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
            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

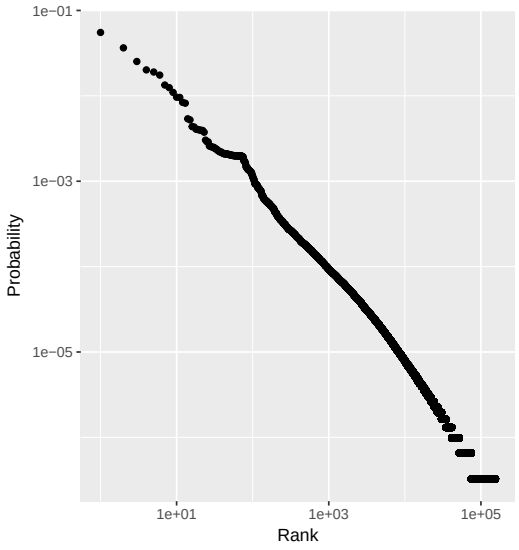

Figure 3: Log-log plot of unigram frequency and probability

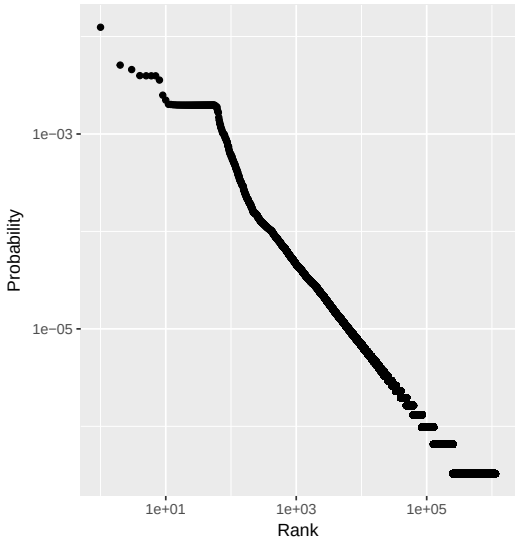

Figure 4: Log-log plot of bigram frequency and probability

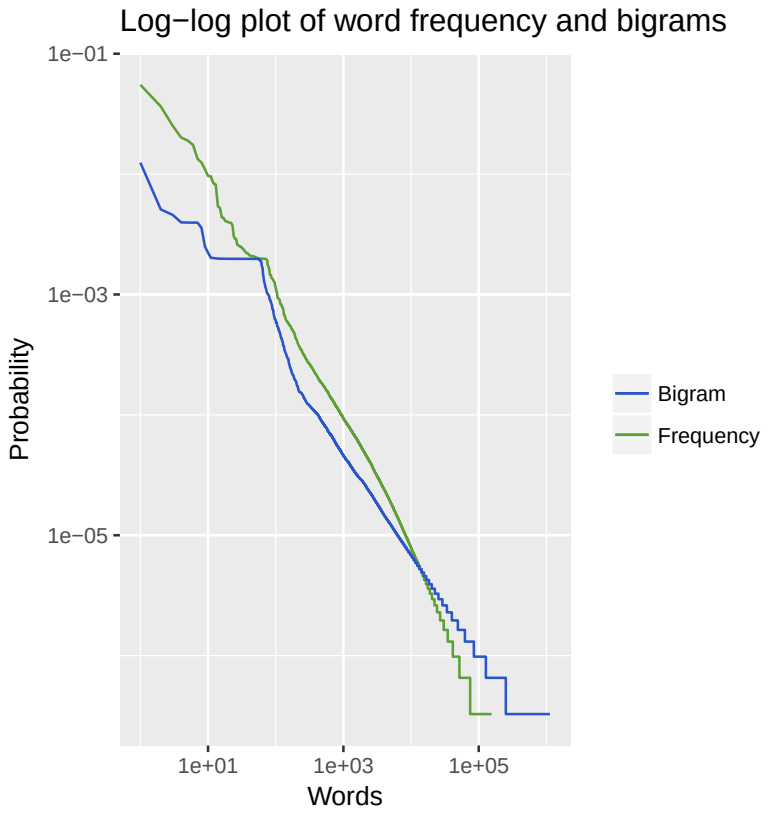

Figure 5: Log-log plot of unigram and bigram frequencies and probabilities

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

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

```
24
25 # execute on both data frames
26 vocab_growth(corpusGrowth)
27 vocab_growth(corpusGrowthRev)
```

Listing 3: Python script to tokenize and find frequencies and calculate C parameters

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). 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.

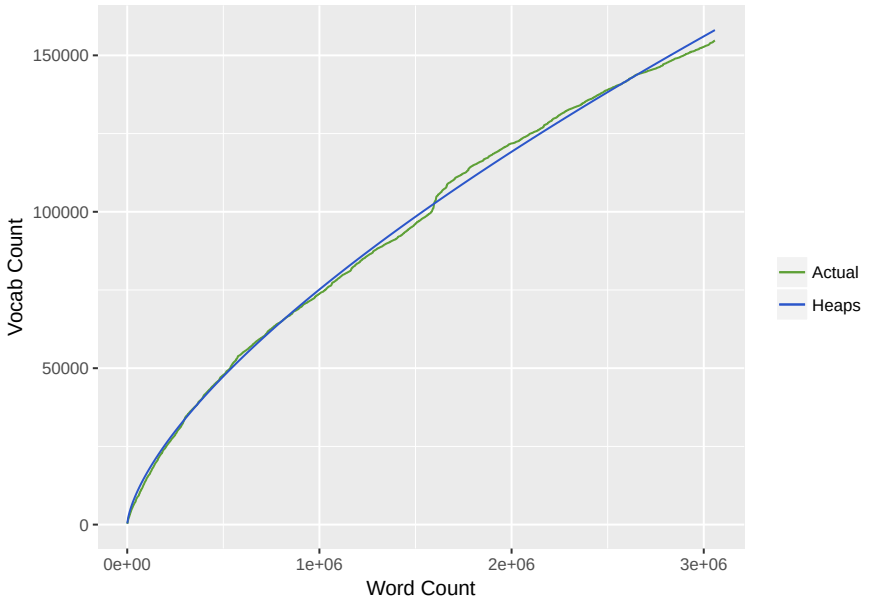

Figure 6: Growth of vocabulary vs overall corpus

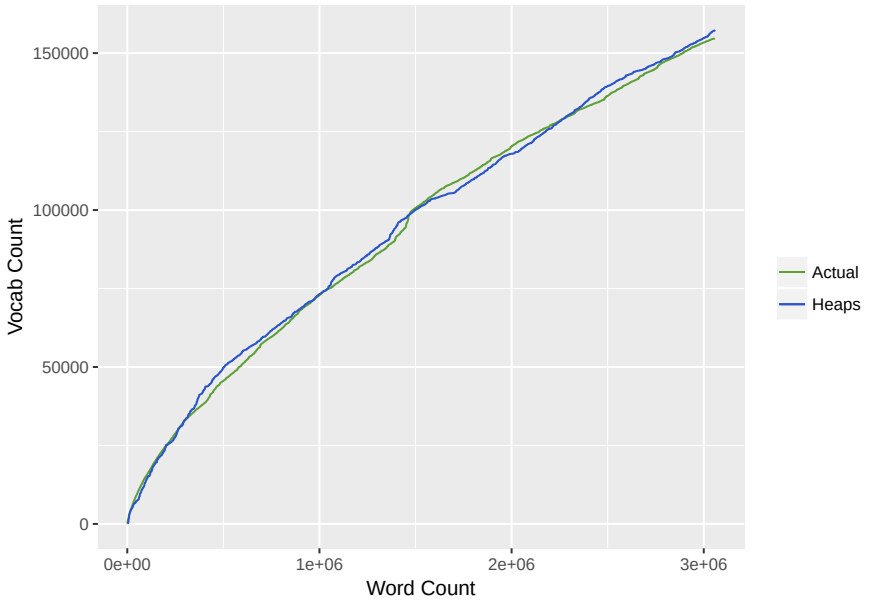

Figure 7: Growth of vocabulary vs overall corpus with documents traversed in reverse

3

Question

Answer

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.