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.isalpha()'`. 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.

```
1
2  #!/usr/bin/python
3  import io
4  import os
5
6  import html2text
7  import unicodecsv as csv
8  import nltk
9
10 html_files = []
11 # traverse the directory to list all the html files in the directory
12 for root, dirs, files in os.walk(os.path.abspath('./articles')):
13     for file in files:
14         if file.endswith('.html'):
15             filepath = os.path.join(root, file)
16             html_files.append(filepath)
17
18 #html_files = html_files[:10]
```

```
19
20 tokens = []
21 # process each html file
22 for idx, file in enumerate(html_files):
23     print('{} of {}. Processing {}'.format(idx+1, len(html_files), file))
24     print('=' * 30)
25
26     # get text only from each file -> remove all tags
27     h = html2text.HTML2Text()
28     h.ignore_links = True
29     text = h.handle(u' '.join([line.strip() for line in io.open(file, "r", encoding="
       utf-8").readlines()]))
30
31     # get all words from text by splitting by whitespace
32     for word in text.split():
33         if word.isalpha():
34             tokens.append(word)
35
36 # create bigrams from tokens
37 bigrams = list(nltk.bigrams(tokens))
38
39 # count the frequency for each word
40 counts = {}
41 for token in tokens:
42     counts.setdefault(token, 0)
43     counts[token] += 1
44
45 # count the frequency for each bigram
46 counts2 = {}
47 for token in bigrams:
48     counts2.setdefault(token, 0)
49     counts2[token] += 1
50
51 # convert dict to 2d list
52 table = []
53 for count in counts:
54     table.append([count, counts[count]])
55
56 table2 = []
57 for count2 in counts2:
58     table2.append([count2, counts2[count2]])
59
60 # sort list by freq (2nd column)
61 table = sorted(table, key=lambda x:x[1], reverse=True)
62 table2 = sorted(table2, key=lambda x:x[1], reverse=True)
63
64 # add columns :
65 # - rank (3rd col)
66 # - prob (4th col)
67 # - c (5th col)
68 tmp_table=[]
69 for idx, row in enumerate(table):
70     rank = idx + 1
71     prob = float(row[1]) / len(tokens)
72     c = rank * prob
73     tmp_table.append(row + [rank, prob, c])
74
75 tmp_table2=[]
76 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. 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 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.

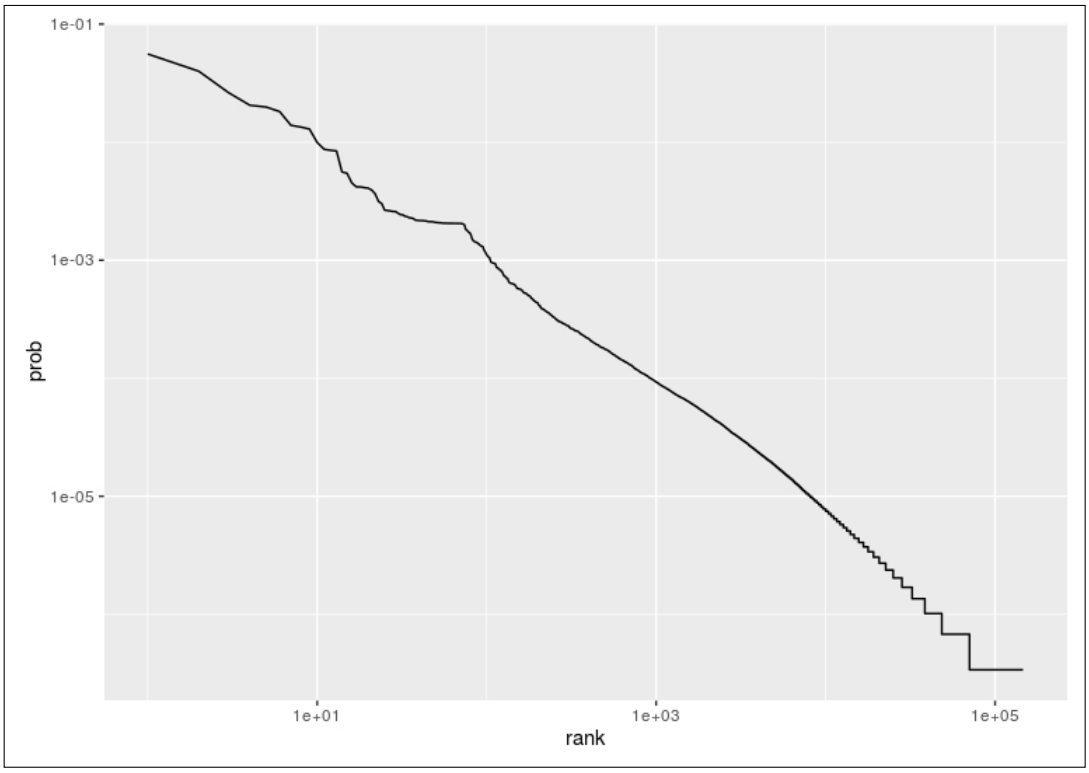

Figure 1: Log-log rank-frequency for words in Wikipedia Collection (Wiki small)

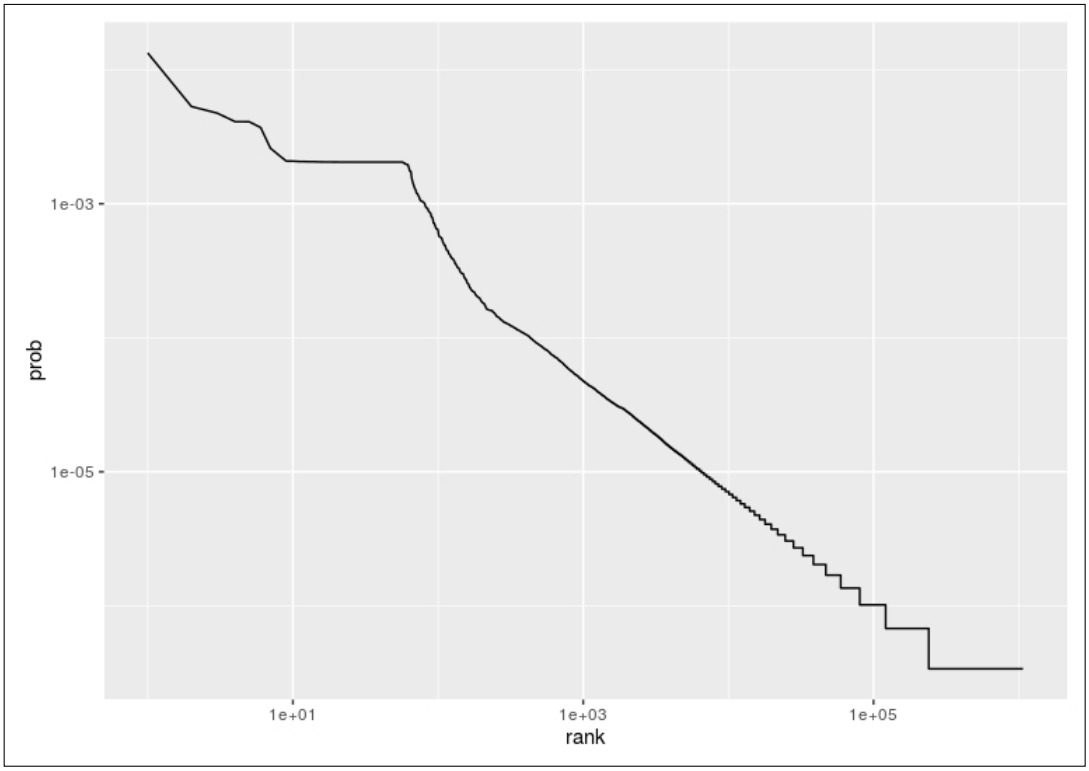

Figure 2: Log-log rank-frequency for bigrams in Wikipedia Collection (Wiki small)

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'. Beside that, 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 3 show the plot for vocabulary growth for the Wikipedia collection.

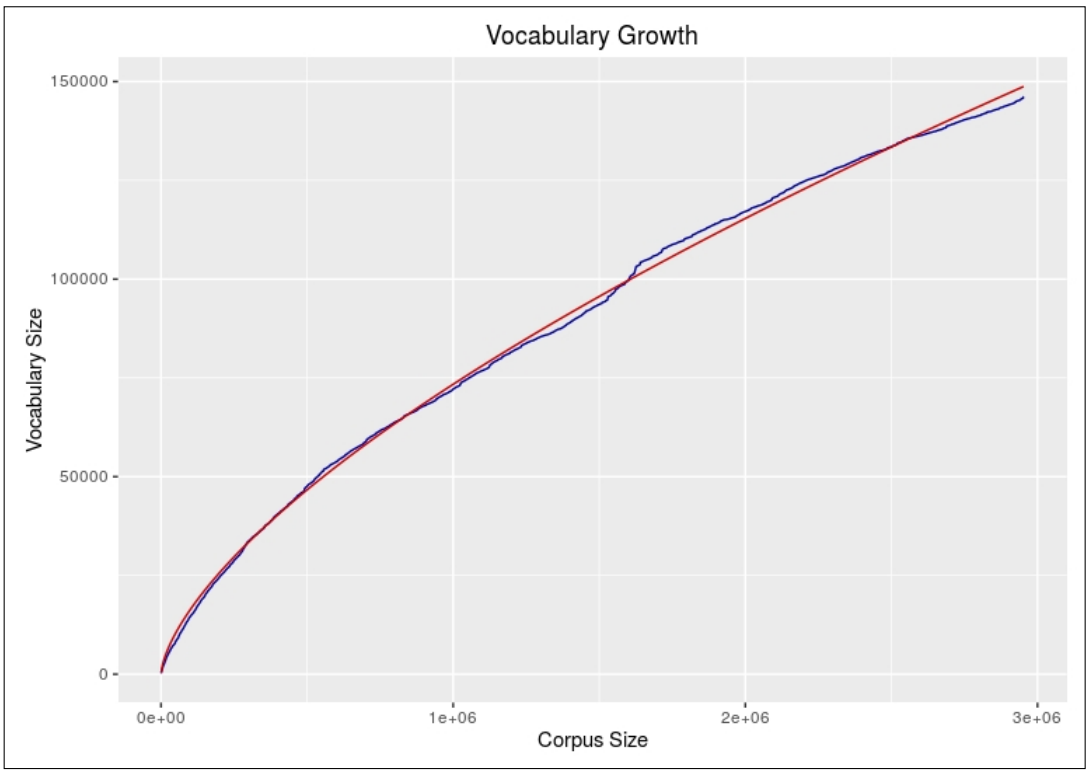

Figure 3: Vocabulary growth for the Wikipedia Collection (Wiki small)

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 4 shows the plot for vocabulary growth for the Wikipedia collection in reversed order.

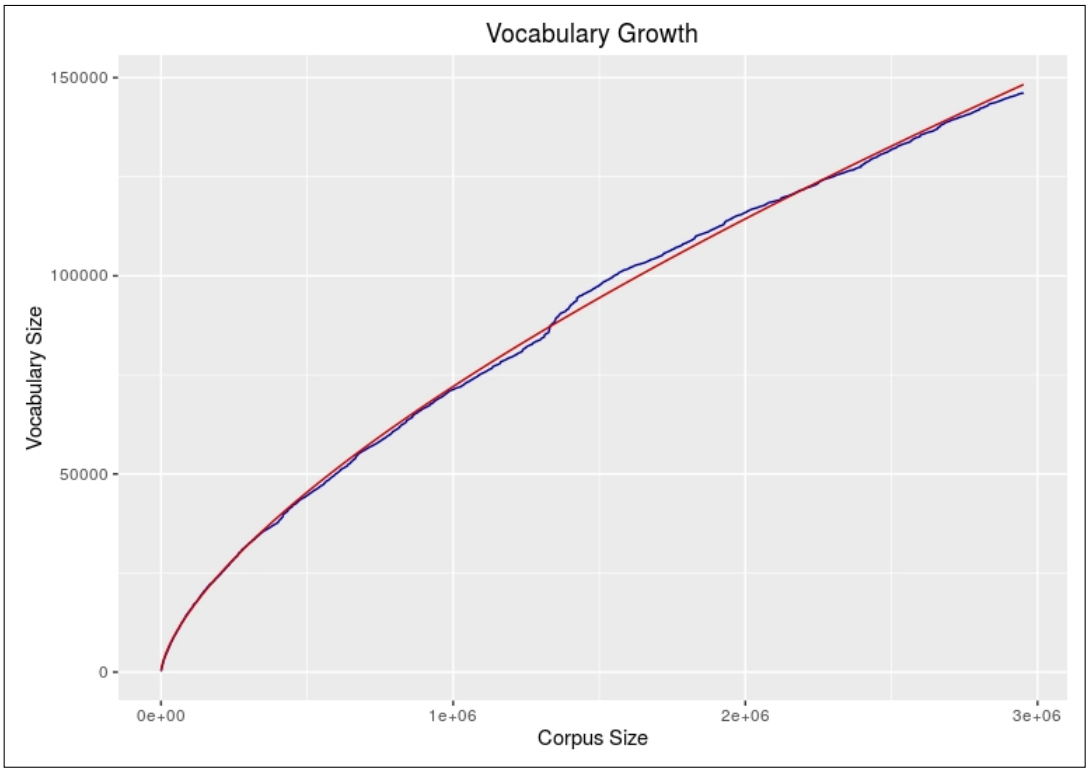

Figure 4: Vocabulary growth for the Wikipedia Collection (Wiki small)

The complete code for calculating the vocabulary growth can be seen in listing 2

```
#!/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)

bigrams = []
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.isalpha():
            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.isalpha():
            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 2: Source code for calculating vocabulary growth

Estimation of the parameters for Heap's law are done using non-linear least square (nls), which is provided in R. Table 3 show the parameter of Heap's law both for ascending and descending order.

No	Parameter	Values
1	k ascending	8.8941676
2	b ascending	0.6527393
3	k descending	7.2333319
4	b descending	0.6663944

Table 3: Heap parameter for Wikipedia Collection

The R code for creating the plot and calculating the parameter value can be seen in listing 3.

```

require(ggplot2)

mydata <- read.csv("/media/erikaris/DATA/ODU/Semester 3/intro_to_info_retrieval/
    assignments/a2/code_report/4_2-voc_corpus.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="#000099") + geom_line(data=mydata, aes(x=corpus_size, y=predict(fit)),
     color="#CC0000") + labs(title='Vocabulary Growth',x = 'Corpus Size', y = '
    Vocabulary Size') + scale_colour_manual(name='', values=c('Important line'='grey'
    , 'Point values'='red'), guide='legend')

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

mydata <- read.csv("/media/erikaris/DATA/ODU/Semester 3/intro_to_info_retrieval/
    assignments/a2/code_report/4_2-voc_corpus_reverse.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="#000099") + geom_line(data=mydata, aes(x=corpus_size, y=predict(fit)),
     color="#CC0000") + labs(title='Vocabulary Growth',x = 'Corpus Size', y = '
```

```
    Vocabulary Size') + scale_colour_manual(name='', values=c('Important line'='grey'
    , 'Point values'='red'), guide='legend')
```

Listing 3: 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

I choose randomly 5 documents from the Wikipedia collections. These 5 documents are:

1. './articles/a/b/c/ABC_Wasp_3b25.html'
2. './articles/a/b/c/ABC_In_Concert_6d5f.html'
3. './articles/a/b/d/Abdus_Salam_(disambiguation)_0602.html'
4. './articles/a/b/d/Abd-Allah_ibn_Amr_f58f.html'
5. './articles//a/b/d/Abdul_Haq_Vidyarthi_582b.html'

Table 4 show 10 differences in the stemming between Porter and Krovetz that could have impact on ranking.

No	Original	Porter	Krovetz
1	united states	unit state	united states
2	available	avail	available
3	engine	engin	engine
4	general	gener	general
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 4: Examples of differences in stemming between Porter and Krovetz

The source code for processing the Porter and Krovetz stemmer can be seen in listing 4.

```
1
2 #!/usr/bin/python
3 import io
4 import sys
5
6 import html2text as html2text
7 import krovetzstemmer
8 from nltk import PorterStemmer
```

```
9
10 # Instantiate porter stemmer
11 porter = PorterStemmer()
12 krovetz = krovetzstemmer.Stemmer()
13
14 if len(sys.argv) < 6:
15     print('Usage :')
16     print('python 4_6.py <file_1> ... <file_5>')
17
18 # Assuming all arguments are file
19 files = []
20 for arg in range(1, len(sys.argv)):
21     files.append(sys.argv[arg])
22
23 # Get contents of each file
24 results = {}
25 for idx, file in enumerate(files):
26     print('{} of {}. Processing {}'.format(idx + 1, len(files), file))
27     print('=' * 30)
28
29     # get text content
30     h = html2text.HTML2Text()
31     h.ignore_links = True
32     text = h.handle(u' '.join([line.strip() for line in io.open(file, "r", encoding="
    utf-8").readlines()]))
33
34     # remove whitespace
35     words = []
36     for word in text.split():
37         if word.isalpha():
38             words.append(word.lower())
39     text = u' '.join(words)
40
41     porter_result = []
42     krovetz_result = []
43     for c in words:
44         porter_result.append(porter.stem(c))
45         krovetz_result.append(krovetz.stem(c))
46
47     results[file] = {}
48     results[file]['original'] = text
49     results[file]['porter'] = u' '.join(porter_result)
50     results[file]['krovetz'] = u' '.join(krovetz_result)
51
52 # print results
53 txt_results = []
54 for file in results:
55     txt_results.append(u'Stemmer result of {}'.format(file))
56     txt_results.append(u'{}'.format('=' * 60))
57     txt_results.append(u'Original text \t= {}\n'.format(results[file]['original']))
58     txt_results.append(u'Porter result \t= {}\n'.format(results[file]['porter']))
59     txt_results.append(u'Krovetz result \t= {}\n'.format(results[file]['krovetz']))
60     txt_results.append(u'\n')
61
62     print(u'\n'.join(txt_results))
63
64     # also write to file
65     f = io.open('4_6-result.txt', "w", encoding="utf-8")
66     for txt_result in txt_results:
```

```
67          f.write(txt_result + '\n')
```

Listing 4: 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 5 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 5: Wikipedia documents with the most inlinks

The complete source code for processing the inlinks can be seen in listing 5.

```
1
2  #!/usr/bin/python
3  import os
4  from pprint import pprint
5
6  import unicodecsv as csv
7  from bs4 import BeautifulSoup
8  from tabulate import tabulate
9
10 html_files = []
11 # 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
        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 5: 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 [5]. 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 6. This code run using terminal input (figure 5).

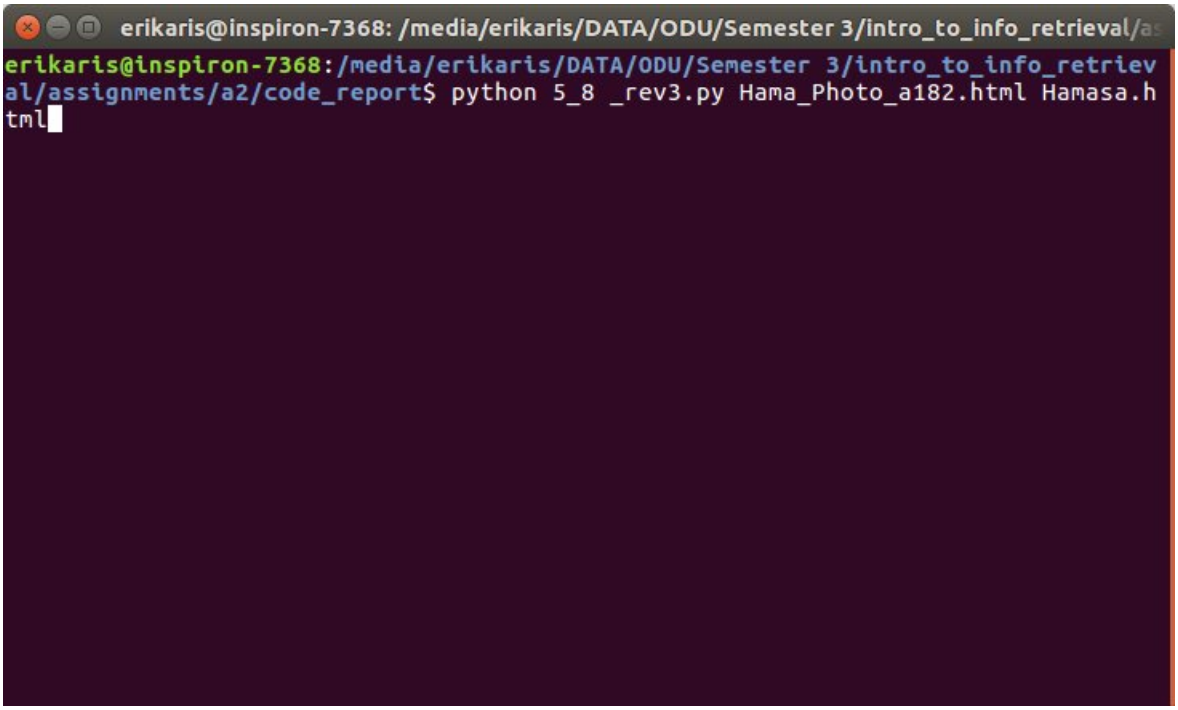

Figure 5: Terminal input for running the inverted index

```
1
2  #!/usr/bin/python
3  import unicodecsv as csv
4  import io
5  import os
6  import sys
7
8  import html2text
9  from tabulate import tabulate
10
11 # Assuming all arguments are file
12 files = []
13 for arg in range(1, len(sys.argv)):
14     files.append(sys.argv[arg])
```

```
15
16 file_words_index = {}
17 all_words = set()
18
19 # Read all files
20 for file in files:
21     # get text content
22     h = html2text.HTML2Text()
23     h.ignore_links = True
24     text = h.handle(u' '.join([line.strip() for line in io.open(file, "r", encoding="
       utf-8").readlines()]))
25
26     words = [word.lower() for word in text.split() if word.isalpha()]
27     all_words |= set(words)
28     file_words_index[file.split(os.pathsep)[-1]] = words
29
30 # Invert words and files
31 word_files_index = {}
32 for word in all_words:
33     files = []
34     for file, words in file_words_index.items():
35         if word in words:
36             files.append(file)
37     word_files_index[word] = sorted(set(files))
38
39 # Convert to 2d array
40 table = []
41 for word in word_files_index:
42     table.append([word, u', '.join(word_files_index[word])])
43
44 print tabulate(table, headers=["word", "files"])
45
46 # write the output to csv file
47 out_file = os.path.join(os.getcwd(), '5_8-inverted_index.csv')
48 with open(out_file, "wb") as f:
49     writer = csv.writer(f)
50     writer.writerow(["word", "files"])
51     writer.writerows(table)
```

Listing 6: Source code for simple inverted index

Table 6 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 6: 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] RosettaCode.org. Inverted index. `https://www.rosettacode.org/wiki/Inverted_index#Python`, 2016. [Online; accessed 12-October-2016].