Assignment 2

CS834 Introduction to Information Retrieval
Fall 2017

Polykarpos Thomadakis

October 7, 2017

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.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?

Answer

To answer this question, I wrote a Python script that parses each documents in the Wikipedia collection and extracts all of its tokens. Then, count the frequency of each token, create bigrams from those tokens and count the frequency of the bigrams.
Finally, for both tokens and bigrams rank them by their frequency and compute their probability and the 'c' value. The output is written on a csv file for easier visualization into tables. For the purposes of this assignment I used the Wiki small collection of the book's website.
The script can be found in listing 1.

```
import os
import csv
from bs4 import BeautifulSoup
import nltk

# Set the path to search for html files
root_dir = 'en'

# Set the tokenizer exrpresion
tokenizer = nltk.RegexpTokenizer(r'\w+')

# Find all files we need to parse
html_files = []
for root,dirs,files in os.walk(root_dir):
  for file in files:
    if file.endswith('.html'):
      html_files.append(os.path.join(root, file))



# Get the tokens for all files
tokens_found = {}
num_words = 0
tokens_in_bigrams = {}
for file in html_files:
  with open(file) as curr_file:
    print file
    soup = BeautifulSoup(curr_file.read(),'html.parser')
    tokens= tokenizer.tokenize(soup.get_text())
    for token in tokens:
      token = token.encode('utf-8')
      if token not in tokens_found:
        tokens_found[token] = 0
      tokens_found[token] +=1
      num_words +=1
    # Get the bigrams from the tokens
    for token in nltk.bigrams(tokens):
      token = (token[0].encode('utf-8'),token[1].encode('utf-8'))
      if token not in tokens_in_bigrams:
        tokens_in_bigrams[token] = 0
      tokens_in_bigrams[token] += 1

# Write token resutls to a file
f = open('results.csv','w+')
writer = csv.writer(f)
writer.writerow(('Rank','Frequency','Word','Probability','c'))
rank=0
for w in sorted(tokens_found,key=tokens_found.get,reverse=True):
  rank+=1
  writer.writerow( (rank ,tokens_found[w] ,w ,float(tokens_found[w])/num_words ,(
    float(tokens_found[w])/num_words)*rank ))

f.close()


# Write bigram resutls to a file
f = open('results_bigrams.csv','w+')
writer = csv.writer(f)
writer.writerow(('Rank','Frequency','Word','Probability','c'))
rank=0
for w in sorted(tokens_in_bigrams,key=tokens_in_bigrams.get,reverse=True):
  rank+=1
  writer.writerow( (rank,tokens_in_bigrams[w],w,float(tokens_in_bigrams[w])/num_words
    ,(float(tokens_in_bigrams[w])/num_words)*rank ))

f.close()
```

Listing 1: Script to extract toke and bigram information

The top 10 tokens in terms of frequency are shown in Table 1 . In file results.csv you can find the complete results for the tokens' frequency ranking in this collection. Table 2 shows the same information for the case of the bigrams as can be also found in results_bigrams.csv. To generate the requested graphs I used gnuplot on the results.csv, results_bigrams.csv files. The log-log rank-frequency plots for the tokens, bigrams and their comparison are presented in Figures 1, 2 and 3, respectively.

Table 1: Top ten ranked tokens in Wiki small collection

Rank	Frequency	Word	Probability	c
1	169695	the	0.038954823010881046	0.038954823010881046
2	111772	of	0.025658142417703502	0.051316284835407004
3	77940	and	0.01789174050778201	0.05367522152334603
4	62273	a	0.014295257334373996	0.057181029337495984
5	60351	Wikipedia	0.01385404710527524	0.06927023552637619
6	59125	in	0.013572609154767917	0.08143565492860749
7	54327	to	0.012471190487121803	0.08729833340985262
8	41122	is	0.009439878793443827	0.07551903034755061
9	33865	by	0.0077739773196822915	0.06996579587714062
10	31434	The	0.007215922133969974	0.07215922133969974

Table 2: Top ten ranked bigrams in Wiki small collection

Rank	Frequency	Word	Probability	c
1	37574	(’of’, ’the’)	0.008625407465221982	0.008625407465221982
2	15151	(’in’, ’the’)	0.003478031311693678	0.010434093935081035
3	14010	(’is’, ’a’)	0.0032161057802672054	0.012864423121068821
4	12147	(’the’, ’free’)	0.0027884394655892752	0.013942197327946377
5	12088	(’free’, ’encyclopedia’)	0.0027748955511684497	0.016649373307010697
6	12088	(’Wikipedia’, ’the’)	0.0027748955511684497	0.019424268858179147
7	12086	(’About’, ’Wikipedia’)	0.0027744364354253706	0.027744364354253707
8	10932	(’Wikipedia’, ’user’)	0.0025095266516688857	0.03011431982002663
9	10932	(’by’, ’Wikipedia’)	0.0025095266516688857	0.032623846471695514
10	9622	(’user’, ’s’)	0.002208805839952252	0.030923281759331532

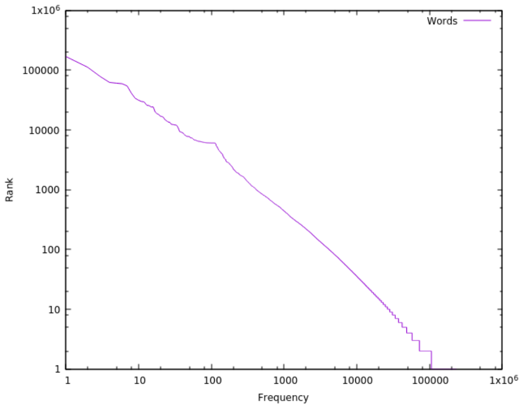

Figure 1: The rank-frequency curve for the words in the Wiki small collection

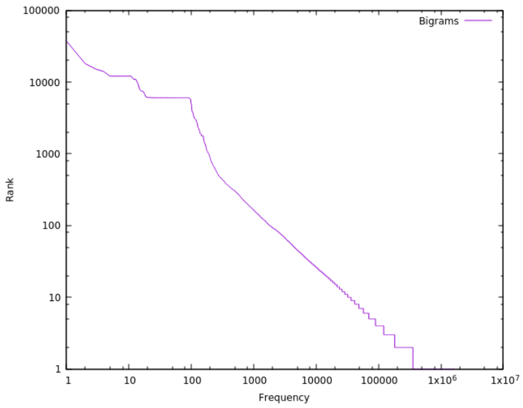

Figure 2: The rank-frequency curve for the bigrams in the Wiki small collection

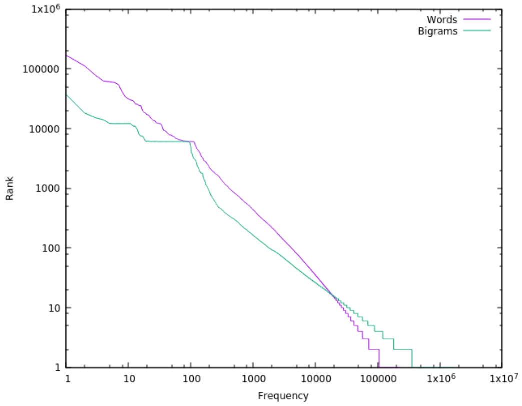

Figure 3: The rank-frequency curve for the bigrams and words in the Wiki small collection

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

For this assignment we should count the number of words per document and the number of unique words. This way we have a analogy between the total number of words and the number of unigue words in different points steps of parsing the collection of documents. I used the Wiki small collection again. The script used is shown in listing 2. It can take as option argument the -r option that will make it parse the documents in reverse order for the purposes of this assignment. The parameters that need to be defined are k and β from Heap's law:

$$v = kn^{\beta}$$

In order to estimate those values, I used non-linear least square fitting from python. The script used for this purpose is presented in listing 3. The curves produced by using the values found from this fitting and the curves generated by the experiments are shown in figure 4 and figure 5. Using this methodology I get the results shown in table 3, which point out that the order in which the files are visited can impact the vocabulary growth curve.

```
import os
import csv
from bs4 import BeautifulSoup
import nltk
import argparse


parser = argparse.ArgumentParser('vocgrowth')
parser.add_argument('--reverse','-r',help='Visit files in reverse order',default=
    False,type=bool)
args = parser.parse_args();
reverse = args.reverse
# Set the path to search for html files
root_dir = 'en'

# Set the tokenizer exrpresion
tokenizer = nltk.RegexpTokenizer(r'\w+')

# Find all files we need to parse
html_files = []
for root,dirs,files in os.walk(root_dir):
  for file in files:
    if file.endswith('.html'):
      html_files.append(os.path.join(root, file))

if reverse:
  html_files.reverse()
# Get the tokens for all files
tokens_found = []
voc_growth = []
i = 0;
for file in html_files:
  i+=1
  with open(file) as curr_file:
    print file
    soup = BeautifulSoup(curr_file.read(),'html.parser')
    tokens= tokenizer.tokenize(soup.get_text())
    for token in tokens:
      token = token.encode('utf-8')
      tokens_found.append(token)
    voc_growth.append([i ,len(tokens_found),len(set(tokens_found))])

# Write token resutls to a file
if reverse:
  f = open('voc_growth_reverse.csv','w+')
else:
  f = open('voc_growth.csv','w+')
writer = csv.writer(f)
writer.writerow(('Files','Words','Vocabulary_size'))
for v in voc_growth:
  writer.writerow( (v[0],v[1],v[2]))

f.close()
```

Listing 2: Script to extract the vocabulary growth information

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import leastsq
import csv

f = open('voc_growth_reverse.csv','r') # Or voc_growth.csv :in order parsed files
i=0
xdata =[]
ydata = []
for line in f:
  if(i==0):
    i=1
    continue
  parts = line.split(',')
  xdata.append(int(parts[1]))
  ydata.append(int(parts[2].strip()))
f.close()
# print len(xdata)
# print len(ydata)
xx= np.array(xdata)
yy= np.array(ydata)
p0 = np.array([1,1])

def my_fun(par,x,y):
  return y-par[0]*np.power(x,par[1])

def fun_eval(par,x):
  return par[0]*np.power(x,par[1])

# print 1*p0[0]
result = leastsq(my_fun,p0[:],args=(xx,yy))
f = open('fit.csv','w+')
writer = csv.writer(f)
for x in xx:
  writer.writerow((x,fun_eval(result[0],x)))
k,b = result[0]
print "k = "+str(k),
print "b = "+str(b)
f.close()
```

Listing 3: Script that calculates the parameters of Heap's law using non-linear least square fitting

Table 3: Heap's law parameters

	In order	Reverse
k	25.1230	6.807
β	0.597	0.680

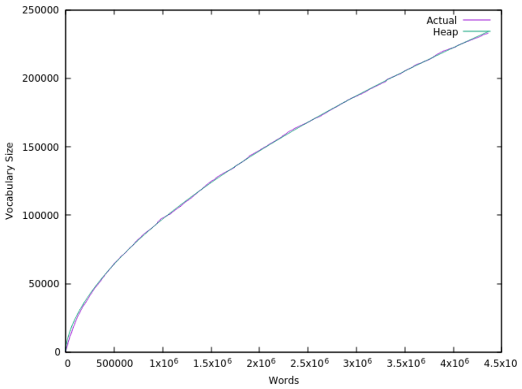

Figure 4: The vocabulary growth for the Wiki small collection

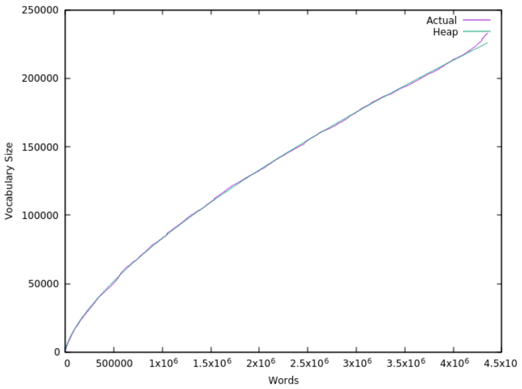

Figure 5: The vocabulary growth for the Wiki small collection parsed in reverse order

Table 4: Comparison between the two stemming algorithms for the five html files chosen

No	File	Porter stems #	Krovetz stems #
1	Agia_Dynati_f24a.html	230	227
2	Agnes_of_Glasgow_3312.html	346	342
3	Agoranomos.html	454	450
4	Astolfo.html	628	632
5	Parand.html	738	746

Question 4.6

Process five Wikipedia documents using the Porter stemmer and the Krovetz stemmer. Compare the number of stems produced and find 10 examples of differences in the stemming that could have an impact on ranking.

Answer

The five documents I chose for this assignment are shown below in table 4 with the number of stems per algorithm. I use existing Python's libraries to retrieve the results of both stemming algorithms. Krovetz's algorithm is more sophisticated and this can be realized by the 10 differences presented in table 5. As one can notice, Porter stemming produces many stems that have no meaning in English, thus, impacting the overalll ranking of the document. The script used to collect this information is given in listing 4. This scripts produces 2 files per original document, containing the Porter and Krovatz stemming respectively. The output can be found in my github repository in filenames ending in .krovetz or .porter.

```
import io
import sys
import os
from bs4 import BeautifulSoup
import nltk
from nltk.stem import *
import krovetzstemmer

tokenizer = nltk.RegexpTokenizer(r'\w+')
porterizer = PorterStemmer()
krovetzizer = krovetzstemmer.Stemmer()

print('~~~~~~~~~~~~~~~~~~~~~~~~~Starting~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~')
```

```
if len(sys.argv)< 6:
  print('Please provide 5 files to process')
  print('porter_krovetz <file1> <file2> <file3> <file4> <file5>')
  exit()

files = []
for arg in sys.argv[1:]:
  files.append(arg)

porter_stems = []
krovetz_stems = []
header = ('No','File','Porter stems #','Krovetz stems #')
results = []
i=0
print header
for file in files:
  i+=1
  with open(file) as curr_file:
    # print file
    soup = BeautifulSoup(curr_file.read(),'html.parser')
    tokens=  tokenizer.tokenize(soup.get_text())
    porter = []

    f = open(os.path.split(file)[1],'w')
    p = open(os.path.split(file)[1]+'.porter','w')
    k = open(os.path.split(file)[1]+'.krovetz','w')
    words = 0;
    for token in tokens:
      if(token.isalnum()):
        words+=1
        f.write(token.lower().encode('utf-8')+' ')
        porter_stems.append(porterizer.stem(token))
        krovetz_stems.append(krovetzizer.stem(token))
        p.write(porterizer.stem(token).encode('utf-8')+' ')
        k.write(krovetzizer.stem(token).encode('utf-8')+' ')
        if(words%15 == 0):
          f.write('\n')
          p.write('\n')
          k.write('\n')
  results.append((i,os.path.split(file)[1],len(set(porter_stems)),len(set(
    krovetz_stems))))

  print results[i-1]
```

Listing 4: Script to extract stems from the 5 documents using the two given algorithms

Table 5: Ten differences in the stems produced by the 2 algorithms

Original	Porter	Krovetz
produces	produc	produce
believe	believ	believe
greece	greec	greece
because	becaus	because
his	hi	his
zeus	zeu	zeus
during	dure	during
infamous	infam	infamous
massacre	massacr	massacre
division	divis	division

Table 6: Top ten documents with the most inlinks with their anchors

No	Inlinks #	Document	Anchor texts
1	123	en/articles/b/r/a/Brazil.html	Brazil,BRA,Brazilian
2	44	en/articles/a/u/g/August_26.html	08-26,26,August 26,26 August
3	17	en/articles/m/a/n/Manga.html	manga,Manga
4	14	en/articles/m/a/g/Magazine.html	magazine,magazines,Magazine
5	12	en/articles/m/o/l/Mollusca.html	Mollusca
6	11	en/articles/v/i/c/Victoria_of_the_United_Kingdom_5e8e.html	Victoria,Queen,Victoria of the United Kingdom,Queen Victoria
7	8	en/articles/s/c/r/Screenwriter.html	Writer(s),screenwriter,Screenwriter
8	7	en/articles/t/u/s/Tuscany.html	Tuscany
9	6	en/articles/k/i/d/Kidney.html	kidneys,Renal,kidney
10	6	en/articles/t/o/t/Tottenham_Hotspur_F.C._6bd2.html	Tottenham Hotspur,Tottenham

Question 4.8

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

Answer

The methodology for this assignment is to parse all files in the Wiki small collection, maintaining a dictionary of documents to number of inlinks and another of inlinks to anchor texts. Once all files have been parsed, the first dictionary is sorted by the number of inlinks and then the second dictionary is printed based on the order that the first dictionary has given to its keys. Only the links that point to file in the collection are considered for this assignment. The results of the algorithm can are presented in table 6. The Python script for this assignment is presented in listing 5

```
import os
import csv
import operator
from bs4 import BeautifulSoup
import nltk

root_dir = 'en'

# Set the tokenizer exrpresion
tokenizer = nltk.RegexpTokenizer(r'\w+')

# Find all files we need to parse
html_files = []
for root,dirs,files in os.walk(root_dir):
  for file in files:
    if file.endswith('.html'):
      html_files.append(os.path.join(root, file))
file_str = []
for file in html_files:
  file_str.append( str(file))
inlinks = {}
anchor_texts = {}

for file in html_files:
  print file
  soup = BeautifulSoup(open(file),'html.parser')
  links = soup.find_all('a')
  for link in links:
    new_link = link.get('href')
    if not new_link:
      continue
    if new_link.startswith('http'):
      continue
    new_link = new_link.replace('../','')
    new_link = new_link.encode('utf-8')
    new_link = 'en/'+new_link
    if new_link in file_str and file !=new_link:
      if new_link not in inlinks:
        inlinks[new_link] = 0
        anchor_texts[new_link] = set()
      inlinks[new_link] += 1
      anchor_texts[new_link].add(link.text)
i=0
f = open("Inlinks_count","w")
writer = csv.writer(f)
writer.writerow(('No','Document','Inlinks #','Anchor texts'))
for k,v in sorted(inlinks.items(), key=operator.itemgetter(1),reverse=True):
  i+=1
  print i,str(v), k,
  anchors = ''
  for it in anchor_texts[k]:
    anchors+=it+','
  writer.writerow((i,str(v), k,anchors))
  if(i==10):
    break;
f.close()
```

Listing 5: Script to extract the top 10 documents with most inlinks and their anchor texts

Question 5.8

Write a program that can build a simple inverted index of a set of text docu- ments. 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

In order to create the inverted index, I parse all the documents given, maintaining a dictionary of tokens to sets of files. For each token I keep all the documents that contain this term. The script used takes as command argument a list of documents to parse, or a filepath to search for all .html files below it and creates a file containing the inverted index. A part of the file produced is shown in table 7. I used the files Enayetpur.html,Gabrias.html for this assignment. The script that generates the inverted index is presented in listing 6

```
import os
import csv
from bs4 import BeautifulSoup
import nltk
import argparse

tokenizer = nltk.RegexpTokenizer(r'\w+')

parser = argparse.ArgumentParser('invindex')
group = parser.add_mutually_exclusive_group(required= True)
group.add_argument('-l','--list', nargs='+', help='List of documents to parese', required=False)
group.add_argument('-p','--path', help='Path to directory to parse all of its files',
     required=False)
args = parser.parse_args();

html_files = []
if(args.path):
  root_dir = args.path
  for root,dirs,files in os.walk(root_dir):
    for file in files:
      if file.endswith('.html'):
        html_files.append(os.path.join(root, file))
else:
  html_files = [x for x in args.list if x.endswith('.html')]

inv_index = {}


for file in html_files:
  with open(file) as f:
    soup = BeautifulSoup(f.read(),'html.parser')
    tokens=  tokenizer.tokenize(soup.get_text())
    for token in tokens:
      token = token.encode("utf-8")
      if token not in inv_index:
        inv_index[token] = set()
      inv_index[token].add(os.path.split(file)[1])
  # break;

outfile = open("inv_file.csv","w")
for token in inv_index:
  outfile.write(token+",")
  i=0
  outfile.write('"')
  for file in inv_index[token]:
    i+=1
    outfile.write(file)
    if(i != len(inv_index[token])):
      outfile.write(",")
  outfile.write('"\n')
```

Listing 6: Script to create an inverted index

Table 7: The 30 first rows of the inverted index produced

Token	Documents
Based	Enayetpur.html,Gabrias.html
help	Enayetpur.html,Gabrias.html
just	Enayetpur.html
text	Enayetpur.html,Gabrias.html
Coordinates	Gabrias.html
produced	Enayetpur.html
sources	Enayetpur.html
geography	Gabrias.html
adding	Enayetpur.html
Recently	Enayetpur.html
also	Enayetpur.html
Navigation	Enayetpur.html,Gabrias.html
7EMonobook	Enayetpur.html,Gabrias.html
YUNUS	Enayetpur.html
Here	Enayetpur.html
thana	Enayetpur.html
to	Enayetpur.html,Gabrias.html
window	Enayetpur.html,Gabrias.html
Alaibot	Gabrias.html
Nederlands	Gabrias.html
rich	Enayetpur.html
under	Enayetpur.html,Gabrias.html
SieBot	Gabrias.html
main	Enayetpur.html,Gabrias.html
geographical	Gabrias.html
lacking	Enayetpur.html
Khukni	Enayetpur.html
LordAnubisBOT	Gabrias.html
material	Enayetpur.html
deductible	Enayetpur.html,Gabrias.html