# Assignment 2

CS834-F16: Introduction to Information Retrieval

Fall 2016

Erika Siregar

CS Department - Old Dominion University

October 13, 2016

## Question 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?`

### Answer

For this question, I use the 'Wiki small' test collections which can be downloaded from `http://www.searchengines-book.com`. Plotting the rank-frequency values can be done using these following steps:

1. Traverse the directory containing the test collections and list all the HTML files in that directory.
2. Get the text content from each HTML files using a python library 'html2text' [1]. I think using this library is simpler than using 'Beautiful Soup' [2]. It also gives us a cleaner 'html-stripped' result.
3. Tokenize the text using function '`text.split()`' and '`word.isalnum()`'. Each token is equal to one word.
4. Create the bigrams using function `nltk.bigrams()` provided by python library 'nltk' [3].
5. For both tokens and bigrams, do:
   (a) Count their frequencies (the number of times they appear in the whole collections).
   (b) Rank the tokens and the bigrams. The token or bigram that have the highest frequency will have rank = 1. The second highest frequency token or bigram will have rank = 2, and so on.
   (c) Compute their probability and the 'c' value.
6. Write the output to a csv file.

The complete code for the steps above can be seen in listing 1.

```python

#!/usr/bin/python
import io
import os

import html2text
import unicodecsv as csv
import nltk

html_files = []
# traverse the directory to list all the html files in the directory
for root, dirs, files in os.walk(os.path.abspath('./articles')):
    for file in files:
        if file.endswith('.html'):
            filepath = os.path.join(root, file)
            html_files.append(filepath)

#html_files = html_files[:10]
```

```python

tokens = []
# process each html file
for idx, file in enumerate(html_files):
    print('{} of {}. Processing {}'.format(idx+1, len(html_files), file))
    print('=' * 30)

    # get text only from each file -> remove all tags
    h = html2text.HTML2Text()
    h.ignore_links = True
    text = h.handle(u' '.join([line.strip() for line in io.open(file, "r", encoding="
    utf-8").readlines()]))

    # get all words from text by splitting by whitespace
    for word in text.split():
        if word.isalpha():
            tokens.append(word)

# create bigrams from tokens
bigrams = list(nltk.bigrams(tokens))

# count the frequency for each word
counts = {}
for token in tokens:
    counts.setdefault(token, 0)
    counts[token] += 1

# count the frequency for each bigram
counts2 = {}
for token in bigrams:
    counts2.setdefault(token, 0)
    counts2[token] += 1

# convert dict to 2d list
table = []
for count in counts:
    table.append([count, counts[count]])

table2 = []
for count2 in counts2:
    table2.append([count2, counts2[count2]])

# sort list by freq (2nd column)
table = sorted(table, key=lambda x:x[1], reverse=True)
table2 = sorted(table2, key=lambda x:x[1], reverse=True)

# add columns :
# - rank (3rd col)
# - prob (4th col)
# - c (5th col)
tmp_table=[]
for idx, row in enumerate(table):
    rank = idx + 1
    prob = float(row[1]) / len(tokens)
    c = rank * prob
    tmp_table.append(row + [rank, prob, c])

tmp_table2=[]
for idx2, row2 in enumerate(table2):
    rank = idx2 + 1
    prob = float(row2[1]) / len(bigrams)
    c = rank * prob
    row2[0] = ', '.join(row2[0])
    tmp_table2.append(row2 + [rank, prob, c])

# write the output to csv file
out_file = os.path.join(os.getcwd(), 'rank_freq.csv')
with open(out_file, "wb") as f:
    writer = csv.writer(f)
    writer.writerow(["word", "frequency", "rank", "prob", "c"])
    writer.writerows(tmp_table)

out_file2 = os.path.join(os.getcwd(), 'rank_freq_bigram.csv')
with open(out_file2, "wb") as f2:
    writer = csv.writer(f2)
    writer.writerow(["bigram", "frequency", "rank", "prob", "c"])
    writer.writerows(tmp_table2)

print('number of html files that are processed {}'.format(len(html_files)))
```

Listing 1: Tokenizing the content of Wikipedia collection

Table 1 and 2 show the top 20 words and top 20 bigrams with the highest ranks, respectively. The complete tables for the words and bigrams ranks are uploaded on github ('rank_freq_rev1.csv' and 'rank_freq_bigram_rev1.csv').

| word | frequency | rank | prob | c |
|---|---|---|---|---|
| the | 164719 | 1 | 0.0557882337 | 0.0557882337 |
| of | 117749 | 2 | 0.0398800911 | 0.0797601823 |
| and | 77442 | 3 | 0.0262286221 | 0.0786858662 |
| a | 60672 | 4 | 0.020548836 | 0.082195344 |
| in | 58548 | 5 | 0.0198294642 | 0.0991473208 |
| to | 53620 | 6 | 0.0181604131 | 0.1089624789 |
| is | 40996 | 7 | 0.0138848246 | 0.0971937725 |
| by | 39665 | 8 | 0.0134340318 | 0.1074722547 |
| Wikipedia | 38128 | 9 | 0.0129134695 | 0.1162212251 |
| was | 29307 | 10 | 0.0099259088 | 0.0992590877 |
| for | 25666 | 11 | 0.0086927483 | 0.0956202313 |
| on | 25190 | 12 | 0.0085315331 | 0.1023783977 |
| The | 24856 | 13 | 0.0084184116 | 0.1094393506 |
| as | 16526 | 14 | 0.0055971464 | 0.078360049 |
| with | 16087 | 15 | 0.0054484626 | 0.0817269395 |
| from | 13328 | 16 | 0.0045140244 | 0.0722243898 |
| Current | 12344 | 17 | 0.0041807561 | 0.071072853 |
| About | 12340 | 18 | 0.0041794013 | 0.0752292236 |
| registered | 12148 | 19 | 0.0041143733 | 0.0781730936 |
| that | 12025 | 20 | 0.0040727148 | 0.0814542962 |

Table 1: Most frequent 20 words from Wikipedia Collection (Wiki small)

| **bigram** | **frequency** | **rank** | **prob** | **c** |
|---|---|---|---|---|
| of, the | 39363 | 1 | 0.0133317528 | 0.0133317528 |
| in, the | 15699 | 2 | 0.0053170538 | 0.0106341075 |
| is, a | 14030 | 3 | 0.0047517845 | 0.0142553534 |
| a, registered | 12098 | 4 | 0.0040974404 | 0.0163897615 |
| About, Wikipedia | 12086 | 5 | 0.0040933761 | 0.0204668806 |
| by, Wikipedia | 10932 | 6 | 0.0037025308 | 0.0222151851 |
| to, the | 7672 | 7 | 0.0025984099 | 0.018188869 |
| under, the | 6804 | 8 | 0.0023044292 | 0.0184354335 |
| From, the | 6149 | 9 | 0.0020825889 | 0.0187433003 |
| terms, of | 6144 | 10 | 0.0020808955 | 0.0208089549 |
| the, free | 6105 | 11 | 0.0020676867 | 0.0227445535 |
| is, available | 6098 | 12 | 0.0020653159 | 0.0247837904 |
| for, is | 6083 | 13 | 0.0020602356 | 0.0267830622 |
| by, This | 6082 | 14 | 0.0020598969 | 0.0288385562 |
| the, terms | 6063 | 15 | 0.0020534618 | 0.0308019271 |
| text, is | 6057 | 16 | 0.0020514297 | 0.0328228749 |
| was, last | 6053 | 17 | 0.0020500749 | 0.0348512739 |
| This, page | 6052 | 18 | 0.0020497362 | 0.0368952524 |
| the, GNU | 6049 | 19 | 0.0020487202 | 0.0389256835 |
| the, Wikimedia | 6048 | 20 | 0.0020483815 | 0.04096763 |

Table 2: Most frequent 20 bigrams from Wikipedia Collection (Wiki small)

We just finished doing the first part of our task, which are creating tokens and bigrams. Next step is plotting the rank-frequency values of the tokens and bigrams into a log-log graph. There are 3 graph that we will create using R [4]:

1. A log-log rank-frequency plot for the words.
2. A log-log rank-frequency plot for the bigrams.
3. The combination of log-log rank-frequency plot for words and bigrams.

Figures 1, 2, and 3 show the log-log rank-frequency plot for the words, bigrams, and the combination of words and bigrams, respectively. Instead of frequency, I use probability for the y-axis.

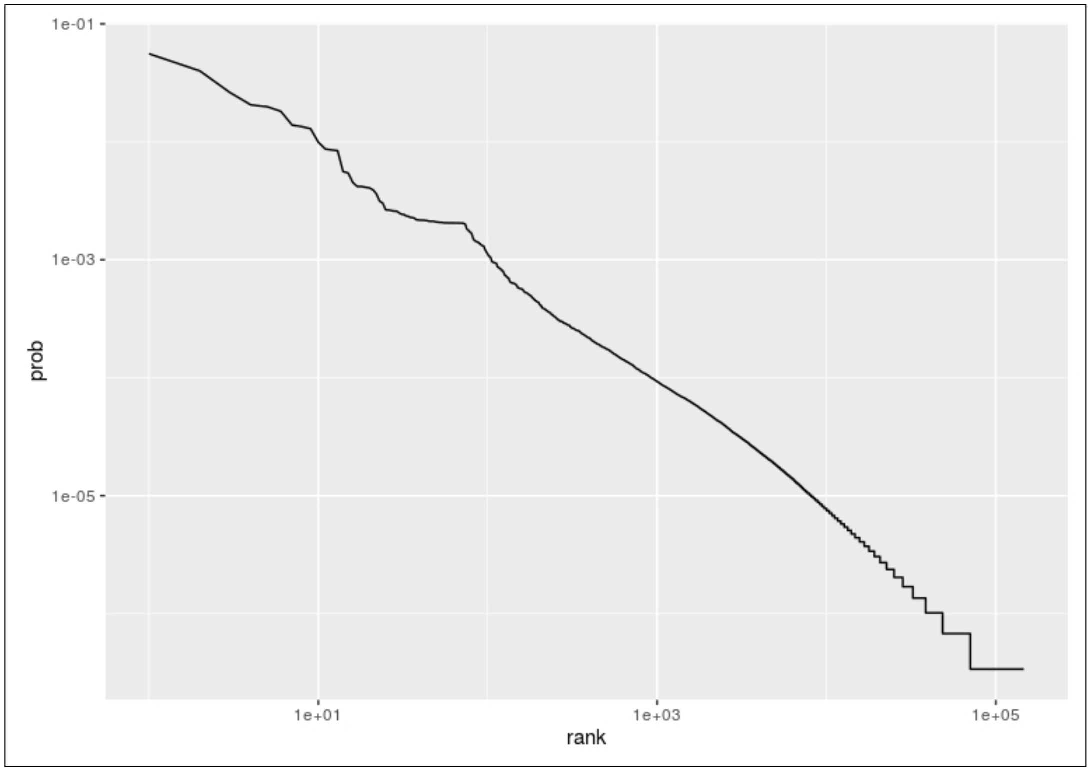


Figure 1: Log-log rank-frequency for words in Wikipedia Collection (Wiki small)

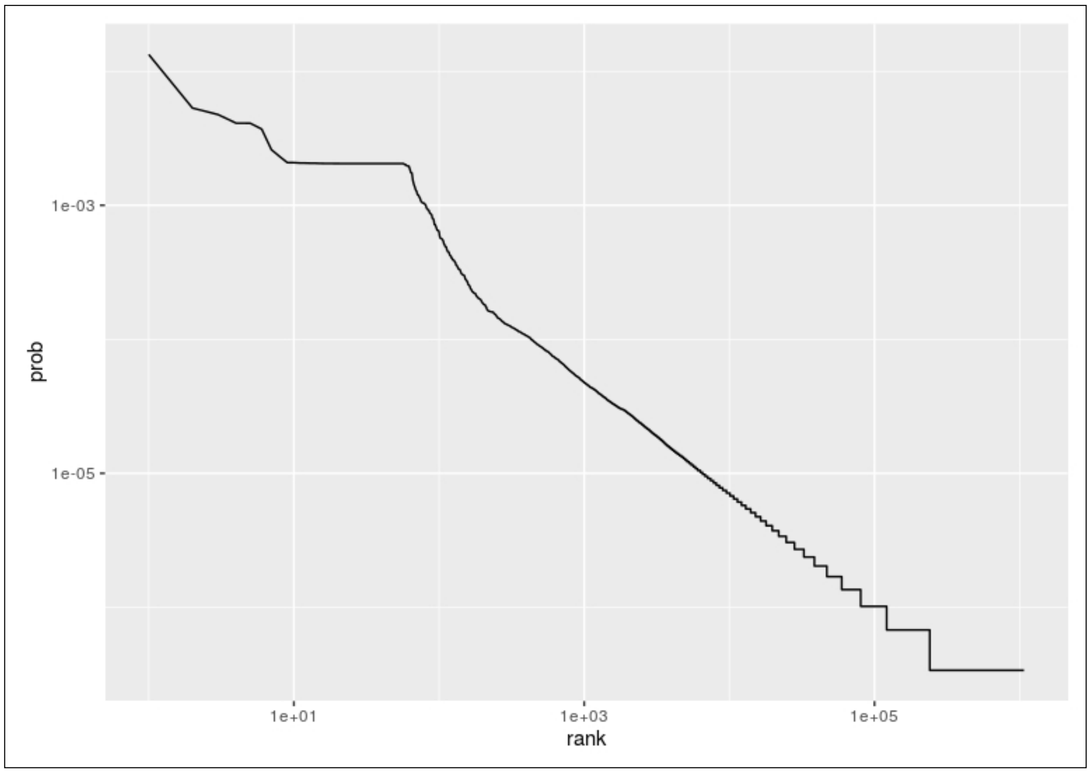


Figure 2: Log-log rank-frequency for bigrams in Wikipedia Collection (Wiki small)

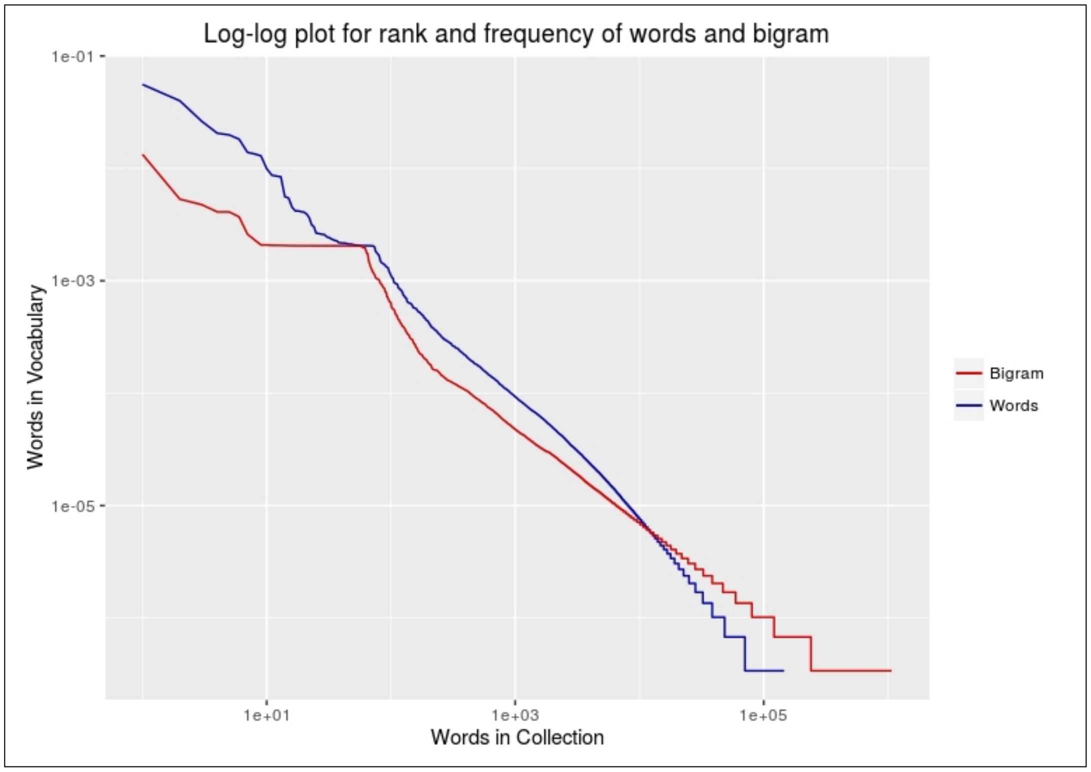


Figure 3: Log-log rank-frequency for bigrams and words in Wikipedia Collection (Wiki small)

The R code for creating the plot can be seen in listing:

```

require(ggplot2)

mydata <- read.csv("/home/erikaris/PycharmProjects/IR-A2/rank_freq.csv", head=TRUE,
    sep = ',')
ggplot(data=mydata, aes(x=rank, y=prob)) + geom_point() + scale_x_log10() + scale_y_
    log10()


#==========================================================================
#
require(ggplot2)

mydata <- read.csv("/home/erikaris/PycharmProjects/IR-A2/rank_freq_bigram.csv", head=
    TRUE, sep = ',')
ggplot(data=mydata, aes(x=rank, y=prob)) + geom_point() + scale_x_log10() + scale_y_
    log10()
#ggplot(data=mydata, aes(x=rank, y=prob)) + geom_point() + scale_x_log10() + scale_y_
    log10()




#==========================================================================
# combination

require(ggplot2)

```

```
mydata1 <- read.csv("rank_freq_rev1.csv", head=TRUE, sep = ',')
mydata2 <- read.csv("rank_freq_bigram_rev1.csv", head=TRUE, sep = ',')

ggplot(data=mydata1, aes(x=rank, y=prob)) + geom_line(data=mydata1, aes(x=rank, y=
    prob, color="Words")) + geom_line(data=mydata2, aes(x=rank, y=prob, color="Bigram
    ")) + scale_colour_manual(name='', values=c('Words'='#000099', 'Bigram'='#CC0000'
    ), guide='legend') + scale_x_log10() + scale_y_log10() + labs(title='Log-log plot
     for rank and frequency of words and bigram',x = 'Words in Collection', y = '
    Words in Vocabulary')
```

Listing 2: Code for plotting vocabulary growth and calculating the Heap's parameter

From the plots we can see that the best value for c is 0.1.

---

## Question 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?`

### Answer:

The idea to solve this problem is simply calculated the number of words in every document and store it as 'corpus' (words in collection). At the same time, we also calculate the number of unique words in every document and store it as 'vocabulary'. Then, we can create the plot of vocabulary growth using R. Figure 4 show the plot for vocabulary growth for the Wikipedia collection.

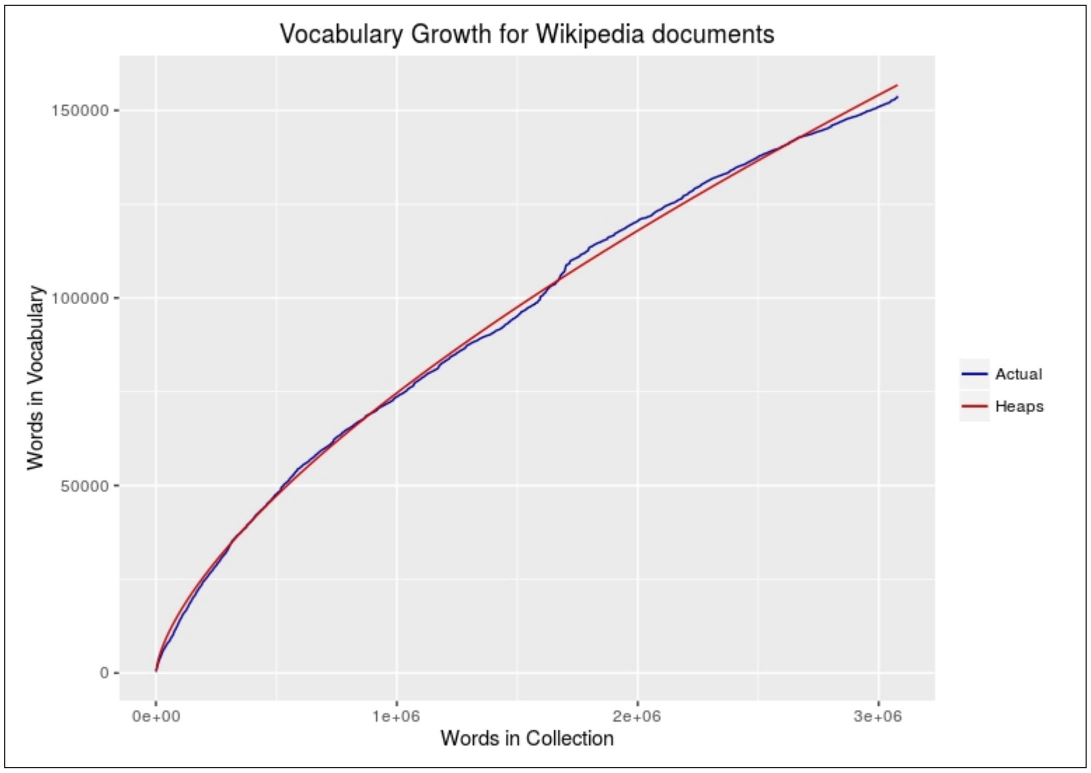


Figure 4: Vocabulary growth for the Wikipedia Collection (Wiki small)

To find out whether or not the order of document processing affect the vocabulary growth, I modify the code so that the documents are being processed in reversed order. Figure 5 shows the plot for vocabulary growth for the Wikipedia collection in reversed order.

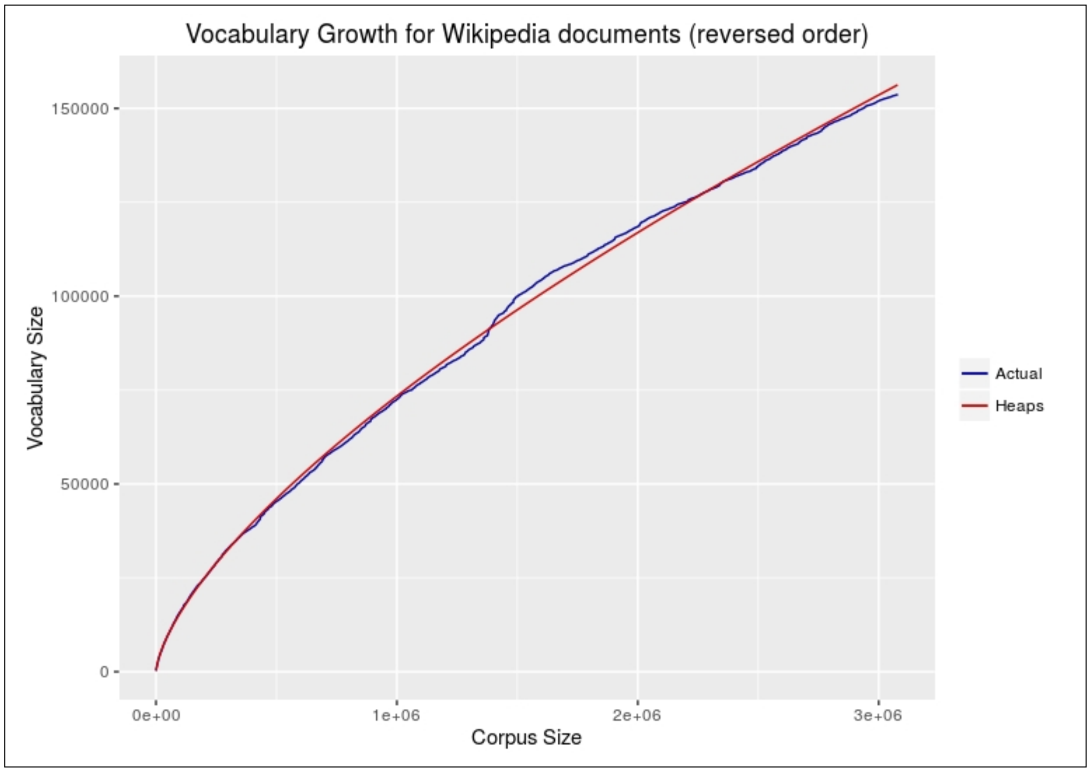


Figure 5: Vocabulary growth for the Wikipedia Collection (Wiki small)

The complete code for calculating the vocabulary growth can be seen in listing 3

```python
#!/usr/bin/python
import io
import os
import html2text
import unicodecsv as csv

html_files = []
# traverse the directory to list all the html files in the directory
for root, dirs, files in os.walk(os.path.abspath('./articles')):
    for file in files:
        if file.endswith('.html'):
            filepath = os.path.join(root, file)
            html_files.append(filepath)

corpus = []
voc_corpus = []
# process each html file
for idx, file in enumerate(html_files):
    print('{} of {}. Processing {}'.format(idx + 1, len(html_files), file))
    print('=' * 30)

    # get text only from each file -> remove all tags
    h = html2text.HTML2Text()
    h.ignore_links = True
    text = h.handle(u' '.join([line.strip() for line in io.open(file, "r", encoding="utf-8").readlines()]))

    # get all words from text by splitting by whitespace
    tokens = []
    for word in text.split():
        if word.isalnum():
            tokens.append(word)

    # corpus is cummulative of tokens
    corpus += tokens
    # voc is unique list of corpus
    vocs = set(corpus)

    # count the size of corpus and vocabularies in the docs[file]
    voc_corpus.append([idx+1, len(corpus), len(vocs)])

print('\n\n the size of corpus {}'.format(len(corpus)))
print('\n\n the size of vocabularies {}'.format(len(vocs)))

out_file = os.path.join(os.getcwd(), '4_2-voc_corpus.csv')
with open(out_file, "wb") as f:
    writer = csv.writer(f)
    writer.writerow(["docs", "corpus_size", "vocabulary_size"])
    writer.writerows(voc_corpus)


# Process reverse list
print('\n\n Processing reverse list...')
html_files.reverse()


corpus = []
voc_corpus = []
# process each html file
for idx, file in enumerate(html_files):
    print('{} of {}. Processing {}'.format(idx + 1, len(html_files), file))
    print('=' * 30)

    # get text only from each file -> remove all tags
    h = html2text.HTML2Text()
    h.ignore_links = True
    text = h.handle(u' '.join([line.strip() for line in io.open(file, "r", encoding="utf-8").readlines()]))

    # get all words from text by splitting by whitespace
    tokens = []
    for word in text.split():
        if word.isalnum():
            tokens.append(word)

    # corpus is cummulative of tokens
    corpus += tokens
    # voc is unique list of corpus
    vocs = set(corpus)

    # count the size of corpus and vocabularies in the docs[file]
    voc_corpus.append([idx+1, len(corpus), len(vocs)])

```

```
print('\n\n the size of corpus in reverse order {}'.format(len(corpus)))
print('\n\n the size of vocabularies in reverse order {}'.format(len(vocs)))

out_file = os.path.join(os.getcwd(), '4_2-voc_corpus_reverse.csv')
with open(out_file, "wb") as f:
    writer = csv.writer(f)
    writer.writerow(["docs", "corpus_size", "vocabulary_size"])
    writer.writerows(voc_corpus)
```

Listing 3: Source code for calculating vocabulary growth

The relationship between the size of corpus and the size of vocabulary was empirically defined by Heap [5] to be:

$$v = kn^{\beta}$$

. Estimation of the parameters for Heap's law are done using non-linear least square (nls) [6], which is available in R. Table 3 show the parameter of Heap's law both for ascending and descending order. From figure 4 and 5 we can see that both plots look the same. There is a sligt difference in the parameter values between documents the reversed-order documents and the non-reversed-order documents. But, this difference is not really significant. Therefore, we can conclude that the order in which the documents are processed does not really make any difference towards the vocabulary growth.

| No | Parameter | Values |
|---|---|---|
| 1 | k ascending | 8.223226 |
| 2 | b ascending | 0.659675 |
| 3 | k descending | 6.813402 |
| 4 | b descending | 0.672038 |
| 5 | number of documents | 6043 |

Table 3: Heap parameter for Wikipedia Collection

The R code for creating the plot and calculating the parameter value can be seen in listing 4.

```

require(ggplot2)

mydata <- read.csv("4_2-voc_corpus_rev1.csv", head=TRUE, sep = ',')
x <- mydata$corpus_size
y <- mydata$vocabulary_size

#model
fit<-nls(y~k*(x^b), data = mydata, start = list(k=1,b=1))
summary(fit)
#get some estimation of goodness of fit
cor(y, predict(fit))

ggplot(data=mydata, aes(x=corpus_size, y=vocabulary_size)) + geom_line(aes(group =
    1, color="Actual")) + geom_line(data=mydata, aes(x=corpus_size, y=predict(fit),
    color="Heaps")) + labs(title='Vocabulary Growth for Wikipedia documents',x = '
    Words in Collection', y = 'Words in Vocabulary') + scale_colour_manual(name='',
    values=c('Actual'='#000099', 'Heaps'='#CC0000'), guide='legend')




#==================================================================================
# the reverse order

require(ggplot2)

mydata <- read.csv("4_2-voc_corpus_reverse_rev1.csv", head=TRUE, sep = ',')
x <- mydata$corpus_size
y <- mydata$vocabulary_size

#model
fit<-nls(y~k*(x^b), data = mydata, start = list(k=1,b=1))
summary(fit)
#get some estimation of goodness of fit
cor(y, predict(fit))

ggplot(data=mydata, aes(x=corpus_size, y=vocabulary_size))  + geom_line(aes(group =
    1, color="Actual")) + geom_line(data=mydata, aes(x=corpus_size, y=predict(fit),
    color="Heaps")) + labs(title='Vocabulary Growth for Wikipedia documents (reversed
     order)',x = 'Corpus Size', y = 'Vocabulary Size') + scale_colour_manual(name='',
     values=c('Actual'='#000099', 'Heaps'='#CC0000'), guide='legend')
```

Listing 4: Code for plotting vocabulary growth and calculating the Heap's parameter

---

## Question 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.`

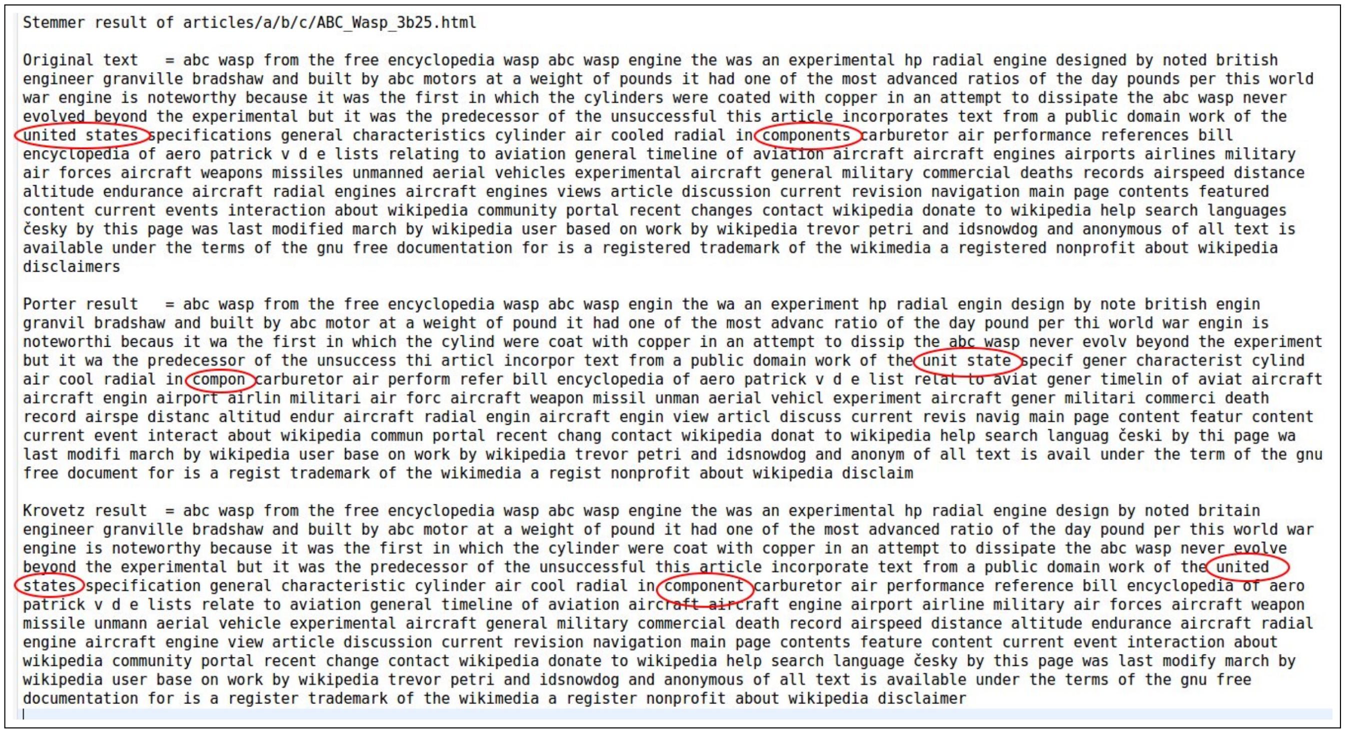
### Answer

Stemming using Porter stemmer is quite easy to do since this stemmer is provided by nltk [7]. Fortunately, there is also a python library for stemming with Krovetz algorithm [8]. So, our task now is to create a script that utilizes these 2 libraries. The logic is simple: get the text content of the documents and use Porter and Krovetz for stemming.

To do the stemming, I randomly choose 5 documents from the Wikipedia collections. These 5 documents are:

1. ABC_Wasp_3b25.html
2. ABC_In_Concert_6d5f.html
3. Abdus_Salam_(disambiguation)_0602.html
4. Abd-Allah_ibn_Amr_f58f.html
5. Abdul_Haq_Vidyarthi_582b.html

Figure 6 shows the comparison of stemming result between Porter and Krovetz. I circle some of the words that I consider will affect the ranking. The complete stemming result for all 5 documents is uploaded on github under a file named '4_6-result_rev3.txt'.



```
Stemmer result of articles/a/b/c/ABC_Wasp_3b25.html

Original text   = abc wasp from the free encyclopedia wasp abc wasp engine the was an experimental hp radial engine designed by noted british
engineer granville bradshaw and built by abc motors at a weight of pounds it had one of the most advanced ratios of the day pounds per this world
war engine is noteworthy because it was the first in which the cylinders were coated with copper in an attempt to dissipate the abc wasp never
evolved beyond the experimental but it was the predecessor of the unsuccessful this article incorporates text from a public domain work of the
united states specifications general characteristics cylinder air cooled radial in components carburetor air performance references bill
encyclopedia of aero patrick v d e lists relating to aviation general timeline of aviation aircraft aircraft engines airports airlines military
air forces aircraft weapons missiles unmanned aerial vehicles experimental aircraft general military commercial deaths records airspeed distance
altitude endurance aircraft radial engines aircraft engines views article discussion current revision navigation main page contents featured
content current events interaction about wikipedia community portal recent changes contact wikipedia donate to wikipedia help search languages
česky by this page was last modified march by wikipedia user based on work by wikipedia trevor petri and idsnowdog and anonymous of all text is
available under the terms of the gnu free documentation for is a registered trademark of the wikimedia a registered nonprofit about wikipedia
disclaimers

Porter result   = abc wasp from the free encyclopedia wasp abc wasp engin the wa an experiment hp radial engin design by note british engin
granvil bradshaw and built by abc motor at a weight of pound it had one of the most advanc ratio of the day pound per thi world war engin is
noteworthi becaus it wa the first in which the cylind were coat with copper in an attempt to dissip the abc wasp never evolv beyond the experiment
but it wa the predecessor of the unsuccess thi articl incorpor text from a public domain work of the unit state specif gener characterist cylind
air cool radial in compon carburetor air perform refer bill encyclopedia of aero patrick v d e list relat to aviat gener timelin of aviat aircraft
aircraft engin airport airlin militari air forc aircraft weapon missil unman aerial vehicl experiment aircraft gener militari commerci death
record airspe distanc altitud endur aircraft radial engin aircraft engin view articl discuss current revis navig main page content featur content
current event interact about wikipedia commun portal recent chang contact wikipedia donat to wikipedia help search languag česki by thi page wa
last modifi march by wikipedia user base on work by wikipedia trevor petri and idsnowdog and anonym of all text is avail under the term of the gnu
free document for is a regist trademark of the wikimedia a regist nonprofit about wikipedia disclaim

Krovetz result  = abc wasp from the free encyclopedia wasp abc wasp engine the was an experimental hp radial engine design by noted britain
engineer granville bradshaw and built by abc motor at a weight of pound it had one of the most advanced ratio of the day pound per this world war
engine is noteworthy because it was the first in which the cylinder were coat with copper in an attempt to dissipate the abc wasp never evolve
beyond the experimental but it was the predecessor of the unsuccessful this article incorporate text from a public domain work of the united
states specification general characteristic cylinder air cool radial in component carburetor air performance reference bill encyclopedia of aero
patrick v d e lists relate to aviation general timeline of aviation aircraft aircraft engine airport airline military air forces aircraft weapon
missile unmann aerial vehicle experimental aircraft general military commercial death record airspeed distance altitude endurance aircraft radial
engine aircraft engine view article discussion current revision navigation main page contents feature content current event interaction about
wikipedia community portal recent change contact wikipedia donate to wikipedia help search language česky by this page was last modify march by
wikipedia user base on work by wikipedia trevor petri and idsnowdog and anonymous of all text is available under the terms of the gnu free
documentation for is a register trademark of the wikimedia a register nonprofit about wikipedia disclaimer
```

Figure 6: Stemming comparison (Porter vs Krovetz)

Table 5 show 10 differences in the stemming between Porter and Krovetz that could have impact on ranking. From this table we can see that Porter arbitraly dissect the words. For example, Porter stem the word 'united' into 'unit', which have different meaning. Moreover, when this word is combined with the word next to it, they form different term. The term 'united states' is clearly different with 'unit state'. Therefore, we can see that it will definetely impact the ranking. The comparison of the number of stems produces by both Porter and Krovetz can be seen on table 4. From table 4 we can see that Krovetz produced more stems compare to that of Porter. This is because Porter is more naive in 'stemming' the word. So, there could be two words that are grouped together into the same stem while, in fact, they do not belong in the same stem.

| **No** | **document** | **number of stems** | |
|---|---|---|---|
| | | Porter | Krovetz |
| 1 | ABC_Wasp_3b25.html | 146 | 148 |
| 2 | ABC_In_Concert_6d5f.html | 138 | 142 |
| 3 | Abdus_Salam_(disambiguation)_0602.html | 91 | 93 |
| 4 | Abd-Allah_ibn_Amr_f58f.html | 471 | 480 |
| 5 | Abdul_Haq_Vidyarthi_582b.html | 165 | 170 |
| **Total** | | **1011** | **1033** |

Table 4: The number of stems produced by Porter and Krovetz

| No | Original | Porter | Krovetz |
|---|---|---|---|
| 1 | united states | unit state | united states |
| 2 | available | avail | available |
| 3 | engine | engin | engine |
| 4 | component | compon | component |
| 5 | airspeed | airspe | airspeed |
| 6 | movie | movi | movie |
| 7 | tradition | tradit | tradit |
| 8 | eventually | eventu | eventually |
| 9 | since | sinc | since |
| 10 | navigation | navig | navigation |

Table 5: Examples of differences in stemming between Porter and Krovetz

The source code for processing the Porter and Krovetz stemmer can be seen in listing 5.

```

#!/usr/bin/python
import io
import sys

import html2text as html2text
import krovetzstemmer
from nltk import PorterStemmer

# Instantiate porter stemmer
porter = PorterStemmer()
krovetz = krovetzstemmer.Stemmer()

if len(sys.argv) < 6:
    print('Usage :')
    print('python 4_6.py <file_1> ... <file_5>')

# Assuming all arguments are file
files = []
for arg in range(1, len(sys.argv)):
    files.append(sys.argv[arg])

# Get contents of each file
results = {}
for idx, file in enumerate(files):
    print('{} of {}. Processing {}'.format(idx + 1, len(files), file))
    print('=' * 30)

    # get text content
    h = html2text.HTML2Text()
    h.ignore_links = True
    text = h.handle(u' '.join([line.strip() for line in io.open(file, "r", encoding="
    utf-8").readlines()]))

    # remove whitespace
    words = []
    for word in text.split():
        if word.isalpha():
            words.append(word.lower())
    text = u' '.join(words)

    porter_result = []
    krovetz_result = []
    for c in words:
        porter_result.append(porter.stem(c))
        krovetz_result.append(krovetz.stem(c))

    results[file] = {}
    results[file]['original'] = text
    results[file]['porter'] = u' '.join(porter_result)
    results[file]['krovetz'] = u' '.join(krovetz_result)

# print results
txt_results = []
for file in results:
    txt_results.append(u'Stemmer result of {}'.format(file))
    txt_results.append(u'{}'.format('=' * 60))
    txt_results.append(u'Original text \t= {}\n'.format(results[file]['original']))
    txt_results.append(u'Porter result \t= {}\n'.format(results[file]['porter']))
    txt_results.append(u'Krovetz result \t= {}\n'.format(results[file]['krovetz']))

    num_stems_porter = len(set(results[file]['porter'].split()))
    txt_results.append(u'Number of stems produced by Porter \t= {}\n'.format(
    num_stems_porter))

    num_stems_krovetz = len(set(results[file]['krovetz'].split()))
    txt_results.append(u'Number of stems produced by Krovetz \t= {}\n'.format(
    num_stems_krovetz))
    txt_results.append(u'\n')

    print(u'\n'.join(txt_results))

    # also write to file
    f = io.open('4_6-result.txt', "w", encoding="utf-8")
    for txt_result in txt_results:
        f.write(txt_result + '\n')
```

Listing 5: Source code for stemming document using Porter Stemmer and Krovetz Stemmer

---

## Question 4.8

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

### Answer

Here is the algorithm to find the document with the most inlinks:

1. Use Beautiful Soup [2] to extract all links from the Wikipedia documents.
2. Store the links in a key-value format, where key = document and value = link.
3. Group and count the links by the destination (the one that is written in 'href'). This will be the number of inlinks for each destination.
4. Sort the data by the number of inlinks in a descending order.

Table 6 shows 10 Wikipedia documents with the most inlinks and their anchor text.

| No | Link | Inlinks | Anchor Text |
|---|---|---|---|
| 1 | Brazil.html | 83 | Brazil, BRA, Brazilian |
| 2 | August_26.html | 25 | 08-26, 26, August 26, 26 August |
| 3 | Manga.html | 14 | manga, Manga |
| 4 | Magazine.html | 13 | magazine, magazines, Magazine |
| 5 | Mollusca.html | 12 | Mollusca |
| 6 | Victoria_of_the_United_Kingdom_5e8e.html | 8 | Queen Victoria, Queen, Victoria of the United Kingdom, Victoria |
| 7 | Screenwriter.html | 7 | Writer(s), screenwriter, Screenwriter |
| 8 | Kidney.html | 6 | kidneys, Renal, kidney |
| 9 | Tottenham_Hotspur_F.C._6bd2.html | 5 | Tottenham Hotspur, Tottenham |
| 10 | Tuscany.html | 4 | Tuscany |

Table 6: Wikipedia documents with the most inlinks

The complete source code for processing the inlinks can be seen in listing 6.

```

#!/usr/bin/python
import os
from pprint import pprint

import unicodecsv as csv
from bs4 import BeautifulSoup
from tabulate import tabulate

html_files = []
# traverse the directory to list all the html files in the directory
for root, dirs, files in os.walk(os.path.abspath('./articles')):
    for file in files:
        if file.endswith('.html'):
            filepath = os.path.join(root, file)
            html_files.append(filepath)

all_links = {}
all_anchor_text = {}
# process each html file
for idx, file in enumerate(html_files):
    # just for debugging
    print('{} of {}. Processing {}'.format(idx+1, len(html_files), file))
    print('=' * 30)

    # find all anchors
    soup = BeautifulSoup(open(file), 'html.parser')
    anchors = soup.find_all('a', href=True)
    print('Found {} anchors'.format(len(anchors)))

    for a in anchors:
        link = a['href']
        # anchor text
        text = a.string or ''

        # In this case, link is relative path points to other html file
        # just process non http link
```

```python
        if not link.startswith('http'):
            try:
                # convert to absolute path
                link = os.path.join(os.path.dirname(file), link)
                link = os.path.abspath(link)
            except:
                pass

        # all_link : key is source, value is list of destination
        all_links.setdefault(file, [])
        # all_anchor_text : key is source, value is list of anchor text
        all_anchor_text.setdefault(file, [])

        # append only if:
        # - file != link
        # - all_links do not contain link
        # - link is file --> ignore http
        if file != link and link not in all_links[file] and os.path.isfile(link):
            all_links[file].append(link)
            all_anchor_text[file].append(text)

link_freq = {}
link_text = {}
for src in all_links:
    # all destinations in each src
    dests = all_links[src]
    # all anchor text in each src
    texts = all_anchor_text[src]

    for idx, dest in enumerate(dests):
        link_freq.setdefault(dest, 0)
        link_freq[dest] += 1

        link_text.setdefault(dest, [])
        link_text[dest].append(texts[idx])

# convert dict to 2d list
link_freq_table = []
for link in link_freq:
    link_freq_table.append([link, link_freq[link]])

# sort list by freq (2nd column)
link_freq_table = sorted(link_freq_table, key=lambda x:x[1], reverse=True)

# append anchor texts in 3rd column
tmp_link_freq_table = []
for row in link_freq_table:
    # append anchor texts in 3rd column
    row += [u', '.join(set(link_text[row[0]]))]
    # convert full-path link to filename only
    row[0] = os.path.basename(row[0])
    tmp_link_freq_table.append(row)
link_freq_table = tmp_link_freq_table

# process only top 10 results
link_freq_table = link_freq_table[:10]

# write the output to csv file
out_file = os.path.join(os.getcwd(), '4_8-link_freq.csv')
```

```
with open(out_file, "wb") as f:
    writer = csv.writer(f)
    writer.writerow(["link", "frequency", "texts"])
    writer.writerows(link_freq_table)

# print the resulting table
print tabulate(link_freq_table, headers=["link", "frequency", "texts"])
```

Listing 6: Source code for finding 10 Wikipedia documents with the most inlinks

---

## Question 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:

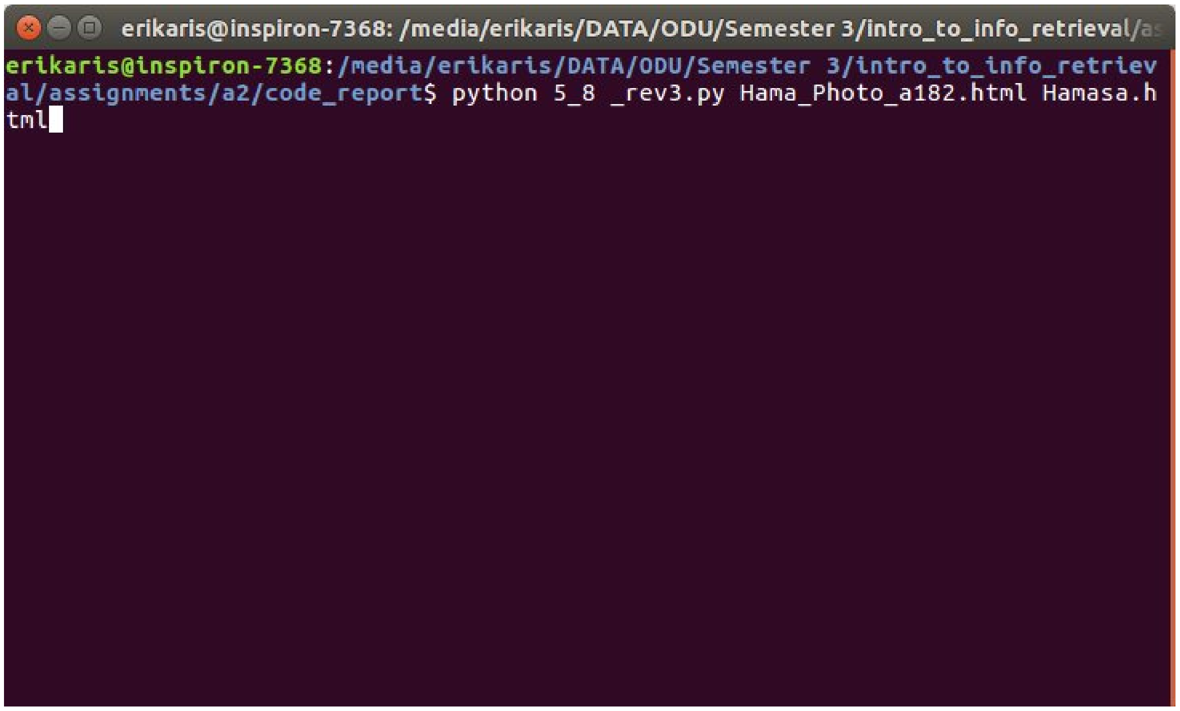
```
% ./your-program A B

blue B
brown A
fox A B
quick A
slow B
the A B
```

### Answer

To create an inverted index, we need to ‘swap’ the role of words vs documents. There are many nice examples that we can find on the internet about how to create inverted index. One that I find easier to understand is the one provided by RosettaCode.org [9]. The algorithm is quite simple:

1. List all words in each document. We will get a key-value pair, where document is the key and words are the value.
2. ‘Swap’ the role of words and documents. Now, word is the key and document is the value.
3. Write the output to a csv file.

The source code for the inverted index can be seen in listing 7. This code run using terminal input (figure 7).



```
erikaris@inspiron-7368: /media/erikaris/DATA/ODU/Semester 3/intro_to_info_retrieval/as
erikaris@inspiron-7368:/media/erikaris/DATA/ODU/Semester 3/intro_to_info_retriev
al/assignments/a2/code_report$ python 5_8 _rev3.py Hama_Photo_a182.html Hamasa.h
tml
```

Figure 7: Terminal input for running the inverted index

```python

#!/usr/bin/python
import unicodecsv as csv
import io
import os
import sys

import html2text
from tabulate import tabulate

# Assuming all arguments are file
files = []
for arg in range(1, len(sys.argv)):
    files.append(sys.argv[arg])

file_words_index = {}
all_words = set()

# Read all files
for file in files:
    # get text content
    h = html2text.HTML2Text()
    h.ignore_links = True
    text = h.handle(u' '.join([line.strip() for line in io.open(file, "r", encoding="
    utf-8").readlines()]))

    words = [word.lower() for word in text.split() if word.isalpha()]
    all_words |= set(words)
    file_words_index[file.split(os.pathsep)[-1]] = words

# Invert words and files
word_files_index = {}
```

```
for word in all_words:
    files = []
    for file, words in file_words_index.items():
        if word in words:
            files.append(file)
    word_files_index[word] = sorted(set(files))

# Convert to 2d array
table = []
for word in word_files_index:
    table.append([word, u', '.join(word_files_index[word])])

print tabulate(table, headers=["word", "files"])

# write the output to csv file
out_file = os.path.join(os.getcwd(), '5_8-inverted_index.csv')
with open(out_file, "wb") as f:
    writer = csv.writer(f)
    writer.writerow(["word", "files"])
    writer.writerows(table)
```

Listing 7: Source code for simple inverted index

Table 7 shows 20 first rows of the inverted index created from Wikipedia document ‘Hama_Photo_a182.html’ and ‘Hamasa.html’. These 2 Wikipedia documents are chosen randomly. The complete list of the inverted index is uploaded on github under file named ‘5_8-inverted_index_rev1.csv’.

| No | word | documents |
|---|---|---|
| 1 | all | articles/h/a/m/Hama_Photo_a182.html, articles/h/a/m/Hamasa.html |
| 2 | help | articles/h/a/m/Hama_Photo_a182.html, articles/h/a/m/Hamasa.html |
| 3 | german | articles/h/a/m/Hama_Photo_a182.html |
| 4 | photo | articles/h/a/m/Hama_Photo_a182.html |
| 5 | supported | articles/h/a/m/Hamasa.html |
| 6 | founded | articles/h/a/m/Hama_Photo_a182.html |
| 7 | including | articles/h/a/m/Hama_Photo_a182.html |
| 8 | filters | articles/h/a/m/Hama_Photo_a182.html |
| 9 | world | articles/h/a/m/Hama_Photo_a182.html |
| 10 | pvac | articles/h/a/m/Hama_Photo_a182.html |
| 11 | bombing | articles/h/a/m/Hama_Photo_a182.html |
| 12 | current | articles/h/a/m/Hama_Photo_a182.html, articles/h/a/m/Hamasa.html |
| 13 | based | articles/h/a/m/Hama_Photo_a182.html, articles/h/a/m/Hamasa.html |
| 14 | equipment | articles/h/a/m/Hama_Photo_a182.html |
| 15 | flash | articles/h/a/m/Hama_Photo_a182.html |
| 16 | hanke | articles/h/a/m/Hama_Photo_a182.html |
| 17 | languages | articles/h/a/m/Hama_Photo_a182.html |
| 18 | to | articles/h/a/m/Hama_Photo_a182.html, articles/h/a/m/Hamasa.html |
| 19 | under | articles/h/a/m/Hama_Photo_a182.html, articles/h/a/m/Hamasa.html |
| 20 | extensive | articles/h/a/m/Hama_Photo_a182.html |

Table 7: 20 first rows of the inverted index created from 2 Wikipedia documents

## References

[1] Alireza Savand. html2text 2016.9.19. `https://pypi.python.org/pypi/html2text`, 2016. [Online; accessed 11-October-2016].

[2] Leonard Richardson. Beautiful Soup. `https://www.crummy.com/software/BeautifulSoup/`, 2016. [Online; accessed 20-September-2016].

[3] NLTK Project. NLTK 3.0 documentation. `http://www.nltk.org/`, 2016. [Online; accessed 11-October-2016].

[4] The R Foundation. The R Project for Statistical Computing. `https://www.r-project.org/`, 2016. [Online; accessed 12-October-2016].

[5] Bruce Croft, Donald Metzler, and Trevor Strohman. *Search Engines: Information Retrieval in Practice*. Addison-Wesley Publishing Company, USA, 1st edition, 2009.

[6] Lionel Hertzog. First steps with Non-Linear Regression in R. `https://www.r-bloggers.com/first-steps-with-non-linear-regression-in-r/`, 2016. [Online; accessed 12-October-2016].

[7] NLTK Project. nltk.stem package. `http://www.nltk.org/api/nltk.stem.html#module-nltk.stem.porter`, 2016. [Online; accessed 11-October-2016].

[8] Ruey-Cheng Chen. KrovetzStemmer 0.4. `https://pypi.python.org/pypi/KrovetzStemmer/0.4`, 2016. [Online; accessed 12-October-2016].

[9] RosettaCode.org. Inverted index. `https://www.rosettacode.org/wiki/Inverted_index#Python`, 2016. [Online; accessed 12-October-2016].