 'ori:\n', text, '\n'
print 'porter:\n', PorterStemmer.useStemer(text), '\n'
print 'krov:\n', krov.stem(text), '\n'
```

1. I read the text files to get HTML text.

2. I used the *boilerplate* removal library (Justext) to extract plain text.

3. I applied Porter and Krovetz stemmers to plain text of five documents from *wiki-small-files.txt*; this is demonstrated in listing 6, and the result is saved in wikiDoc1.txt, wikiDoc2.txt, wikiDoc3.txt, wikiDoc4.txt and wikiDoc5.txt.

4. I compared the output of Krovetz and Porter stemmers for the following 10 examples

Table 1: Stemming Comparison

Original Word	PorterStemmer Result	KrovetzStemmer Result
Archives	archiv	archives
Dublirer	dublir	dublirer
Federal	feder	federal
studied	studi	studied
Her husband was Robert Dublirer	her husband wa robert dublir	her husband was robert dublirer
Arts Project	art project	arts project
encyclopediaDorothy	encyclopediadorothi	encyclopediadorothy
Aristodemos	aristodemo	aristodemos
Aristodemos Kaldis	aristodemo kaldi	aristodemos kaldis
Art Students League	art student leagu	art students league

From the stemmed result in Table 1, Krovetz returned almost the same word as the original words. The result is meaningful and can be use as a query. Krovetz doesn't have so much impact on ranking.

Porter stem results are almost not English words. Most of the results cannot be identified with the original word. This will have impact on ranking. Krovetz will produce a better ranking.

From the example in Table 1, it Porter have more stemmed words than Krovetz.

References

[1] Countvectorizer. http://scikitlearn.org/stable/modules/feature_extraction.html. Accessed: 2017-10-10.

[2] Information Retrieval in Practice. http://bl.ocks.org/majetisiri/316e3a1537b469154779. Accessed: 2017-10-10.

[3] Justext. https://pypi.python.org/pypi/jusText/2.1.1. Accessed: 2017-10-10.

[4] Porterstemmer. https://pypi.python.org/pypi/PorterStemmer. Accessed: 2017-10-10.