Assignment 2

Fall 2016
CS834 Introduction to Information Retrieval
Dr. Michael Nelson

Mathew Chaney

October 13, 2016

Contents

List of Figures

Listings

List of Tables

1 Introduction

The filevisitor.py script, found in Listing 4, was used to complete the bulk of the work in completing these exercises. The scripts main function is to search the collection for documents and then perform some operations on each depending on its configuration. It is used to determine word and bigram counts (Question 4.1), vocabulary size and growth (Question 4.2), in-link count (Question 4.8) and build an inverted index for the document collection (Question 5.8).

In addition to the filevisitor.py script two R scripts (buildgraphs.R, found in Listing 5, and graphvocab.R, found in Listing 6), were used to create the graphics used for each exercise.

Finally, to demonstrate the completed inverted index for exercise 5.8, the search.py script was created. This script can be found in Listing 7.

2 Question 4.1

2.1 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?

2.2 Approach

The `FileVisitor` class found in Listing 1 recursively searches the directories of the Wikipedia collection.

```
15 class FileVisitor(object):
16     def __init__(self, root, counters=[NullCounter()]):
17         self.root = root
18         self.counters = counters
19         self.visited = 0
20 
21     def visit(self, folder=''):
22         items = os.listdir(self.root + folder)
23         for item in items:
24             # if self.visited == 101:
25             #     return
26             filepath = self.root + folder + os.sep + item
27             if isfile(filepath):
28                 sys.stdout.write("\rprocessing doc #%i" % self.visited)
29                 sys.stdout.flush()
30                 with open(filepath) as infile:
31                     soup = BeautifulSoup(infile.read(), 'html.parser')
32                     for counter in self.counters:
33                         counter.count(filepath, soup)
34                 self.visited = self.visited + 1
35             elif isdir(filepath):
36                 self.visit(folder + os.sep + item)
37 
38     def run(self):
39         print 'delving into "{0}"'.format(self.root)
40         self.visit()
41         print
42         for counter in self.counters:
43             counter.results()
44         print 'done'
```

Listing 1: The FileVisitor Class

As it finds files it performs various operations on those files to complete the different tasks required to complete the selected exercises. This is done by calling the `counter.count` method, found on line 33.

The BeautifulSoup library [1] is used to remove the HTML tags, and then the NLTK library [2] is used to tokenize the text. Each word is then counted manually using the `count` method of the WordCounter class, found in Listing 2, and the `bigram` method of the NLTK library [2] is used to count the bigrams. The `results` method then writes these counts to a file which will be used to create the graphs found in the Results section.

```
46 class WordCounter(object):
47     def __init__(self):
48         self.tokenizer = nltk.RegexpTokenizer(r'\w+')
49         self.wmap = {}
50         self.invidx = {}
51         self.bgmap = {}
52         self.vocab = {}
53         self.visited = 0
54 
55     def sum(self):
56         sum = 0
57         for k, v in sorted(self.wmap.items(), key=operator.itemgetter(0), reverse=True):
58             sum += v
59         return sum
60 
61     def count(self, filepath, soup):
62         plaintext = soup.get_text()
63         tokens = self.tokenizer.tokenize(plaintext)
64         for s in tokens:
65             if not self.wmap.has_key(s):
66                 self.wmap[s] = 0
67             self.wmap[s] = self.wmap[s] + 1
68             if not self.invidx.has_key(s):
69                 self.invidx[s] = set()
70             self.invidx[s].add(filepath)
71         for b in nltk.bigrams(tokens):
72             if not self.bgmap.has_key(b):
73                 self.bgmap[b] = 0
74             self.bgmap[b] += 1
75         self.visited += 1
76         if self.visited % 100 == 0:
77             s = self.sum()
78             self.vocab[len(self.wmap)] = s
79 
80     def results(self):
81         print 'found {0} words'.format(len(self.wmap))
82         print 'found {0} bigrams'.format(len(self.bgmap))
83         with open('wordcount.dat', 'w') as outfile:
84             for k, v in sorted(self.wmap.items(), key=operator.itemgetter(1), reverse=True):
85                 outfile.write(str(v) + '\t' + k.encode('utf-8') + '\n')
86         with open('bigramcount.dat', 'w') as outfile:
87             for k, v in sorted(self.bgmap.items(), key=operator.itemgetter(1), reverse=True)
                :
88                 outfile.write(str(v) + '\t' + k[0].encode('utf-8') + '\t' + k[1].encode('utf
                    -8') + '\n')
89         with open('invidx.dat', 'w') as outfile:
90             for k, v in sorted(self.invidx.items(), key=operator.itemgetter(1), reverse=True
                ):
91                 outfile.write(k.encode('utf-8') + '\t')
92                 for page in v:
93                     outfile.write(page + '\t')
94                 outfile.write('\n')
95         with open('vocab.dat', 'w') as outfile:
96             for k, v in sorted(self.vocab.items(), key=operator.itemgetter(1)):
97                 outfile.write(str(k) + '\t' + str(v) + '\n')
```

Listing 2: The WordCounter Class

2.3 Results

The `buildgraph.R` script, found in Listing 5 was used to create the following graphs. 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 5.

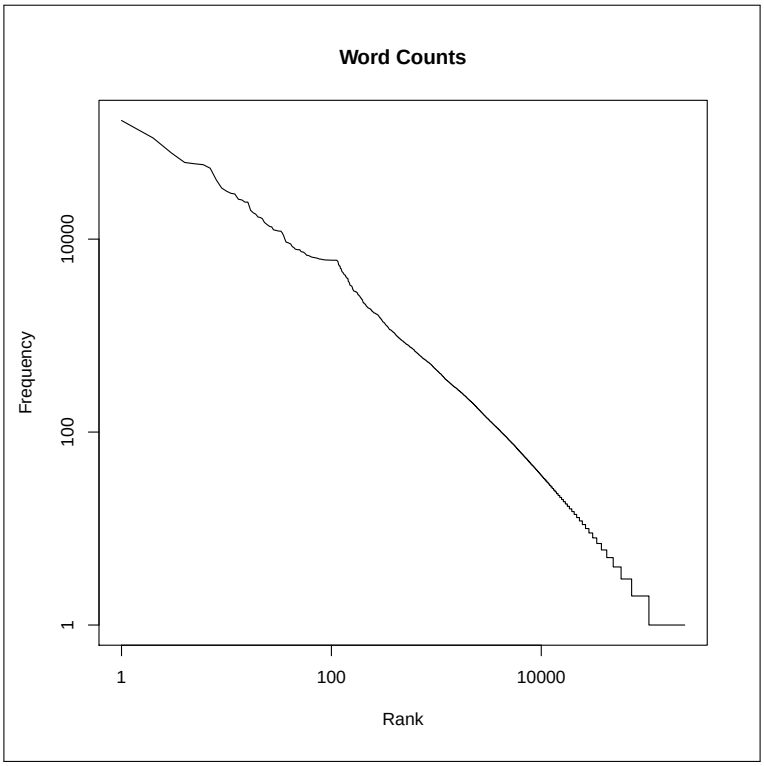

Figure 1: Word Counts for Small Wikipedia Corpus

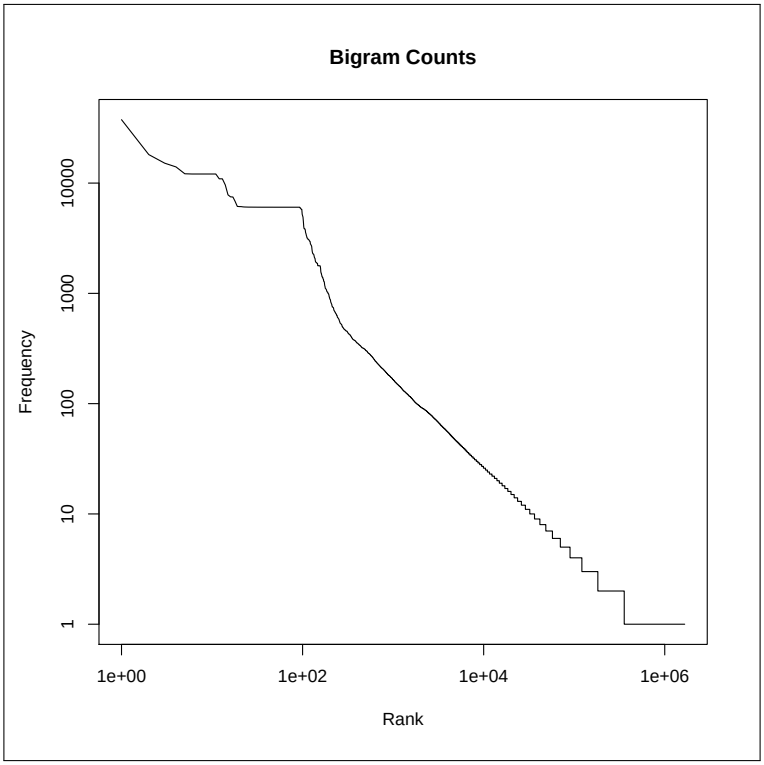

Figure 2: Bigram Counts for Small Wikipedia Corpus

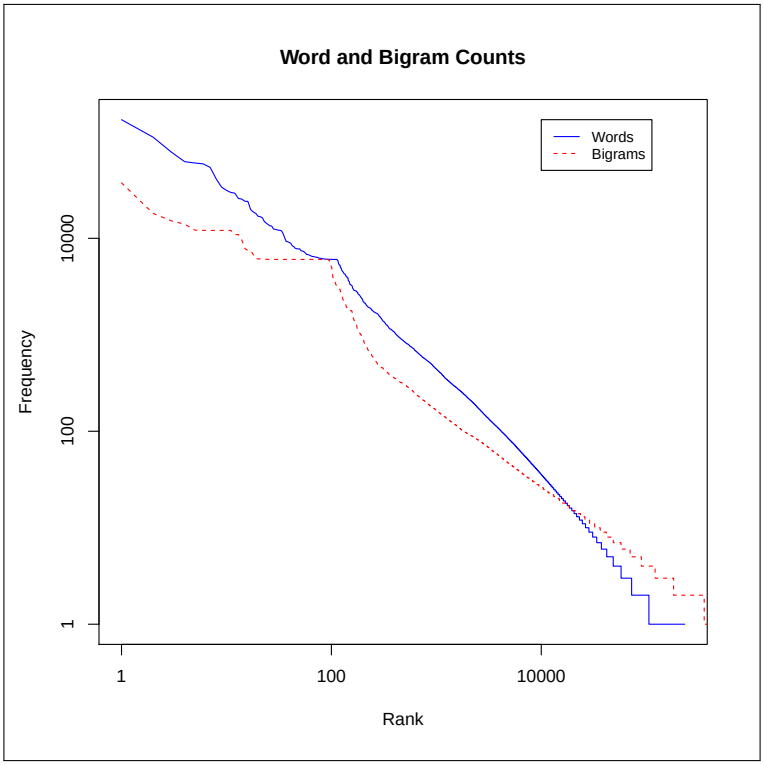

Figure 3: Both Word and Bigram Counts for Small Wikipedia Corpus

3 Question 4.2

3.1 Question

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?

3.2 Answer

The filevisitor.py script found in Listing 4 was modified to also log the vocabulary and total word count after visiting each document in the small Wikipedia collection. The graphvocab.R script found in Listing 6 was then used to create the Vocabulary Growth graph, which can be found in Figure 4.

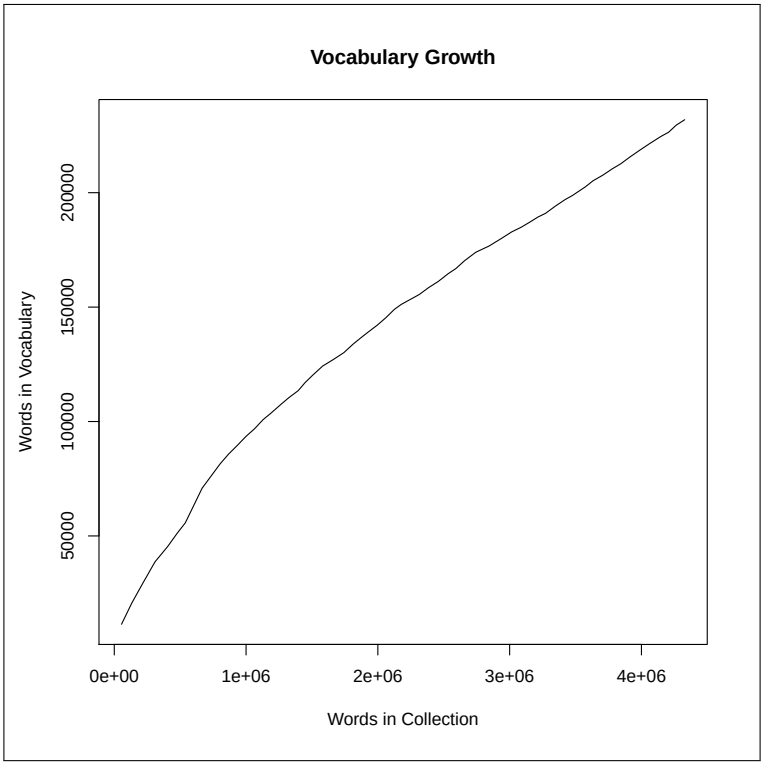

Figure 4: Vocabulary Growth for the Small Wikipedia Collection

4 Question 4.8

4.1 Question

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

4.2 Resources

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

4.3 Answer

The filevisitor.py script found in Listing 4 was modified to track inlink data for each document of the small Wikipedia collection. The pages with the highest inlink count can be found in Table 1.

Page URI	Inlink Count
articles/2/0/0/2007.html	2264
articles/s/m/a/User%7ESmackBot_cc7a.html	1896
articles/2/0/0/2008.html	1770
articles/u/n/i/United_States_09d4.html	1363
articles/2/0/0/2006.html	982
articles/a/l/a/User%7EAlaibot_de3d.html	791
articles/c/y/d/User%7ECydebot_38a6.html	676
articles/l/i/v/Category%7ELiving_people_7259.html	675
articles/b/l/u/User%7EBluebot_e595.html	663
articles/g/e/o/Geographic_coordinate_system.html	655

Table 1: Top Ten Pages With Most Inlinks

5 Question 5.8

5.1 Question

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
```

5.2 Resources

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

5.3 Answer

Again, the filevisitor.py script found in Listing 4 was modified to create the inverted index while visiting each file from the small Wikipedia collection. Afterwards, the search.py 7 script can be used to search for terms within the document collection.

Example output for searching the inverted index for the terms "Guy" and "Gal" can be found in Listing 3.

```
1 [mchaney@mchaney-l search]$ python search.py -t guy gal
2 guy en/articles/s/a/m/Sam_Endicott_c462.html en/articles/b/r/a/Brazil.html en/articles/b/r/o
      /Broderick_Crawford_c49a.html en/articles/h/e/n/Henry_VII_of_England_2c03.html en/
      articles/t/r/o/Tropical_Storm_Chris_(2006)_8d97.html en/articles/e/r/i/Eric_Weddle_ffa0.
      html en/articles/d/e/c/Decahedron.html en/articles/y/o/s/Yosaku.html en/articles/m/a/d/
      Madhu_Sapre_99e7.html en/articles/h/a/p/Happy_Feet_ce15.html en/articles/t/o/t/
      Tottenham_Hotspur_F.C._6bd2.html en/articles/a/f/f/Affirmation_in_law.html en/articles/d
      /1/0/D10.html en/articles/d/o/n/Don_Adams_9d39.html en/articles/b/a/t/
      Battle_of_Turnhout_150e.html en/articles/g/r/e/Greg_Lloyd_5b29.html en/articles/w/i/l/
      William_Forsythe_(actor)_c82e.html en/articles/f/o/x/Fox_News_Channel_controversies_eea8
      .html en/articles/t/h/e/The_Fuck_Buddy_a922.html en/articles/t/a/r/Taranee_Cook_f3aa.
      html en/articles/g/r/a/Grave_Danger_9ee5.html en/articles/c/o/l/Coleman_Hawkins_90ff.
      html en/articles/s/l/a/Slappy_the_Dummy_48f0.html en/articles/n/a/z/Nazi_Party_3bfc.html
       en/articles/t/h/e/The_Bible_and_homosexuality_0961.html en/articles/w/a/l/
      Walter_Emanuel_Jones_2d14.html en/articles/r/e/i/Reid_Paley_6e3c.html en/articles/g/o/g/
      Goguryeo_language.html en/articles/g/i/v/Give_Up_59c0.html en/articles/s/t/r/
      Strapping_Young_Lad_d063.html en/articles/m/a/s/Master_of_Computer_Applications_a97f.
      html en/articles/c/a/l/Calabash_(disambiguation).html en/articles/d/r/i/Driving_test.
      html en/articles/b/e/r/Berlin_Wall_24be.html en/articles/t/h/e/The_Great_Compromise_(
      song)_3e9e.html en/articles/f/l/i/Flip-flop_(electronics).html
3 gal en/articles/a/r/e/Area_code_760.html en/articles/g/e/o/George_W._C._Baker_5636.html en/
      articles/g/e/o/George_Brown,_Jr._784d.html en/articles/f/-/1/F-102_Delta_Dagger_058e.
      html en/articles/h/o/n/Honeoye_Lake_cb81.html en/articles/d/u/m/Dumuzid,
      _the_Shepherd_7fad.html en/articles/s/v/e/Sveadal,_California_1cd9.html en/articles/a/g/
      u/Agua_Dulce,_California_c05c.html en/articles/l/o/m/Lompoc,_California_d9b1.html en/
      articles/t/o/y/Toyota_Verossa_e6a6.html en/articles/f/-/8/F-84_Thunderjet_cdc0.html
```

Listing 3: Search Results for Terms "Guy" and "Gal"

6 Appendix

```
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 = {}
        self.vocab = {}
        self.visited = 0

    def sum(self):
        sum = 0
        for k, v in sorted(self.wmap.items(), key=operator.itemgetter(0), reverse=True):
            sum += v
        return sum

    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] += 1
```

```
75          self.visited += 1
76          if self.visited % 100 == 0:
77              s = self.sum()
78              self.vocab[len(self.wmap)] = s
79
80      def results(self):
81          print 'found {0} words'.format(len(self.wmap))
82          print 'found {0} bigrams'.format(len(self.bgmap))
83          with open('wordcount.dat', 'w') as outfile:
84              for k, v in sorted(self.wmap.items(), key=operator.itemgetter(1), reverse=True):
85                  outfile.write(str(v) + '\t' + k.encode('utf-8') + '\n')
86          with open('bigramcount.dat', 'w') as outfile:
87              for k, v in sorted(self.bgmap.items(), key=operator.itemgetter(1), reverse=True)
                    :
88                  outfile.write(str(v) + '\t' + k[0].encode('utf-8') + '\t' + k[1].encode('utf
                        -8') + '\n')
89          with open('invidx.dat', 'w') as outfile:
90              for k, v in sorted(self.invidx.items(), key=operator.itemgetter(1), reverse=True
                    ):
91                  outfile.write(k.encode('utf-8') + '\t')
92                  for page in v:
93                      outfile.write(page + '\t')
94                  outfile.write('\n')
95          with open('vocab.dat', 'w') as outfile:
96              for k, v in sorted(self.vocab.items(), key=operator.itemgetter(1)):
97                  outfile.write(str(k) + '\t' + str(v) + '\n')
98
99  class InlinkStruct(object):
100     def __init__(self):
101         self.c = 0
102         self.a = []
103
104     def __repr__(self):
105         return str(self.c) + '\t' + str(self.a)
106
107 class InlinkCounter(object):
108     def __init__(self):
109         self.inlinks = {}
110
111     def filter(self, href):
112         if '../' not in href \
113         or 'Wikipedia%7E' in href \
114         or 'Portal%7E' in href \
115         or 'Help%7E' in href \
116         or 'Special%7' in href \
117         or href.replace('../','') == 'index.html':
118             return True
119
120     def count(self, filepath, soup):
121         links = soup.find_all('a')
122         for link in links:
123             if link.has_attr('href'):
124                 href = link['href']
125                 if self.filter(href):
126                     continue
127                 href = href.replace('../', '')
128                 if not self.inlinks.has_key(href):
129                     self.inlinks[href] = InlinkStruct()
130                 self.inlinks[href].c += 1
131                 self.inlinks[href].a.append(link.text)
132
133     def results(self):
134         with open('inlinks.dat', 'w') as outfile:
135             for k, v in sorted(self.inlinks.items(), key=operator.itemgetter(1), reverse=
                    True):
136                 outfile.write(str(v.c) + '\t' + k.encode('utf-8') + '\t' + str(v.a) + '\n')
137
138
139 if __name__ == '__main__':
140     parser = argparse.ArgumentParser('word count')
141     parser.add_argument('-root', '-r', help='the root directory for parsing', default='en')
142     args = parser.parse_args()
143     visitor = FileVisitor(args.root, [WordCounter(), InlinkCounter()])
144     visitor.run()
```

Listing 4: filevisitor.py

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

Listing 5: buildgraphs.R

```
data <- read.table('vocab')

pdf("vocab.pdf")
plot(data$V2, data$V1, type="l", main="Vocabulary Growth",
    ylab="Words in Vocabulary", xlab="Words in Collection")
dev.off()
```

Listing 6: graphvocab.R

```
import argparse

if __name__ == '__main__':
    parser = argparse.ArgumentParser('basic search engine')
    parser.add_argument('-file','-f', help='the file to search', default='invidx.dat')
    parser.add_argument('-terms', '-t', nargs='+', help='terms to search for')
    args = parser.parse_args()

    #initialize results map
    results = {}
    for term in args.terms:
        results[term] = set()

    # iterate over inverted index, matching terms and docs
    with open(args.file) as infile:
        for line in infile:
            parts = line.split('\t')
            term = parts[0]
            docs = parts[1:]
            if term in args.terms:
                for doc in docs:
                    results[term].add(doc)

    # print results
    for term, docs in results.items():
        print term,' '.join(docs)
```

Listing 7: search.py

7 References

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

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

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

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

[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.