Assignment 2

CS 734: Introduction to Information Retrieval

Fall 2017

Grant Atkins

Finished on October 15, 2017

1

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 wrote two files of code, **rankFreq.py** and **rankFreq.R** in with the "small" wiki dataset provided from the textbooks website . The first python file iterates through all of the wiki html files, tokenizes them, finds token frequency, and then writes them to a CSV in descending frequency order. To retrieve the text from each of the html files I used Beautifulsoup. It should be noted when retrieving each token from the html files I did not take into account uppercase or lowercase as same terms, I treated them as different and more than probably affected the outcome of this answer. After the code tokenizing the terms it created a list of unigrams and bigrams for all the terms keeping them in separate lists to count frequencies. I used the NLTK python library to make bigram pairs. The code for this is shown below in Listing 1. The top 10, ranked by token frequency, results are shown below in Figures 1 and 2. The full CSV files can be found in my Github repository [3]. Without removing stop words its apparent that words like "the" and "of" would some of the top unigram and bigram pairs.

```
1  #!/usr/bin/env python3
2
3  from bs4 import BeautifulSoup
4  import os
5  import nltk
6  import csv
7
8
9  def unpackFiles():
10     file_list = []
11     for root, dirs, files in os.walk(os.path.dirname("./data/en/
           articles")):
12         for f in files:
13             if f.endswith(".html"):
14                 path = os.path.join(root, f)
15                 file_list.append(path)
```

```
16
17      return file_list
18
19
20  def tokenizeFiles(file_list):
21      tokens = []
22      for i, f in enumerate(file_list):
23          html = open(f, 'r')
24          soup = BeautifulSoup(html.read(), 'html.parser')
25          text = soup.get_text()
26
27          for word in text.split():
28              if word.isalpha():
29                  tokens.append(word)
30
31      return tokens
32
33
34  def tokenCounts(tokens):
35      bigrams = list(nltk.bigrams(tokens))
36      token_counts = {}
37      bigram_counts = {}
38
39      for t in tokens:
40          token_counts.setdefault(t, 0)
41          token_counts[t] += 1
42
43      for t in bigrams:
44          bigram_counts.setdefault(t, 0)
45          bigram_counts[t] += 1
46
47      return token_counts, bigram_counts, bigrams
48
49
50  def calcProbC(token_list, all_tokens):
51      new_list = []
52      for i, row in enumerate(token_list):
53          # prob = token count / all tokens
54          prob = float(row[1]) / len(all_tokens)
55          # c = pos of freq in list * prob
56          c = (i + 1) * prob
57          new_list.append(row + [prob, c])
58
59      return new_list
60
61
62  def write_csv(filename, token_type, tokens):
63      with open("./data/" + filename, 'w') as f:
64          writer = csv.writer(f)
```

```
65          writer.writerow([token_type, "frequency", "prob", "c"])
66          writer.writerows(tokens)
67
68
69  def convertDimensions(token_counts):
70      '''Make 2D format to write to csv'''
71      d = []
72
73      for t in token_counts:
74          d.append([t, token_counts[t]])
75
76      d = sorted(d, key=lambda x: x[1], reverse=True)
77      return d
78
79
80  if __name__ == "__main__":
81      # get all html files
82      file_list = unpackFiles()
83      # get list of all tokens
84      tokens = tokenizeFiles(file_list)
85      # count tokens. returns unigram, bigram dictionaries, bigram
            entire list
86      tc, bc, bigrams = tokenCounts(tokens)
87      # convert to sorted list based on frequeny
88      t1 = convertDimensions(tc)
89      t2 = convertDimensions(bc)
90      # add calculations to each token(s)
91      t1 = calcProbC(t1, tokens)
92      t2 = calcProbC(t2, bigrams)
93
94      write_csv("rankFreqUnigram.csv", "unigram", t1)
95      write_csv("rankFreqBigram.csv", "bigram", t2)
```

Listing 1: Python script to tokenize and find frequencies and calculate C parameters

To create the graphs I used R's ggplot2 library. The code to create these graphs is shown in Listing 2. The figures created from the afore mentioned code are shown in Figure 3, 4, and 5. For unigrams the best C parameter was 0.14, while for bigrams it was 0.1.

```
1  require(ggplot2)
2
3  unigramFreq <- read.csv("./data/rankFreqUnigram.csv", head =
       TRUE, sep = ',')
4  bigramFreq <- read.csv("./data/rankFreqBigram.csv", head = TRUE,
       sep = ',')
5
6  # unigrams
```

```
7  unigramFreq$rownum <- as.numeric(row.names(unigramFreq))
8
9  ggplot(data=unigramFreq, aes(x=rownum, y=prob)) +
10   geom_point() +
11   scale_x_log10() +
12   scale_y_log10() +
13   labs(x = "Rank", y = "Probability")
14
15 # bigrams
16 bigramFreq$rownum <- as.numeric(row.names(bigramFreq))
17
18 ggplot(data=bigramFreq, aes(x=rownum, y=prob)) +
19   geom_point() +
20   scale_x_log10() +
21   scale_y_log10() +
22   labs(x = "Rank", y = "Probability")
23
24 # merged graphs
25
26 ggplot(data=unigramFreq, aes(x=rownum, y=prob)) +
27   geom_line(data=unigramFreq, aes(x=rownum, y=prob, color="
         Frequency")) +
28   geom_line(data=bigramFreq, aes(x=rownum, y=prob, color="Bigram
         ")) +
29   scale_colour_manual(name='',
30                       values=c('Frequency'='#5EA036', 'Bigram
                             '='#2B56CA'),
31                       guide='legend') +
32   scale_x_log10() +
33   scale_y_log10() +
34   labs(title = "Log-log plot of word frequency and bigrams",
35        x = "Words",
36        y = "Probability")
```

Listing 2: Python script to tokenize and find frequencies and calculate C parameters

	bigram	frequency	prob	c
1	('of', 'the')	38083	0.0124538617	0.01245386
2	('in', 'the')	15578	0.0050943008	0.01018860
3	('is', 'a')	14019	0.0045844783	0.01375343
4	('the', 'free')	12148	0.0039726259	0.01589050
5	('a', 'registered')	12098	0.0039562750	0.01978137
6	('free', 'encyclopedia')	12088	0.0039530048	0.02371803
7	('About', 'Wikipedia')	12086	0.0039523507	0.02766646
8	('by', 'Wikipedia')	10932	0.0035749709	0.02859977
9	('to', 'the')	7653	0.0025026758	0.02252408
10	('under', 'the')	6804	0.0022250368	0.02225037

Figure 1: Top 10 unigrams found

	unigram	frequency	prob	c
1	the	168911	0.0552370756	0.05523708
2	of	111499	0.0364622712	0.07292454
3	and	77222	0.0252530472	0.07575914
4	a	61567	0.0201335676	0.08053427
5	in	58112	0.0190037175	0.09501859
6	to	53513	0.0174997580	0.10499855
7	is	40919	0.0133812830	0.09366898
8	Wikipedia	38128	0.0124685735	0.09974859
9	by	33542	0.0109688652	0.09871979
10	The	29485	0.0096421498	0.09642150

Figure 2: Top 10 bigrams found

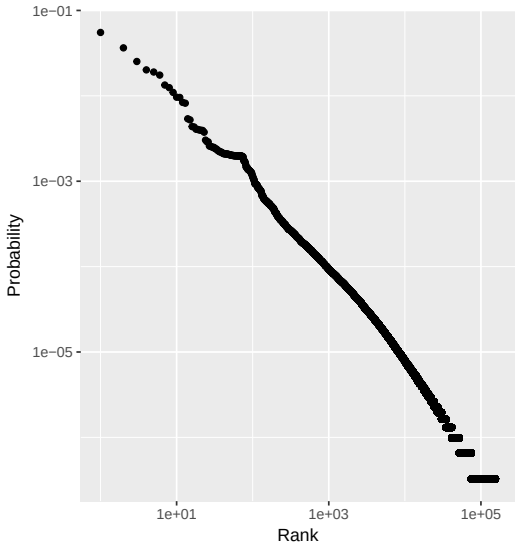

Figure 3: Log-log plot of unigram frequency and probability

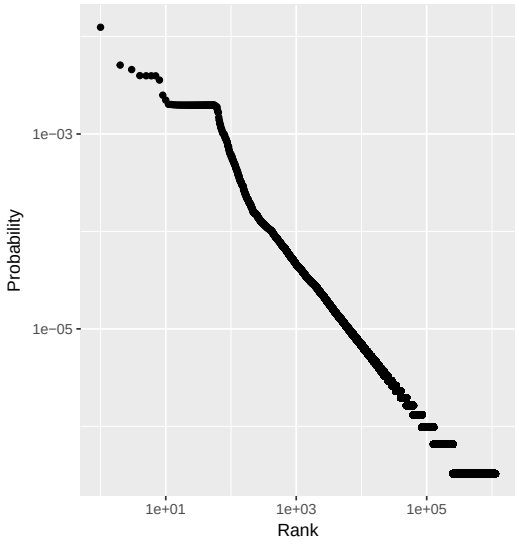

Figure 4: Log-log plot of bigram frequency and probability

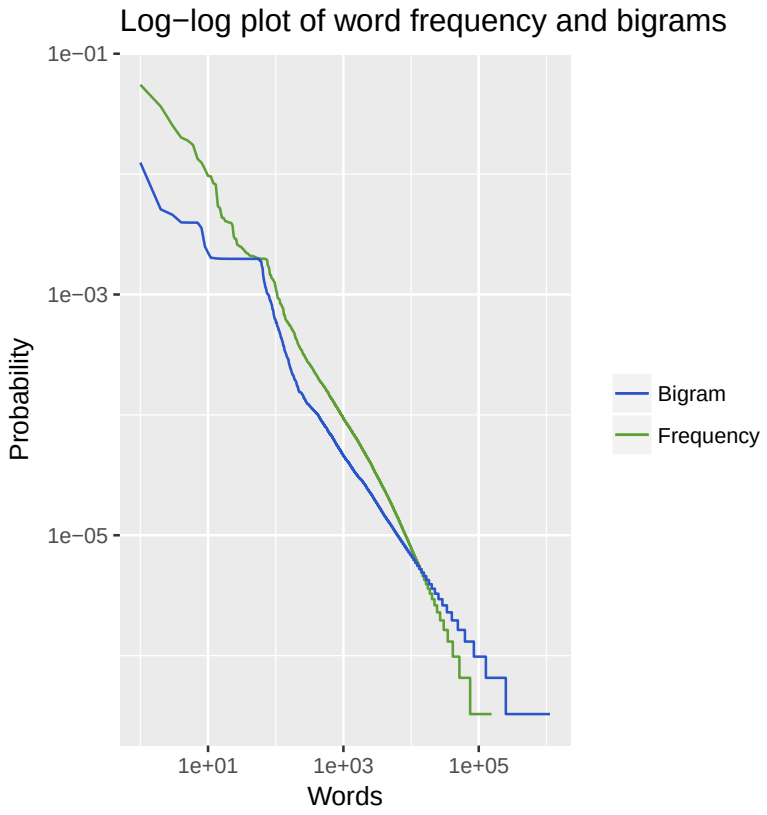

Figure 5: Log-log plot of unigram and bigram frequencies and probabilities

2

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

To answer this question I again created two files, **vocabGrowth.py** and **vocabGrowth.R**. My python script use the Beautifulsoup library to parse out the html documents of the small wikipedia example. Much like my answer to the first question I traversed the html documents in order and then in reverse, each time tokenizing the text of html, taking an overall corpus count, tracking unique vocabulary through each of the html documents, and then saving all of these values to csv file denoting document iteration, corpus size, and vocab size when at that document. The code for this is shown in Listing 3. The goal of this was simply see how the unique vocabulary list grows while the corpus size continues to grow.

```
1  #!/usr/bin/env python3
2
3  from bs4 import BeautifulSoup
4  import os
5  import csv
6
7
8  def unpackFiles():
9      file_list = []
10     for root, dirs, files in os.walk(os.path.dirname("./data/en/
           articles")):
11         for f in files:
12             if f.endswith(".html"):
13                 path = os.path.join(root, f)
14                 file_list.append(path)
15
16     return file_list
17
18
19 def getCorpus(file_list):
20     # track corpus and vocab growths
21     vocab_corpus_counts = []
22     # all tokens found
23     corpus_count = 0
24     vocab = set()
```

```
25
26      for i, f in enumerate(file_list):
27          tokens = []
28          html = open(f, 'r')
29          soup = BeautifulSoup(html.read(), 'html.parser')
30          text = soup.get_text()
31
32          for word in text.split():
33              if word.isalpha():
34                  tokens.append(word)
35                  if word not in vocab:
36                      vocab.add(word)
37
38          corpus_count += len(tokens)
39
40          vocab_corpus_counts.append([(i + 1), corpus_count, len(
                vocab)])
41
42      return vocab_corpus_counts
43
44
45  def write_csv(filename, corpus_growth):
46      with open("./data/" + filename, 'w') as f:
47          writer = csv.writer(f)
48          writer.writerow(["doc_number", "corpus_size", "
                vocab_count"])
49          writer.writerows(corpus_growth)
50
51
52  if __name__ == "__main__":
53      # get all html files
54      file_list = unpackFiles()
55      # get list of all tokens
56      corpus_growth = getCorpus(file_list)
57      write_csv("corpusGrowth.csv", corpus_growth)
58
59      # now traverse documents in reverse order
60      file_list.reverse()
61      corpus_growth = getCorpus(file_list)
62      write_csv("corpusGrowthReverse.csv", corpus_growth)
```

Listing 3: Python script to track vocabulary growth

The relationship between corpus size and vocabulary size was defined in our book as:

$$v = k * n^{\beta}$$

To estimate the parameters for heaps law and plot the vocabulary growth, I used R's ggplot2 to create charts and the built in function non-linear

least squares (nls) as shown in Figure 4. The parameters of K and B were initialized with a value of one. The plot of documents traversed in order is shown in Figure 6 and the plot of the documents traversed in reverse is shown in Figure 7. The plots show vocab count along the y-axis and corpus word count along x-axis. It is apparent that these two graphs are different and that means the order in which the documents are processed does make a difference.

The estimation parameters for Heap's law for in order and reverse, computed by the nls method in R, are shown in Table 1. Some of values do differ slightly, such as the B values for ascending and descending, but there is an apparent difference which further supporting my claim.

```
require(ggplot2)

corpusGrowth <- read.csv("./data/corpusGrowth.csv", head = TRUE,
    sep = ',')
corpusGrowthRev <- read.csv("./data/corpusGrowthReverse.csv",
    head = TRUE, sep = ',')

# vocab growth function assuming data formatted appropriately
vocab_growth <- function(df){
  x <- df$corpus_size
  y <- df$vocab_count

  # fit to non-linear least squares model
  fit <- nls(y~k*(x^b), data = df, start = list(k=1,b=1))
  print(summary(fit))
  cor(y, predict(fit))

  p <- ggplot(data=df, aes(x=corpus_size, y=vocab_count)) +
    geom_line(aes(group = 1, color="Actual")) +
    geom_line(data=corpusGrowth, aes(x=corpus_size, y=predict(
        fit), color="Heaps")) +
    scale_colour_manual(name='', values=c('Actual'='#5EA036', '
        Heaps'='#2B56CA'), guide='legend') +
    labs(title='',x = 'Word Count', y = 'Vocab Count')

  print(p)
}

# execute on both data frames
vocab_growth(corpusGrowth)
vocab_growth(corpusGrowthRev)
```

Listing 4: R script to compute Heap's Law

Number	Parameter	Values
1	Document Count	6043
2	K ascending	7.6679986
3	B ascending	0.6652129
4	K descending	6.276427
5	B descending	0.678342

Table 1: Heap's Law parameters for Small Wiki

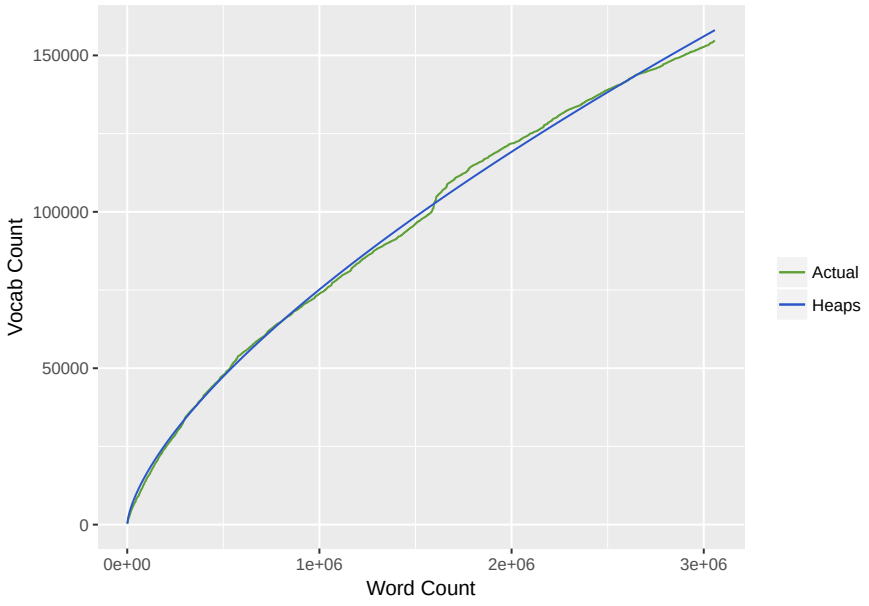

Figure 6: Growth of vocabulary vs overall corpus

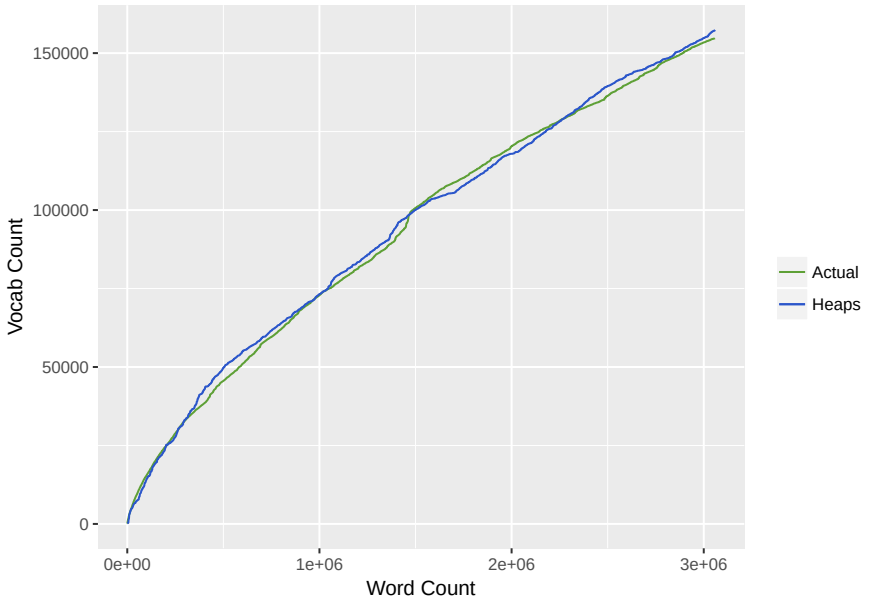

Figure 7: Growth of vocabulary vs overall corpus with documents traversed in reverse

3

Question

4.3. Try to estimate the number of web pages indexed by two different search engines using the technique described in this chapter. Compare the size estimates from a range of queries and discuss the consistency (or lack of it) of these estimates.

Answer

The technique described in the textbook [4] is described as follows:

$$f_{ab} = N \cdot f_a/N \cdot f_b/N = (f_a \cdot f_b)/N \tag{1}$$

Where:
N is the number of documents in the collection.
f_i is the number of document that term i occurs.
f_{ab} is the combined set result.

To estimate N we can use the following equation:

$$N = \frac{(f_a \cdot f_b)}{f_{ab}} \tag{2}$$

The search engines I decided to use in this problem are google and bing. My query term for both search engines was "ocean grove." For Google, my variables f_a, f_b, and f_{ab} are described as follows:

$$f_{ocean} = 1,240,000,000$$

$$f_{grove} = 379,000,000$$

$$f_{ocean\ grove} = 7,900,000$$

These are shown in Figures 8, 9, and 10 respectively. To estimate N we use the following formula:

$$N = \frac{(1,240,000,000 \cdot 379,000,000)}{7,900,000}$$

$$N = 59.5\ billion$$

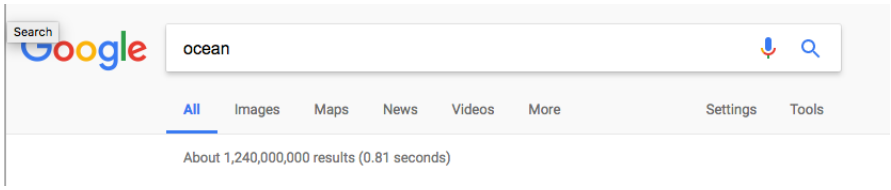

Figure 8: Google ocean query

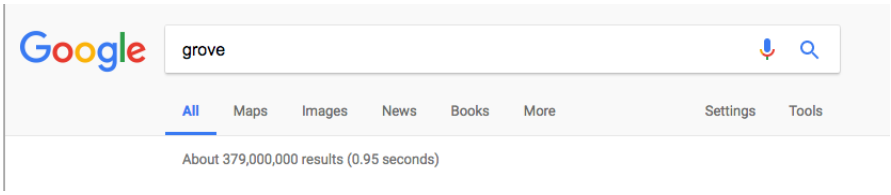

Figure 9: Google grove query

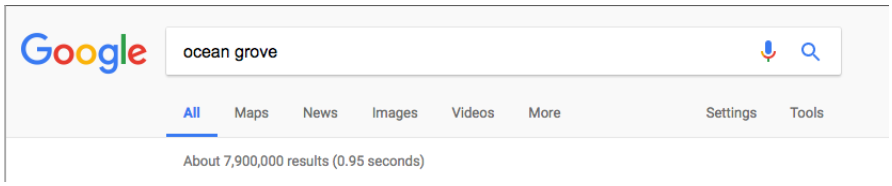

Figure 10: Google ocean grove query

Comparing this estimated index value to what is shown on the `http://www.worldwidewebsize.com/`, as shown in Figure 11, its an approximately 59.5/47.2 or 1.26:1 ratio.

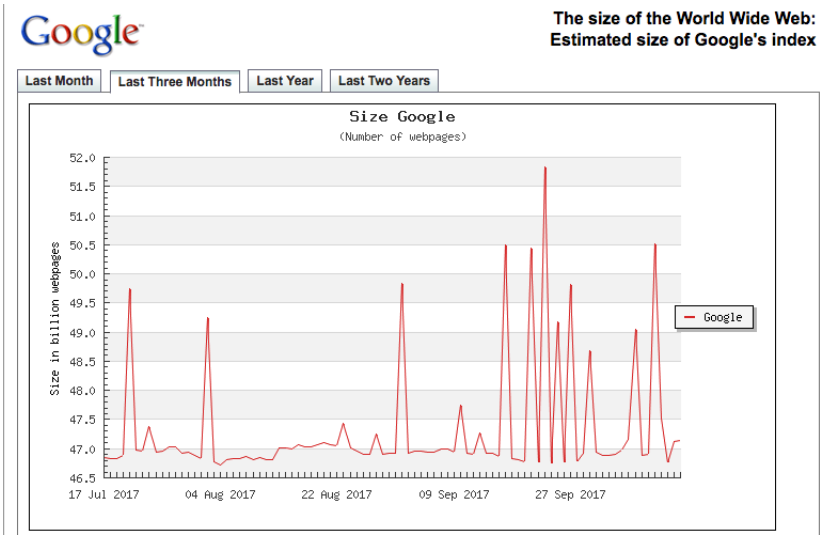

Figure 11: Google's webpage size

For Bing, my variables f_a, f_b, and f_{ab} are described as follows:

$$f_{ocean} = 183,000,000$$

$$f_{grove} = 90,900,000$$

$$f_{ocean\ grove} = 296,000$$

These are shown in Figures 12, 13, and 14 respectively. To estimate N we use the following formula:

$$N = \frac{(183,000,000 \cdot 90,900,000)}{296,000}$$

$$N = 56.2\ billion$$

Comparing this estimated index value to what is shown on the `http://www.worldwidewebsize.com/`, as shown in Figure 15, its an approximately 56.2/6 or 9.37:1 ratio. This is a huge difference compared to google. It

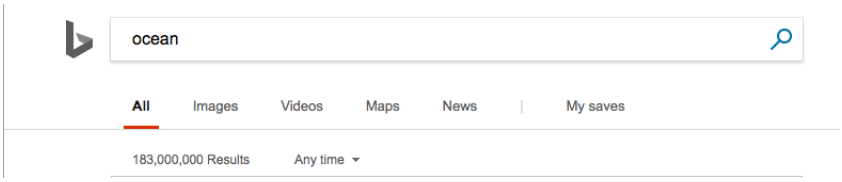

Figure 12: Bing ocean query

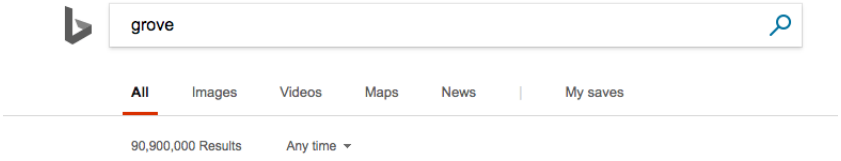

Figure 13: Bing grove query

clearly shows that Bing isn't putting as much resources into their search engine as google is.

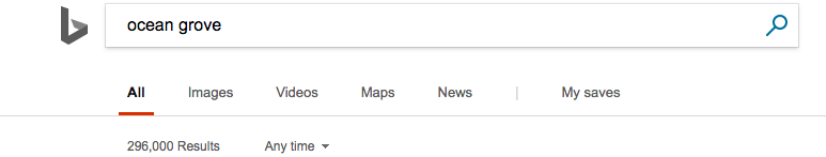

Figure 14: Bing ocean grove query

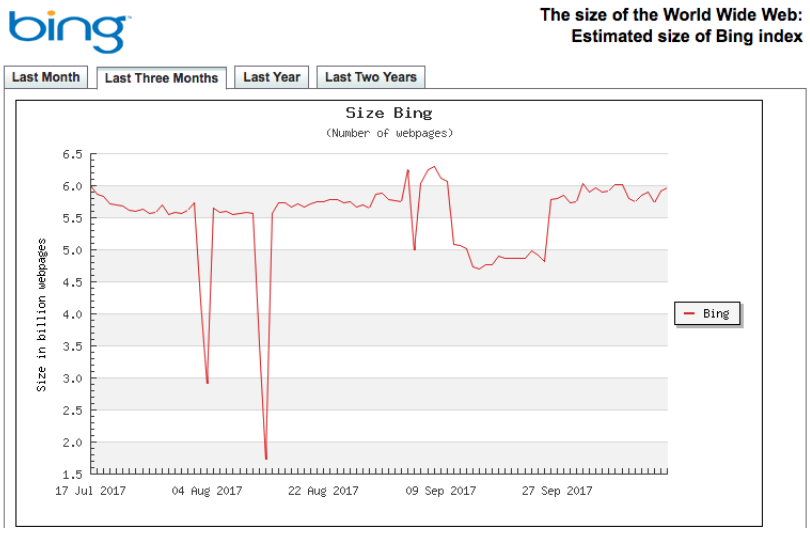

Figure 15: Bing's webpage size

4

Question

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

Answer

To answer this question I wrote a script named **inlinks.py** as shown in Listing 6. This script once again takes advantage of Beautifulsoup to parse html, but this time I also used it to extract all the "a" element tags with href tags. For each of the URIs found I would create a dictionary with their destination and their anchor text to the html element. The results of this is shown below in Table 2. This data is also present on my github page called **inlinks.csv** in the data directory [3]. You should probably go to the github, because I can't format my tables properly.

Link	Inlink Count	Anchor Text
Brazil.html	124	BRA, Brazil, Article, Brazilian
Foreign_hostages_in_Iraq_9250.html	118	edit, Article
List_of_birds_of_the_United_Arab_Emirates_67ef.html	66	edit, Article
2007_New_York_Jets_season_b663.html	64	article, edit, 2007
German_railway_signalling.html	53	edit, Article
August_26.html	50	08-26, 26 August 26, August 26, edit, Arti
IPv6_fdb7.html	50	edit, Article
Symmetry.html	45	edit, Article
List_of_important_publications_in_geology.html	44	edit, Article
List_of_units_using_the_B-26_Marauder_during_World_War_II_9296.html	41	edit, Article

Table 2: Heap's Law parameters for Small Wiki

```
#!/usr/bin/env python3

from bs4 import BeautifulSoup
import os
import csv


def unpackFiles():
    file_list = []
    for root, dirs, files in os.walk(os.path.dirname("./data/en/
        articles")):
        for f in files:
            if f.endswith(".html"):
                path = os.path.join(root, f)
                file_list.append(path)

    return file_list


def findLinks(file_list):
    links = {}
    atexts = {}

    for i, f in enumerate(file_list):
        html = open(f, 'r')
        soup = BeautifulSoup(html.read(), 'html.parser')

```

```
27          for a in soup.find_all("a", href=True):
28              link = a["href"]
29              atext = a.string
30
31              if atext is None:
32                  atext = ""
33
34              # assumption every outside URI has http
35              if not link.startswith('http'):
36                  try:
37                      link = os.path.join(os.path.dirname(f), link
                            )
38                      link = os.path.abspath(link)
39                  except:
40                      pass
41
42              links.setdefault(f, [])
43              atexts.setdefault(f, [])
44
45              if f != link and link not in atexts[f] and os.path.
                    isfile(link):
46                  links[f].append(link)
47                  atexts[f].append(atext)
48
49      return links, atexts
50
51
52  def findDestinations(links, atexts):
53      inlink_count = {}
54      inlink_text = {}
55      for l in links:
56          outlinks = links[l]
57          atext = atexts[l]
58
59          for i, outlink in enumerate(outlinks):
60              inlink_count.setdefault(outlink, 0)
61              inlink_count[outlink] += 1
62
63              inlink_text.setdefault(outlink, [])
64              inlink_text[outlink].append(atext[i])
65
66      d = []
67      for l in inlink_count:
68          # only get file name, remove path
69          filename = l.rsplit("/", 1)[1]
70          d.append([filename, inlink_count[l], set(inlink_text[l])
                ])
71
72      d = sorted(d, key=lambda x: x[1], reverse=True)
```

```
73
74      return d
75
76
77  def write_csv(filename, inlinks):
78      with open("./data/" + filename, 'w') as f:
79          writer = csv.writer(f)
80          writer.writerow(["link", "inlinks", "anchor_text"])
81          writer.writerows(inlinks)
82
83
84  if __name__ == "__main__":
85      # get all html files
86      file_list = unpackFiles()
87      # get list of all tokens
88      links, atexts = findLinks(file_list)
89      inlinks = findDestinations(links, atexts)
90
91      try:
92          inlinks = inlinks[:10]
93      except:
94          print("ERROR: Not enough inlinks")
95          exit()
96
97      write_csv("inlinks.csv", inlinks)
```

Listing 5: Python script to find top 10 inlinks in small Wiki dataset

5

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

Having done something very similar before for extra in CS532 [2], I decided to lookup some old code I found on a website called Rosetta Code [1]. This code basically is a simple way to create an inverted index. You first tokenize each html document you call on the command line and build a dictionary of files whose attributes are unique terms and tracking all the unique terms. You can then iterate through the unique terms and map each file that has the term in the term's dictionary. In essence this is an inverted index. It should be noted that instead of writing to the command prompt the answers, I decided to write it to a CSV file because many documents created many terms. This CSV can be found in my repository in the data directory [3]. An example run can be shown in Figure 16 below, first running the program, then print the top 10 lines of the CSV file.

```
1 #!/usr/bin/env python3
2
3 from bs4 import BeautifulSoup
4 import os
5 import csv
6 import sys
7
8
```

```
Grants-MacBook-Pro:src gatkins$ python3 invertedIndex.py ./data/en/articles/2/0/0/2006_Purdue_Boilermakers_f
ootball_team_41f1.html ./data/en/articles/2/0/0/2007-08_Georgetown_Hoyas_men\'s_basketball_team_a8f4.html &&
 head -n 10 ./data/invertedIndex.csv
word,documents
also,2006_Purdue_Boilermakers_football_team_41f1.html
players,2006_Purdue_Boilermakers_football_team_41f1.html
wikipedia,"2006_Purdue_Boilermakers_football_team_41f1.html, 2007-08_Georgetown_Hoyas_men's_basketball_team_
a8f4.html"
another,2006_Purdue_Boilermakers_football_team_41f1.html
ncaa,"2006_Purdue_Boilermakers_football_team_41f1.html, 2007-08_Georgetown_Hoyas_men's_basketball_team_a8f4.
html"
mcgowen,2006_Purdue_Boilermakers_football_team_41f1.html
powell,2006_Purdue_Boilermakers_football_team_41f1.html
took,2006_Purdue_Boilermakers_football_team_41f1.html
eugene,2006_Purdue_Boilermakers_football_team_41f1.html
Grants-MacBook-Pro:src gatkins$
```

Figure 16: Inverted Index sample run with CSV top 10 items

```
9  def buildIndex(file_list):
10     index = {}
11     full_vocab = set()
12     for i, f in enumerate(file_list):
13         html = open(f, 'r')
14         soup = BeautifulSoup(html.read(), 'html.parser')
15         text = soup.get_text()
16         vocab = set()
17
18         for word in text.split():
19             if word.isalpha():
20                 word = word.lower()
21                 if word not in vocab:
22                     vocab.add(word)
23
24         full_vocab = full_vocab.union(vocab)
25         fname = f
26         # if no path separator leave as is
27         try:
28             fname = f.rsplit("/", 1)[1]
29         except:
30             fname = f
31         # assign words set of words to filename
32         index[fname] = vocab
33
34     return index, full_vocab
35
36
37 def invertIndex(index, full_vocab):
38     newIndex = {}
39     for w in full_vocab:
40         files = []
```

```
41          for f, words in index.items():
42              if w in words:
43                  files.append(f)
44
45          # pick out unique files
46          newIndex[w] = sorted(set(files))
47
48      d = []
49      for w in newIndex:
50          d.append([w, u', '.join(newIndex[w])])
51
52      return d
53
54
55  def write_csv(filename, index):
56      with open("./data/" + filename, 'w') as f:
57          writer = csv.writer(f)
58          writer.writerow(["word", "documents"])
59          writer.writerows(index)
60
61
62  if __name__ == "__main__":
63      # testing with wiki data
64      # python3 invertedIndex.py ./data/en/articles/2/0/0/2006
            _Purdue_Boilermakers_football_team_41f1.html ./data/en/
            articles/2/0/0/2007-08_Georgetown_Hoyas_men\'
            s_basketball_team_a8f4.html
65      filenames = []
66      for arg in range(1, len(sys.argv)):
67          filenames.append(sys.argv[arg])
68
69      # get all html files
70      index, full_vocab = buildIndex(filenames)
71      invertedIndex = invertIndex(index, full_vocab)
72
73      write_csv("invertedIndex.csv", invertedIndex)
```

Listing 6: Python script to build inverted index from list of files

References

[1] "Inverted Index." Roseta Code Inverted index. N.p.,n.d. Web. 22 Feb. 2017. http://rosettacode.org/wiki/Inverted_index#Simple_inverted_index

[2] Atkins, Grant. "CS532 Assignment 3 Repository" Github. N.p., 23 March 2017. Web. 23 March 2017.https://github.com/grantat/cs532-s17/tree/master/assignments/A3/src.

[3] Atkins, Grant. "CS734 Assignment 2 Repository" Github. N.p., 21 September 2017. Web. 14 October 2017.https://github.com/grantat/cs834-f17/tree/master/assignments/A2.

[4] B. Croft, D. Metzler, and T. Strohman. "Search Engines: Information Retrieval in Practice." Pearson, 2009. Web. 14 October 2017. ISBN 9780136072249.