Assignment 2

Fall 2016
CS834 Introduction to Information Retrieval
Dr. Michael Nelson

Mathew Chaney

October 12, 2016

Contents

List of Figures

Listings

List of Tables

1 Question 4.1

1.1 Question

Plot rank-frequency curves (using a log-log graph) for words and bigrams in the Wikipedia collection available through the book website (http://www.search-engines-book.com). Plot a curve for the combination of the two. What are the best values for the parameter c for each curve?

1.2 Resources

The textbook *Search Engines: Information Retrieval in Practice* [1], the Python programming language [2] with the python libraries Beautiful Soup [3] and NLTK [4], and the R programming language [5] were used to answer this question.

1.3 Answer

The wc.py script found in Listing 1 was used to locate each file of the Wikipedia collection obtained from the book download page, available at http://www.search-engines-book.com. The BeautifulSoup library was used to strip out the HTML tags and then the nltk library [4] was used to tokenize the text. The individual words were counted manually and the nltk library [4] was used to count the bigrams.

The word count graph can be found in Figure 1, the bigram count graph can be found in Figure 2, and the combination of the two can be found in Figure 3. The buildgraphs.R script was used to create these graphs and can be found in Listing 2.

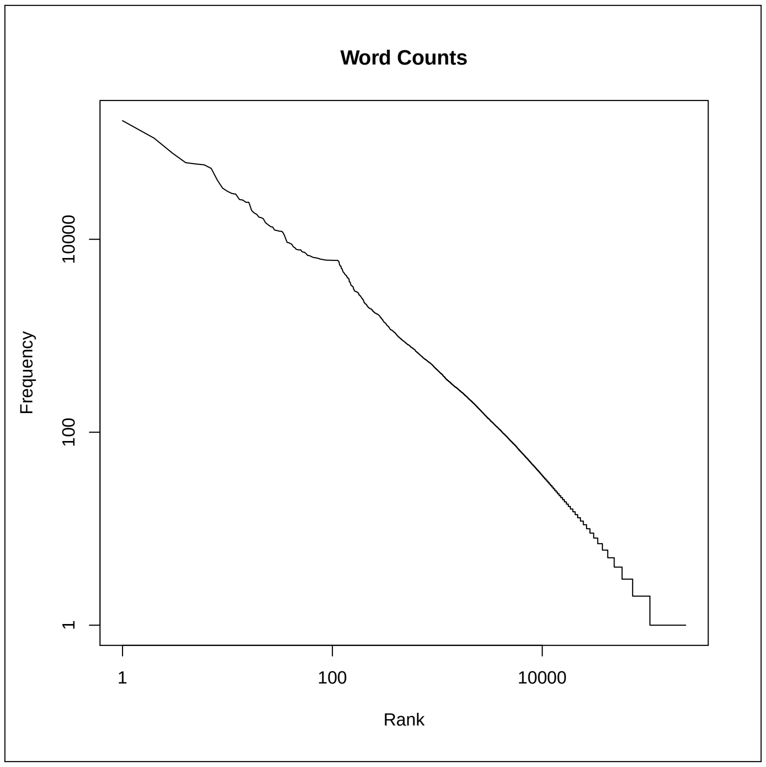

Figure 1: Word Counts for Small Wikipedia Corpus

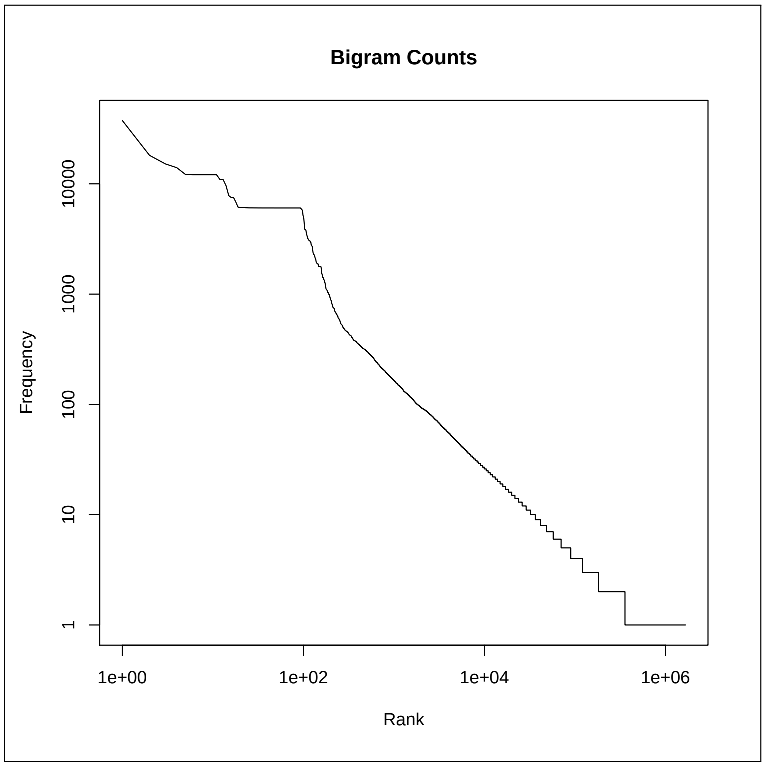

Figure 2: Bigram Counts for Small Wikipedia Corpus

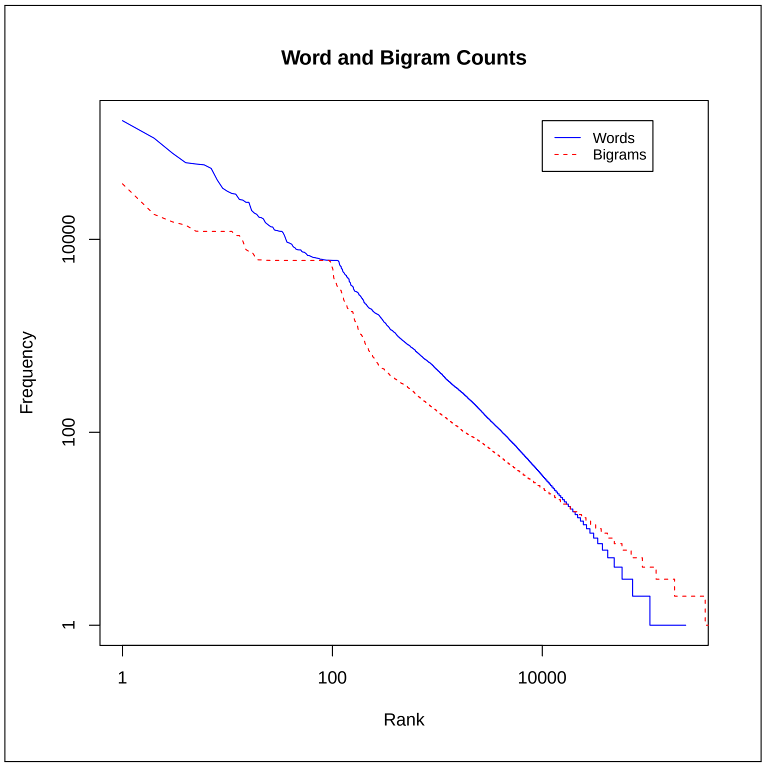

Figure 3: Both Word and Bigram Counts for Small Wikipedia Corpus

2 Appendix A

```
#!/usr/bin/python

import argparse
import os
import operator
import sys
import nltk
from os.path import isdir, isfile
from bs4 import BeautifulSoup

class NullCounter(object):
    def count(self, filepath, rawtext):
        pass
    def results(self):
        pass

class FileVisitor(object):
    def __init__(self, root, counters=[NullCounter()]):
        self.root = root
        self.counters = counters
        self.visited = 0

    def visit(self, folder=''):
        items = os.listdir(self.root + folder)
        for item in items:
            # if self.visited == 101:
            #     return
            filepath = self.root + folder + os.sep + item
            if isfile(filepath):
                sys.stdout.write("\rprocessing doc #%i" % self.visited)
                sys.stdout.flush()
                with open(filepath) as infile:
                    soup = BeautifulSoup(infile.read(), 'html.parser')
                    for counter in self.counters:
                        counter.count(filepath, soup)
                self.visited = self.visited + 1
            elif isdir(filepath):
                self.visit(folder + os.sep + item)

    def run(self):
        print 'delving into "{0}"'.format(self.root)
        self.visit()
        print
        for counter in self.counters:
            counter.results()
        print 'done'

class WordCounter(object):
    def __init__(self):
        self.tokenizer = nltk.RegexpTokenizer(r'\w+')
        self.wmap = {}
        self.invidx = {}
        self.bgmap = {}

    def count(self, filepath, soup):
        plaintext = soup.get_text()
        tokens = self.tokenizer.tokenize(plaintext)
        for s in tokens:
            if not self.wmap.has_key(s):
                self.wmap[s] = 0
            self.wmap[s] = self.wmap[s] + 1
            if not self.invidx.has_key(s):
                self.invidx[s] = set()
            self.invidx[s].add(filepath)
        for b in nltk.bigrams(tokens):
            if not self.bgmap.has_key(b):
                self.bgmap[b] = 0
            self.bgmap[b] = self.bgmap[b] + 1

    def results(self):
        print 'found {0} words'.format(len(self.wmap))
        print 'found {0} bigrams'.format(len(self.bgmap))
        with open('wordcount', 'w') as outfile:
            for k, v in sorted(self.wmap.items(), key=operator.itemgetter(1), reverse=True):
```

```python
                outfile.write(str(v) + '\t' + k.encode('utf-8') + '\n')
        with open('bigramcount', 'w') as outfile:
            for k, v in sorted(self.bgmap.items(), key=operator.itemgetter(1), reverse=True):
                outfile.write(str(v) + '\t' + k[0].encode('utf-8') + '\t' + k[1].encode('utf-8') + '\n')
        with open('invidx', 'w') as outfile:
            for k, v in sorted(self.invidx.items(), key=operator.itemgetter(1), reverse=True):
                outfile.write(k.encode('utf-8') + '\t')
                for page in v:
                    outfile.write(page + '\t')
                outfile.write('\n')

class InlinkCounter(object):
    def __init__(self):
        self.inlinks = {}

    def filter(self, href):
        if '../' not in href:
            return True

    def count(self, filepath, soup):
        links = soup.find_all('a')
        for link in links:
            if link.has_attr('href'):
                href = link['href']
                if self.filter(href):
                    continue
                href = href.replace('../', '')
                if not self.inlinks.has_key(href):
                    self.inlinks[href] = 0
                self.inlinks[href] = self.inlinks[href] + 1

    def results(self):
        with open('inlinks', 'w') as outfile:
            for k, v in sorted(self.inlinks.items(), key=operator.itemgetter(1), reverse=True):
                outfile.write(str(v) + '\t' + k.encode('utf-8') + '\n')


if __name__ == '__main__':
    parser = argparse.ArgumentParser('word count')
    parser.add_argument('-root', '-r', help='the root directory for parsing', default='en')
    args = parser.parse_args()
    visitor = FileVisitor(args.root, [WordCounter(), InlinkCounter()])
    visitor.run()
```

Listing 1: wc.py

```r
#! /usr/bin/Rscript

plotone <- function(data, outfile, title) {
    pdf(outfile)
    plot(data$V1, type='l', log='xy', main=title,
        ylab='Frequency', xlab='Rank', col="black")
    dev.off()
}

plottwo <- function(d1, d2, outfile, title) {
    pdf(outfile)
    y_range <- range(1, d1$V1, d2$V1)
    plot(d1$V1, type='l', log='xy', main=title, ylim=y_range,
        ylab='Frequency', xlab='Rank', col="blue")
    lines(d2$V1, type="l", lty=2, col="red")
    legend(10000, y_range[2], c('Words', 'Bigrams'), cex=0.8,
        col=c('blue', 'red'), lty=1:2)
    dev.off()
}

d1 <- read.table('wordcount')
d2 <- read.table('bigramcount')
plotone(d1, 'wc.pdf', 'Word Counts')
plotone(d2, 'bg.pdf', 'Bigram Counts')
plottwo(d1, d2, 'both.pdf', 'Word and Bigram Counts')
```

Listing 2: buildgraphs.R

3 References

[1] Bruce Croft, Donald Metzler, and Trevor Strohman. *Search Engines: Information Retrieval in Practice*. Pearson, first edition, February 2009.

[2] The Python Programming Language. Available at: https://www.python.org/. Accessed: 2016/09/17.

[3] Leonard Richardson. Beautiful Soup. Available at: https://www.crummy.com/software/beautifulsoup/. Accessed: 2016/09/20.

[4] Team NLTK http://www.nltk.org/team.html. Natural Language Toolkit. Available at: https://www.nltk.org/. Accessed: 2016/10/11.

[5] R Core Team https://cran.r-project.org/doc/FAQ/R-FAQ.html#What-is-R_003f. The R Programming Language. Available at: https://www.r-project.org/. Accessed: 2016/10/11.