

Assignment 2

CS 734: Introduction to Information Retrieval

Fall 2017

Grant Atkins

Finished on October 14, 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 [2]. 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
86     # entire list
87     tc, bc, bigrams = tokenCounts(tokens)
88     # convert to sorted list based on frequency
89     t1 = convertDimensions(tc)
90     t2 = convertDimensions(bc)
91     # add calculations to each token(s)
92     t1 = calcProbC(t1, tokens)
93     t2 = calcProbC(t2, bigrams)
94
95     write_csv("rankFreqUnigram.csv", "unigram", t1)
96     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 =
4   TRUE, sep = ',')
5 bigramFreq <- read.csv("../data/rankFreqBigram.csv", head = TRUE,
6   sep = ',')
7
8 # 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

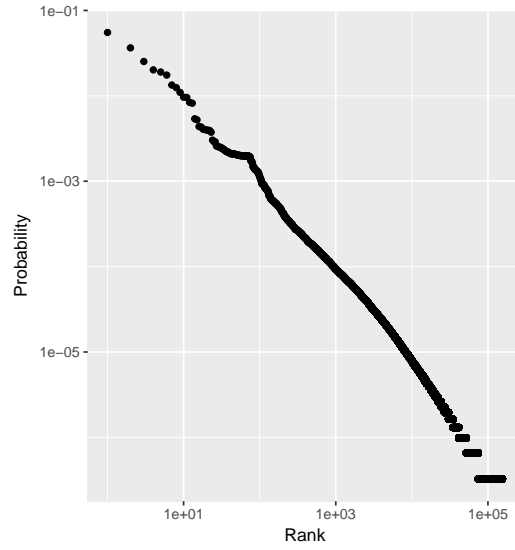


Figure 3: Log-log plot of unigram frequency and probability

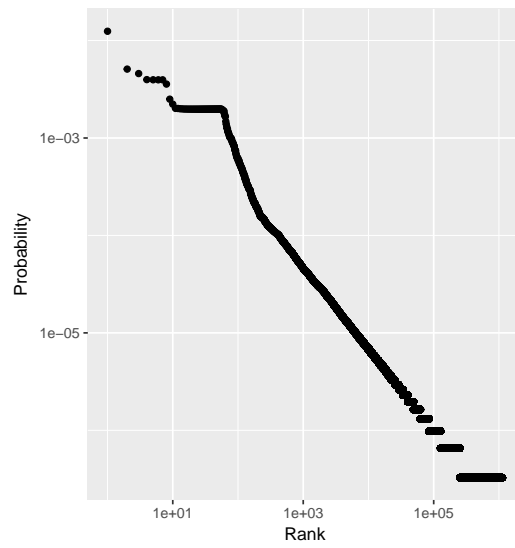


Figure 4: Log-log plot of bigram frequency and probability

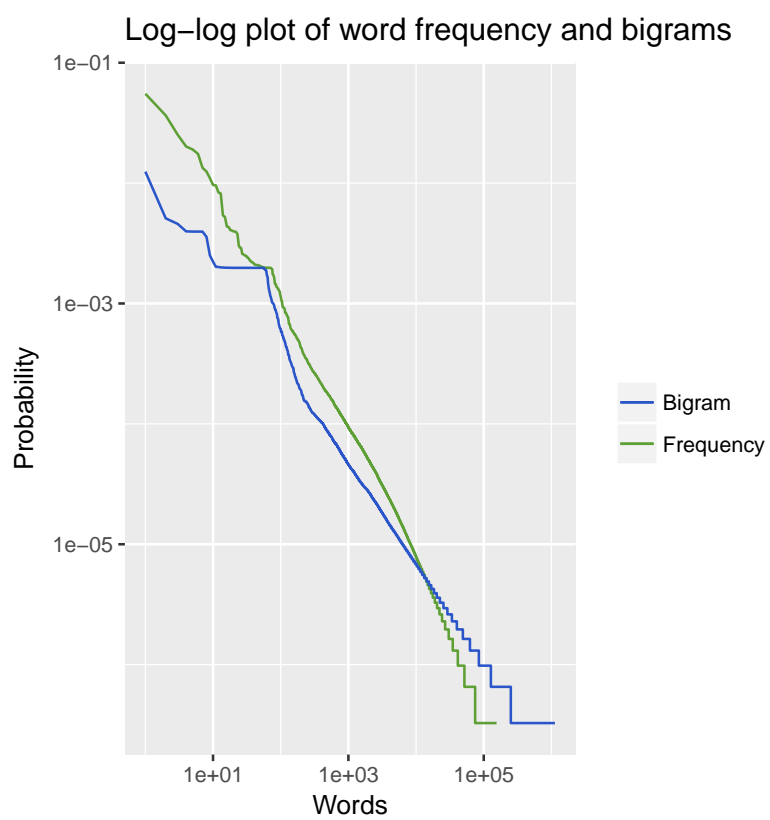


Figure 5: Log-log plot of unigram and bigram frequencies and probabilities

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(
41             vocab)])
42
43     return vocab_corpus_counts
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", "
49             vocab_count"])
50         writer.writerows(corpus_growth)
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.

```

1 require(ggplot2)
2
3 corpusGrowth <- read.csv("./data/corpusGrowth.csv", head = TRUE,
4   sep = ',')
5 corpusGrowthRev <- read.csv("./data/corpusGrowthReverse.csv",
6   head = TRUE, sep = ',')
7
8 # vocab growth function assuming data formatted appropriately
9 vocab_growth <- function(df){
10   x <- df$corpus_size
11   y <- df$vocab_count
12
13   # fit to non-linear least squares model
14   fit <- nls(y~k*(x^b), data = df, start = list(k=1,b=1))
15   print(summary(fit))
16   cor(y, predict(fit))
17
18   p <- ggplot(data=df, aes(x=corpus_size, y=vocab_count)) +
19     geom_line(aes(group = 1, color="Actual")) +
20     geom_line(data=corpusGrowth, aes(x=corpus_size, y=predict(
21       fit), color="Heaps")) +
22     scale_colour_manual(name='', values=c('Actual'='#5EA036', '
23       Heaps'='#2B56CA'), guide='legend') +
24     labs(title='', x = 'Word Count', y = 'Vocab Count')
25
26   print(p)
27 }
28
29 # execute on both data frames
30 vocab_growth(corpusGrowth)
31 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

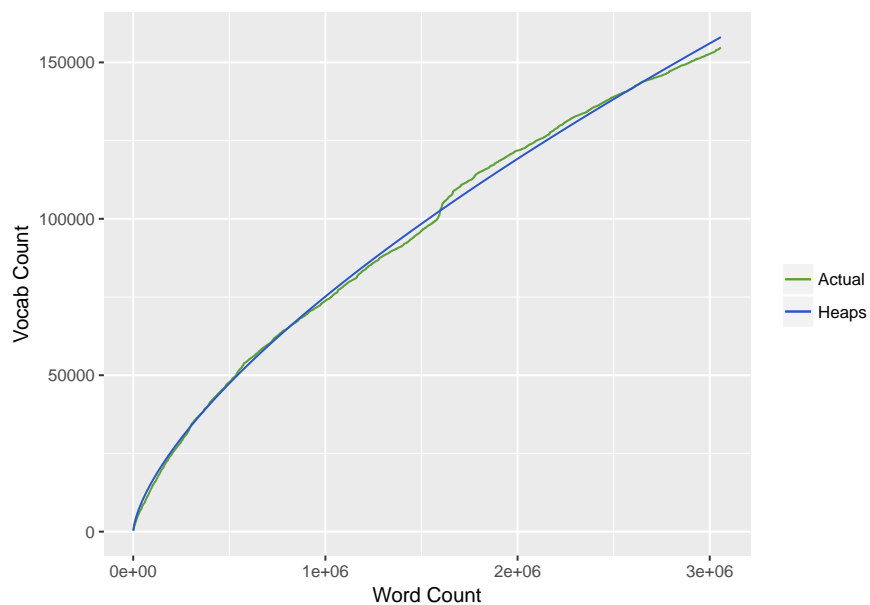


Figure 6: Growth of vocabulary vs overall corpus

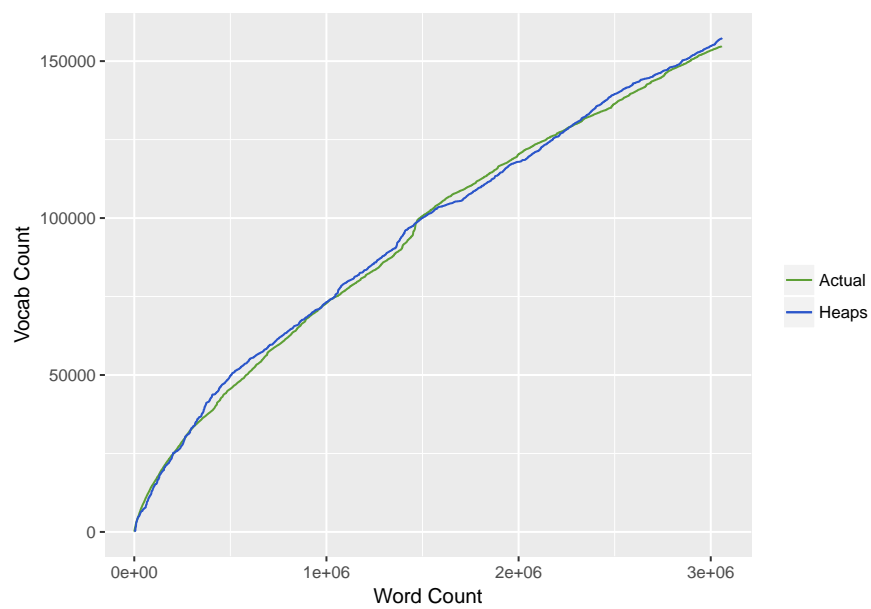


Figure 7: Growth of vocabulary vs overall corpus with documents traversed in reverse

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 [] is described as follows:

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

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:

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

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:

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

$$N = 59.5\ billion$$

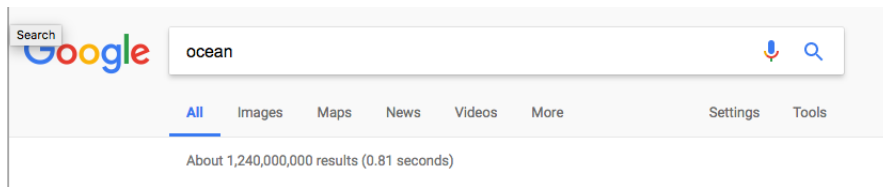


Figure 8: Google ocean query

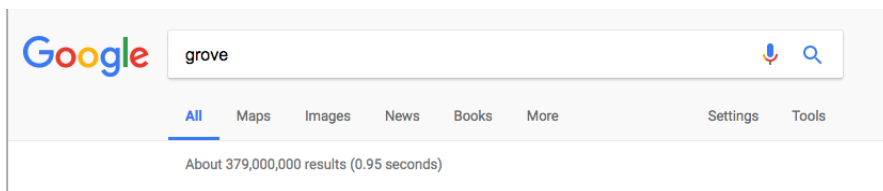


Figure 9: Google grove query

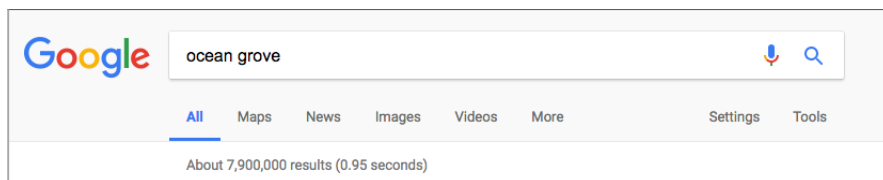


Figure 10: Google ocean grove query

Comparing this estimation 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.

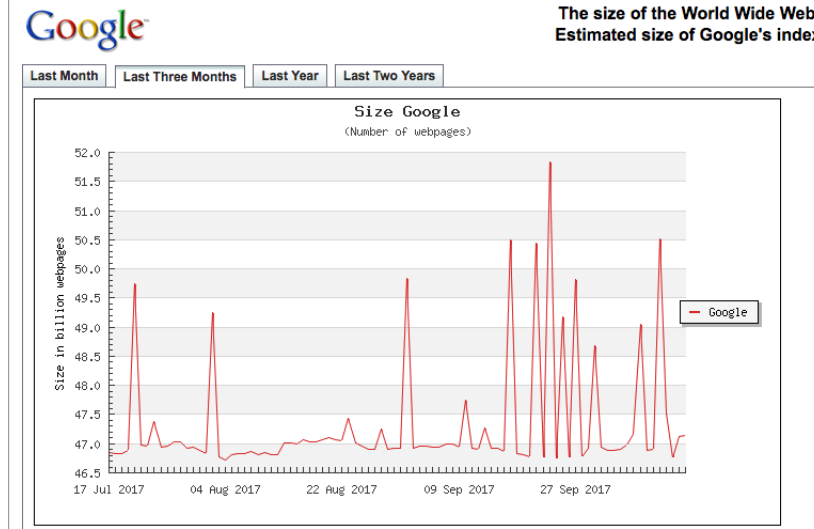


Figure 11: Google's webpage size

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 estimation 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 clearly shows

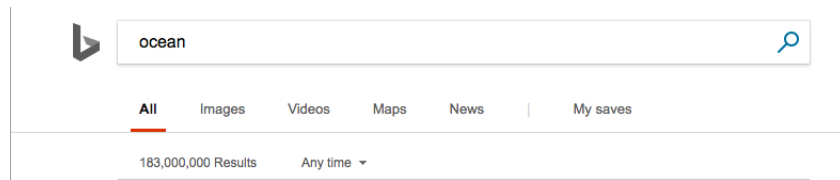


Figure 12: Bing ocean query

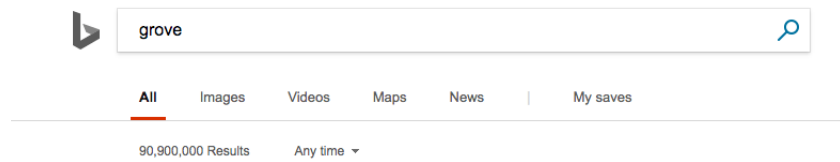


Figure 13: Bing grove query

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

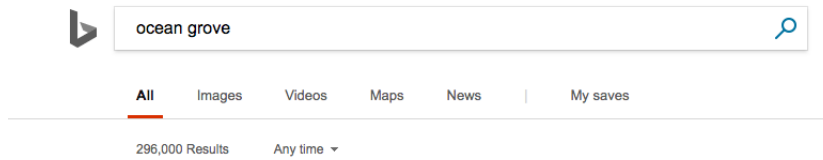


Figure 14: Bing ocean grove query

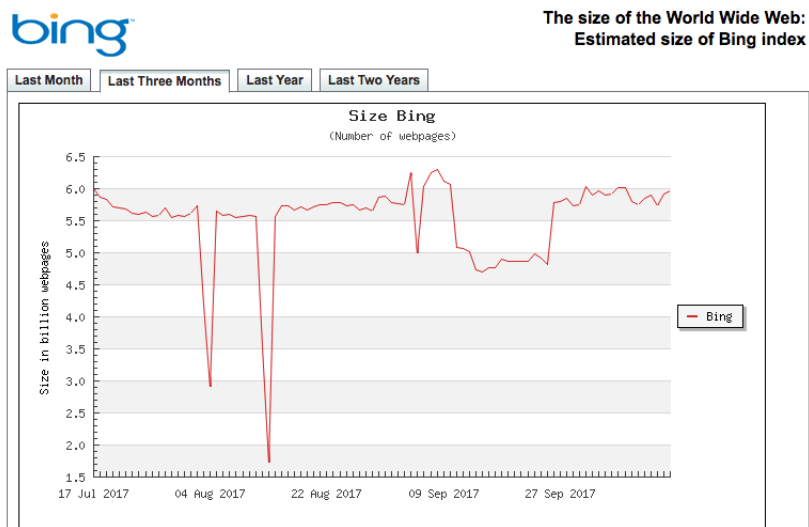


Figure 15: Bing's webpage size

4

Question

Answer

5

Question

Answer

References

- [1] Atkins, Grant. “CS532 Assignment 1 Repository” Github. N.p., 23 March 2017. Web. 23 March 2017.<https://github.com/grantat/cs532-s17/tree/master/assignments/A1/src>.
- [2] Atkins, Grant. “CS734 Assignment 2 Repository” Github. N.p., 21 September 2017. Web. 21 September 2017.<https://github.com/grantat/cs834-f17/tree/master/assignments/A2>.