Assignment 2
Introduction to Information Retrieval
CS734/834

John Berlin

October 12, 2016

Note

In order to make processing of the Wiki dataset timely and repeatable the util python file Source Code 8, was utilized to serialize the files full paths to disk in pickle format. Also included in this serialization was a python set of Wiki articles for both small and large datasets. This is to ensure that self links were not included when doing the calculations for section Q.2; also each question involving the Wiki dataset uses these file paths to read the html files for a given article.

The pickle formated produced when answering the questions about the Wiki dataset are included with this report in tar.gz format if it did not exceed the file size restrictions enforced by Github. For more information concerning this please read this post from Github. Graphv3 generated by section Q.2 is not included with this report as its size is 436mb compressed likewise the inverted index for wiki-large is not included as its size if 1.6gb uncompressed in pickle format.

Q.1 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?
```

Q.1 Answer

Processing of both the small and large Wiki articles was straight forward. Beautiful soup was used to extract the text from the articles as it removes the markup for you automatically. Each articles text was cleaned by replacing characters matched a regular expression looking for consecutive whitespace characters and punctuation (excluding punctuation that is included in valid words i.e *John's*) with a single space character. Tokenization of an articles text was done using the WordPunctTokenizer provided by the nltk library. This tokenizer preserves the punctuation of the text hence the need for removing prior to tokenization. Tokens produced in tokenization that had a length of one were excluded in the vocabulary calculations. Each token that had length greater than one was counted towards the word count and if the token was not in the vocabulary set it was added increasing the vocabulary count by one. The code for this is seen in Source Code 1.

It must be noted that the counts used in the plots are when a new vocabulary item is found. This is due to the size of the files when counting every word. The large data set produced a file of 1.4gb and the small produced a file sizes of 59mb. Table 1 shows the minute differences in the actual counts and that the size of the files is due to the repeated entries in them.

Table 1: Vocab Count Differences

Word Count	Vocab Count	File	Word Count	Vocab Count	File
4,090,585	232,028	Small Vocab	82,961,568	1,526,042	Large Vocab
4,090,629	232,028	Small Vocab Full	82,961,616	1,526,042	Large Vocab Full

Estimation of the parameters for heaps law and plotting was done in R and can be seen in Source Code 2. The parameters K and B were estimated using non-linear least squares (nls) method and both variables initially had values of one at the beginning of the calculation. Values produced by nls were plotted along side the values from the Wiki dataset on the y-axis as vocab count after using the predict method which is the goal of this question. I augmented the second part of this question by adding random order processing when building the vocabulary. This was to answer my own question about the order after plotting the in-order and reverse order plots.

Heaps approximation was close for both the small and large Figure 1. The real values followed the predicted for the first half then fluctuate for the remaining two quarters of the data set. The first fluctuation shows the point where the vocabulary grows slowly in comparison to the estimate. This plateau is due to the probability of

actually seeing new words this far into data set. The second fluctuation can is an increase above the estimation and can be attributed to the data set having more non-English text as Japanese articles about Hyundai are included at the end.

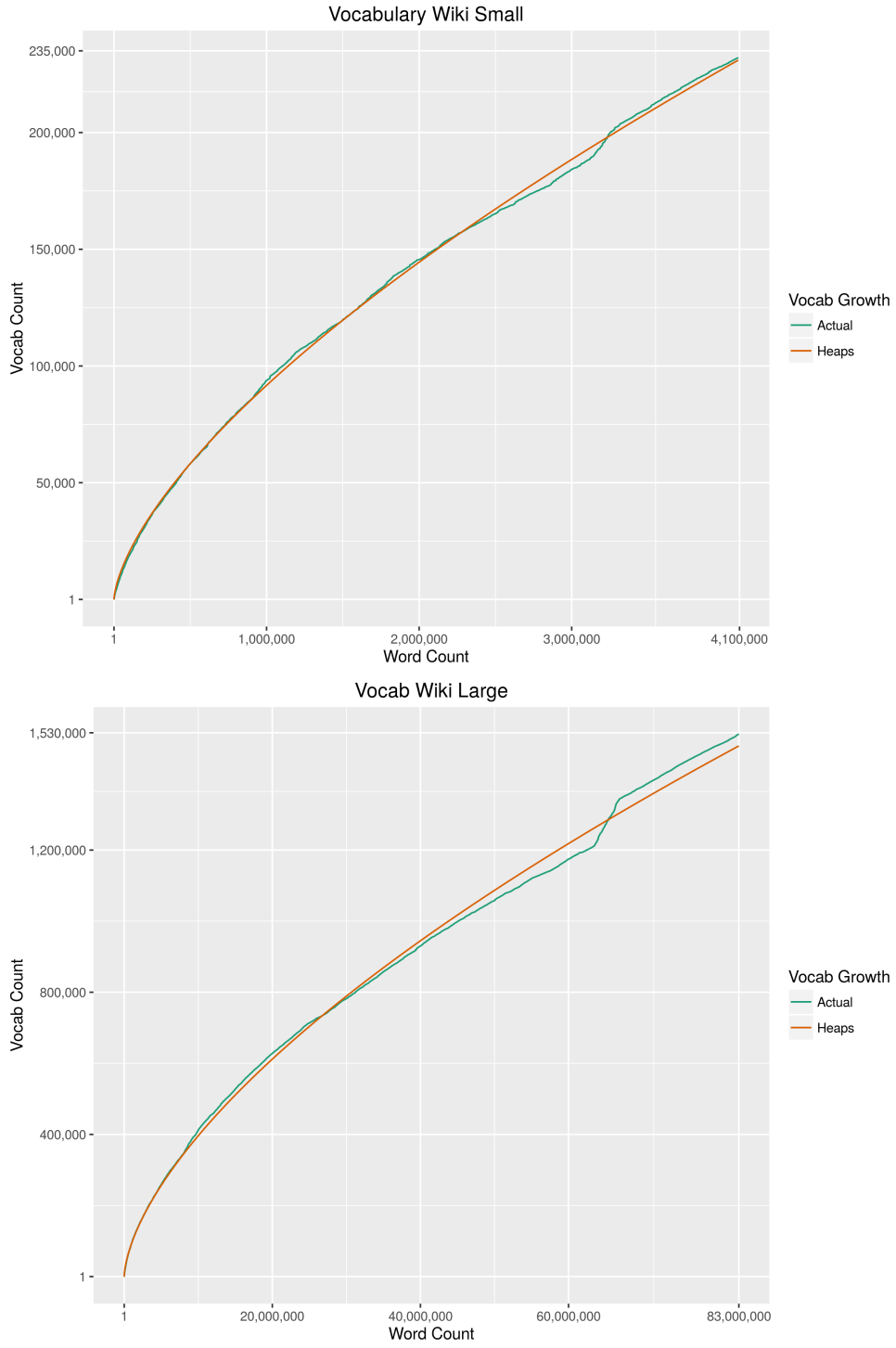

Figure 1: Vocab Growth

Processing the datasets in reverse order Figure 3 shows the reverse of the in order processing but aligns closer to Heaps estimation. As state in the discussion for the in order processing the actual data shows that we incur the spike of new words at the beginning rather than at then where it now levels out.

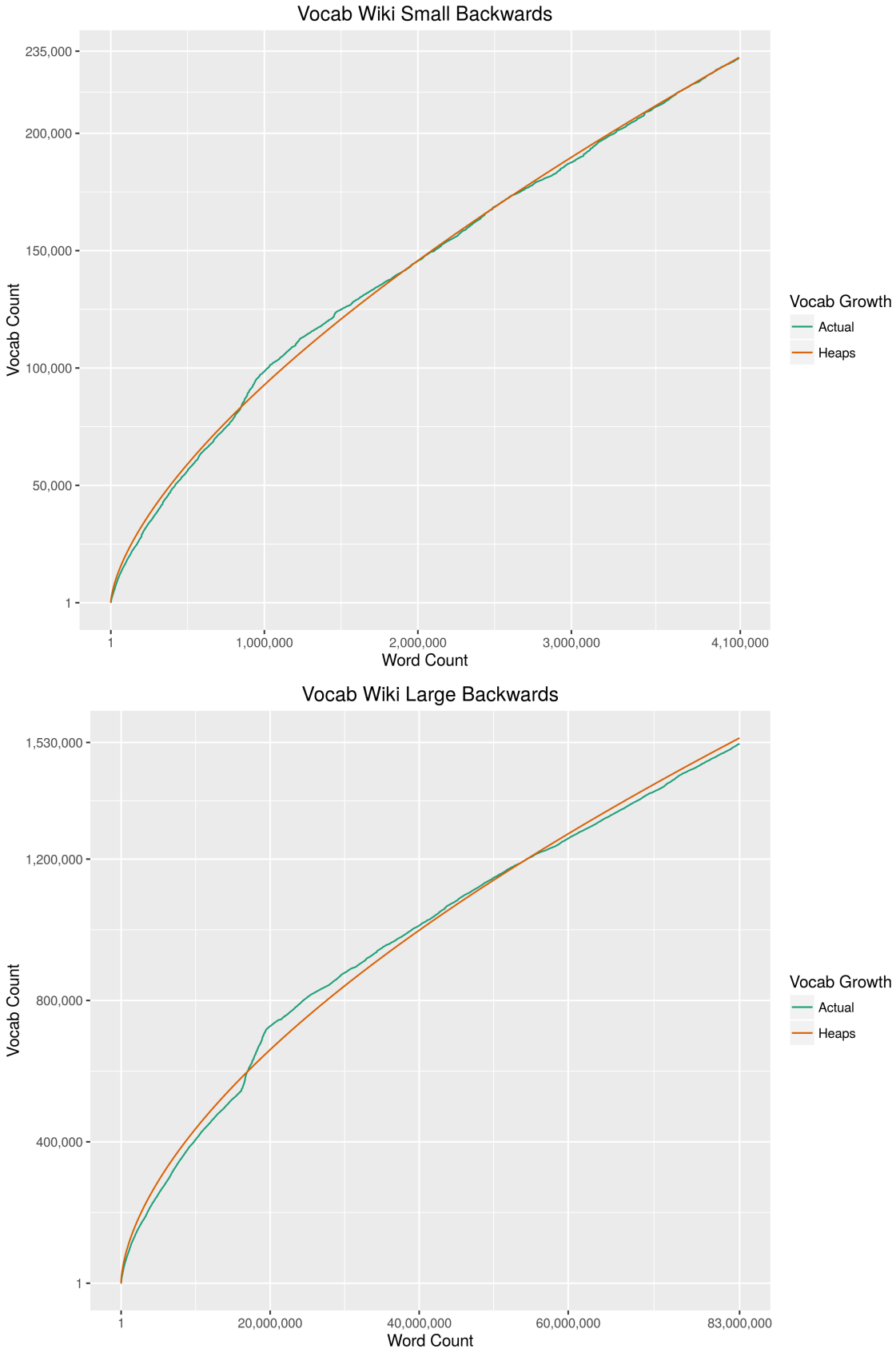

Figure 2: Vocab Growth Reverse Order

I choose to add the random order processing to this question as after plotting the dataset in order and reverse order as the plots caused me to wonder what its affect would be. Figure 3 shows that random selection for processing caused the actual data to be almost one to one with the estimate.

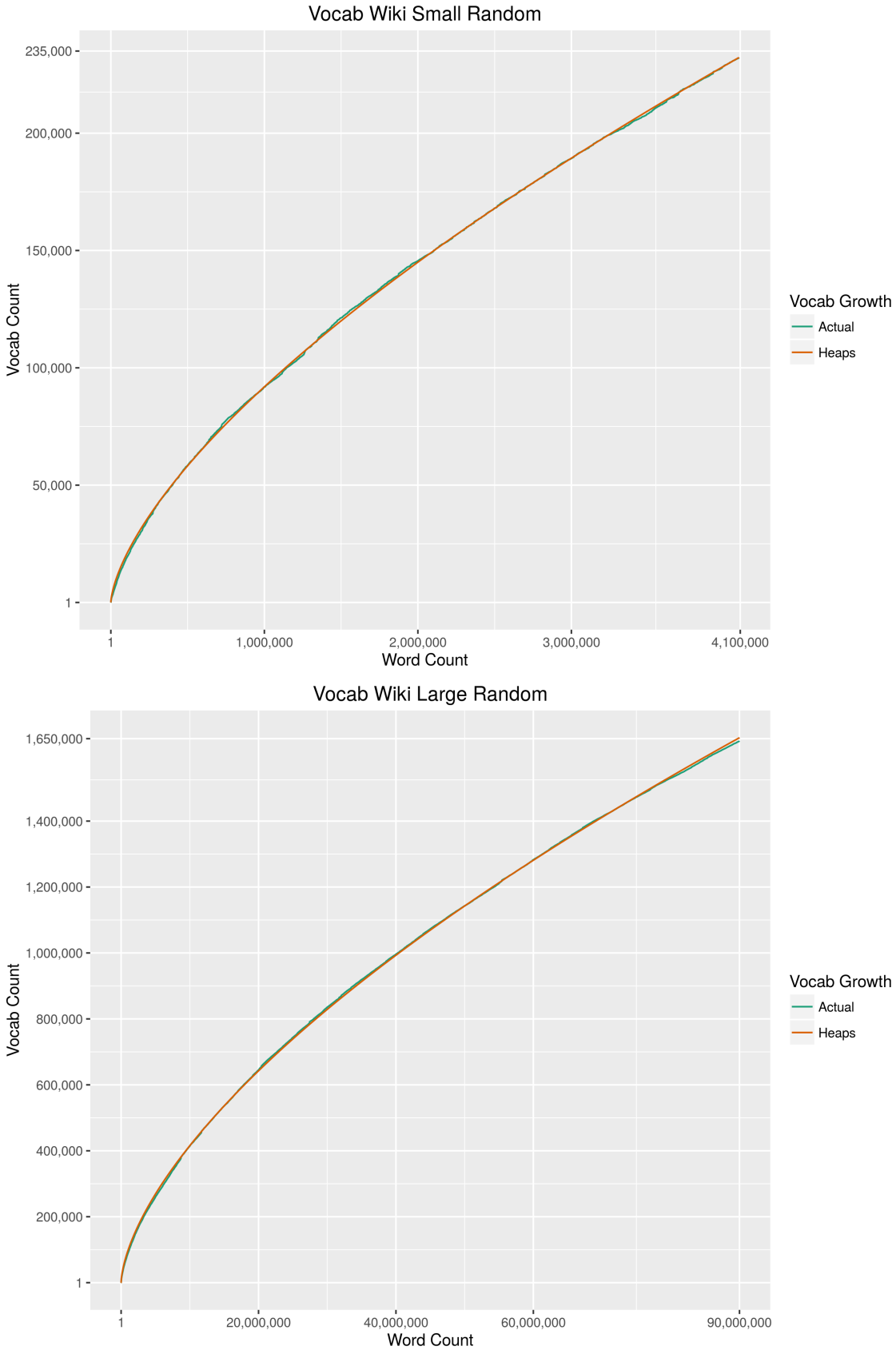

Figure 3: Vocab Growth Random Order

As shown in the plots for vocab processing, the order in which the they are seen does make a difference. The distribution of new words is primarily in the initial documents processed whereas more unique words are present towards the end of the documents. This was shown in Figure 1 and Figure 2 where reversing the order causes the fluctuation in new vocabulary words to happen at the beginning rather than at the end. Furthermore when

processing the documents in random order Figure 3 the distribution of new vocabulary entries to word count that would cause spikes in comparison to the estimation is negated. This negation, in our case with none English texts, shows that using Heaps estimation of vocabulary growth is an incredibly accurate irregardless of the mix of document languages in a dataset.

Source Code 1: Wiki Vocab Processing Code

```
import re
import random
from util import wsmall2, wlarge2, read_pickle, dump_pickle
from bs4 import BeautifulSoup
from nltk.tokenize import WordPunctTokenizer

no_wspace_punk = re.compile('(?:\s+)|[%s]' % re.escape("""!"#$%&()*+,./:;<=>?@[\]^_{|}~"""))


def vocab(wfile, outfile):
    w_list, w_set = read_pickle(wfile)
    toke = WordPunctTokenizer()
    vocab = set()
    vocab_count = 0
    word_count = 0
    with open(outfile, 'w+') as vout:
        vout.write('wc,vc\n')
        for wf in w_list:
            with open(wf, 'r') as wIn:
                wSoup = BeautifulSoup(wIn.read(), 'lxml')
                for token in toke.tokenize(no_wspace_punk.sub(' ', wSoup.text)):
                    if len(token) > 1:
                        word_count += 1
                        if token not in vocab:
                            vocab.add(token)
                            vocab_count += 1
                            out = '%d,%d\n' % (word_count, vocab_count)
                            vout.write(out)


def vocab_backwards(wfile, outfile):
    w_list, w_set = read_pickle(wfile)
    toke = WordPunctTokenizer()
    vocab = set()
    vocab_count = 0
    word_count = 0
    with open(outfile, 'w+') as vout:
        vout.write('wc,vc\n')
        for wf in reversed(w_list):
            with open(wf, 'r') as wIn:
                wSoup = BeautifulSoup(wIn.read(), 'lxml')
                for token in toke.tokenize(no_wspace_punk.sub(' ', wSoup.text)):
                    if len(token) > 1:
                        word_count += 1
                        if token not in vocab:
                            vocab.add(token)
                            vocab_count += 1
                            out = '%d,%d\n' % (word_count, vocab_count)
                            vout.write(out)


def vocab_random(wfile, outfile):
    w_list, w_set = read_pickle(wfile)
    random.shuffle(w_list)
    toke = WordPunctTokenizer()
```

```
    vocab = set()
    vocab_count = 0
    word_count = 0
    with open(outfile, 'w+') as vout:
        vout.write('wc,vc\n')
        for wf in w_list:
            with open(wf, 'r') as wIn:
                wSoup = BeautifulSoup(wIn.read(), 'lxml')
                for token in toke.tokenize(no_wspace_punk.sub(' ', wSoup.text)):
                    if len(token) > 1:
                        word_count += 1
                        if token not in vocab:
                            vocab.add(token)
                            vocab_count += 1
                            out = '%d,%d\n' % (word_count, vocab_count)
                            vout.write(out)


def vocab_small():
    vocab(wsmall2, 'output_files/wsmall-vocab.csv')
    vocab_backwards(wsmall2, 'output_files/wsmall-vocabB.csv')
    vocab_random(wsmall2, 'output_files/wsmall-vocabR.csv')


def vocab_large():
    vocab(wlarge2, 'output_files/wlarge-vocab2.csv')
    vocab_backwards(wlarge2, 'output_files/wlarge-vocabB.csv')
    vocab_random(wlarge2, 'output_files/wlarge-vocabR.csv')


if __name__ == '__main__':
    vocab_small()
    vocab_large()
```

Source Code 2: Wiki Vocab Plot Code

```
library(ggplot2)
library(RColorBrewer)
library(scales)

small <- function() {
  wsv <- read.csv('output_files/wsmall-vocab.csv')
  wsvb  <- read.csv('output_files/wsmall-vocabB.csv')
  wsvr  <- read.csv('output_files/wsmall-vocabR.csv')
  wsvb$c <- rep(c('Actual'),length(wsvb$wc))
  wsv$c <- rep(c('Actual'),length(wsv$wc))
  wsvr$c <- rep(c('Actual'),length(wsvr$wc))

  wsvNLS <- nls(vc ~ K * wc ^ B,data = wsv)
  wsvBNLS <- nls(vc ~ K * wc ^ B,data = wsvb)
  wsvRNLS <- nls(vc ~ K * wc ^ B,data = wsvr)
  ggplot(wsv,aes(x = wc, y = vc)) +
    geom_line(aes(colour=c)) +
    geom_line(data=data.frame(x=wsv$wc,y=predict(wsvNLS),c=rep(c('Heaps'),length(wsv$wc))),aes(x,y,colour=c)) +
    scale_x_continuous(breaks =c(1,1000000, 2000000,3000000, 4100000),labels = comma) +
    scale_y_continuous(breaks = c(1,50000,100000,150000,200000,235000),labels = comma) +
    scale_colour_brewer('Vocab Growth',palette='Dark2') +
    labs(title='Vocabulary Wiki Small',x = 'Word Count', y = 'Vocab Count')
  ggsave('wikiSmallVG2.png')
  ggplot(wsvb,aes(x = wc, y = vc)) +
    geom_line(aes(colour=c)) +
    geom_line(data=data.frame(x=wsvb$wc,y=predict(wsvBNLS),c=rep(c('Heaps'),length(wsvb$wc))),aes(x,y,colour=c)) +
    scale_x_continuous(breaks =c(1,1000000, 2000000,3000000, 4100000),labels = comma) +
    scale_y_continuous(breaks = c(1,50000,100000,150000,200000,235000),labels = comma) +
```

```
    scale_colour_brewer('Vocab Growth',palette='Dark2') +
    labs(title='Vocab Wiki Small Backwards',x = 'Word Count', y = 'Vocab Count')
  ggsave('wikiSmallVGB2.png')
  ggplot(wsvr,aes(x = wc, y = vc)) +
    geom_line(aes(colour=c)) +
    geom_line(data=data.frame(x=wsvr$wc,y=predict(wsvRNLS),c=rep(c('Heaps'),length(wsvr$wc))),aes(x,y,colour=c)) +
    scale_x_continuous(breaks =c(1,1000000, 2000000,3000000, 4100000),labels = comma) +
    scale_y_continuous(breaks = c(1,50000,100000,150000,200000,235000),labels = comma) +
    scale_colour_brewer('Vocab Growth',palette='Dark2') +
    labs(title='Vocab Wiki Small Random',x = 'Word Count', y = 'Vocab Count')
  ggsave('wikiSmallVR2.png')
}

large <- function() {
  wsv <- read.csv('output_files/wlarge-vocab.csv')
  wsvb  <- read.csv('output_files/wlarge-vocabB.csv')
  wsvr  <- read.csv('output_files/wlarge-vocabR.csv')
  wsvb$c <- rep(c('Actual'),length(wsvb$wc))
  wsv$c <- rep(c('Actual'),length(wsv$wc))
  wsvr$c <- rep(c('Actual'),length(wsvr$wc))

  wsvNLS <- nls(vc ~ K * wc ^ B,data = wsv)
  wsvBNLS <- nls(vc ~ K * wc ^ B,data = wsvb)
  wsvRNLS <- nls(vc ~ K * wc ^ B,data = wsvr)
  ggplot(wsv,aes(x = wc, y = vc)) +
    geom_line(aes(colour=c)) +
    geom_line(data=data.frame(x=wsv$wc,y=predict(wsvNLS),c=rep(c('Heaps'),length(wsv$wc))),aes(x,y,colour=c)) +
    scale_x_continuous(breaks =c(1,2e+07, 4e+07, 6e+07, 8.3e+07),labels = comma) +
    scale_y_continuous(breaks = c(1,400000,  800000, 1200000, 1530000),labels = comma) +
    scale_colour_brewer('Vocab Growth',palette='Dark2') +
    labs(title='Vocab Wiki Large',x = 'Word Count', y = 'Vocab Count')
  ggsave('wikiLargeVG2.png')
  ggplot(wsvb,aes(x = wc, y = vc)) +
    geom_line(aes(colour=c)) +
    geom_line(data=data.frame(x=wsvb$wc,y=predict(wsvBNLS),c=rep(c('Heaps'),length(wsvb$wc))),aes(x,y,colour=c)) +
    scale_x_continuous(breaks =c(1,2e+07, 4e+07, 6e+07, 8.3e+07),labels = comma) +
    scale_y_continuous(breaks = c(1,400000,  800000, 1200000, 1530000),labels = comma) +
    scale_colour_brewer('Vocab Growth',palette='Dark2') +
    labs(title='Vocab Wiki Large Backwards',x = 'Word Count', y = 'Vocab Count')
  ggsave('wikiLargeVGB2.png')
  ggplot(wsvr,aes(x = wc, y = vc)) +
    geom_line(aes(colour=c)) +
    geom_line(data=data.frame(x=wsvr$wc,y=predict(wsvRNLS),c=rep(c('Heaps'),length(wsvr$wc))),aes(x,y,colour=c)) +
    scale_x_continuous(breaks =c(1,2e+07, 4e+07, 6e+07, 8.3e+07),labels = comma) +
    scale_y_continuous(breaks = c(1,400000,  800000, 1200000, 1530000),labels = comma) +
    scale_colour_brewer('Vocab Growth',palette='Dark2') +
    labs(title='Vocab Wiki Large Random',x = 'Word Count', y = 'Vocab Count')
  ggsave('wikiLargeVGR2.png')
}

small()
large()
```

Q.2 Question 4.9

```
Compute PageRank for the Wikipedia documents. List the 20 documents
with the highest PageRank values together with the values.
```

Q.2 Answer

The computation of PageRank for both the small and large wiki sets was done using networkx a python "graph" library. It has three methods for computing PageRank: pure python, using scipi and numpy. I choose the scipi method as it uses power iteration on a sparse matrix for the eigenvector calculation rather than simply iteration (pure python) or accessing LAPACKs eigenvalue solver via numpy (which threw errors due to size of the graph). Out links were determined using Beautiful soup to find all the a tags in an article that were relative to the location of the dataset using the regular expression `'([.]{2}/){4}articles'`, filtering out image links and keeping out links that actually pointed to an article in the set. Two other variations were used called second and third. Both filtered out image links but the second only checked if the link was not a self link and the third used `Soup.find_all('a', href=True)` to find a tags with the other filtering methods from the first. It was found that some links use the $\sim$ character that pointed to pages not in the dataset but more times than not pointed to an actual page. To account for this the characters leftmost of the $\sim$ and itself were removed leaving the actual pointed to page which was checked to see if it was in the dataset. The code for this process can be seen in Source Code 2 and the sizes of the graph produces by each filtering method is seen in Table 2.

Table 2: Graph Sizes

Nodes	Edges	Method	Wiki Graph	Nodes	Edges	Method	Wiki Graph
6,043	608	Filter 1	Small	121,818	256,036	Filter 1	Large
233,493	471,722	Filter 2	Small	1,763,461	9,549,718	Filter 2	Large
233,493	471,722	Filter 3	Small	121,818	256,274	Filter 3	Large

The results of filtering method one for both the small and large Wiki dataset can be seen in Table 3. The ranks were not normalized as they show the raw data of the calculation given the sizes of the graphs. Method one for the small dataset produced a rather sparse graph whereas the large set produced a reasonable sized graph. The size of the small graph can only be attributed to the fact that in order to decrease the amount of articles included in the set while keeping some linkage between pages a wider variety had to used. The large graphs nodes UTC seemed odd to appear when using the first filtering method but indeed those pages do appear in the large dataset.

Filtering method 2 produced the largest graph for both the small and large set. Table 4 shows the effect of keeping the extraneous links in the computation. Pages such as About, SmackBot, Bluebot are included and appear in both graphs. Both set graphs are the same except for the order in which the nodes appear.

Filtering method 3 produced the most interesting results seen in Table 5. For the small set the number of nodes and edges remained the same from method 2 whereas the large sets edges increased by 38. What is also very interesting is the scores and pages are the same as was produced by method 1. The answer to why this happened may be found by considering the large sets graph. Its edges increase slightly leaving the PageRank results the same. This answer still leads me to believe human error for the small graph.

Table 3: Wiki Top 20 Page Rank

Wiki Small Page	rank	Wiki Large Page	rank
Brazil.html	0.01739	Record_label.html	0.00855
Fernando_Collor_de_Mello_1c42.html	0.01497	Music_genre.html	0.00812
August_26.html	0.00963	London.html	0.00767
Kidney.html	0.00620	UTC-5_c184.html	0.00551
Diabetic_nephropathy.html	0.00537	Brazil.html	0.00498
Seinfeld.html	0.00201	Radio.html	0.00469
Ronald_Colman_8c1a.html	0.00184	UTC+2_8d21.html	0.00460
Manga.html	0.00172	Paris.html	0.00460
Magazine.html	0.00166	1966.html	0.00383
Broderick_Crawford_c49a.html	0.00166	Romania.html	0.00329
1150.html	0.00154	The_Byrds_72ba.html	0.00306
Pope_Innocent_VIII_650a.html	0.00149	1962.html	0.00304
Transjordan.html	0.00147	1946.html	0.00300
Gilbert_and_Ellice_Islands_8a4e.html	0.00147	July_1.html	0.00291
Mollusca.html	0.00145	UTC+7∼30_a9b0.html	0.00282
Pope_Paul_VI_4529.html	0.00143	Herb_Alpert_5b74.html	0.00281
List_of_Navarrese_monarchs_334f.html	0.00140	UTC+5_6f4a.html	0.00258
F-102_Delta_Dagger_058e.html	0.00136	UTC-10_755d.html	0.00255
Isthmian_League_1d48.html	0.00136	April_15.html	0.00251
Pope_Benedict_IV_b5de.html	0.00135	1930.html	0.00240

Table 4: Wiki Top 20 Page Rank Method 2

Wiki Small Page	rank	Wiki Large Page	rank
Current_events_bb60.html	0.00050	Current_events_bb60.html	0.00130
RecentChanges_e0d0.html	0.00050	Community_Portal_6a3c.html	0.00130
Contact_us_afd6.html	0.00050	Contact_us_afd6.html	0.00130
Community_Portal_6a3c.html	0.00050	RecentChanges_e0d0.html	0.00130
About_8d82.html	0.00050	General_disclaimer_3e44.html	0.00130
Contents_22de.html	0.00050	Featured_content_5442.html	0.00130
General_disclaimer_3e44.html	0.00050	About_8d82.html	0.00130
Featured_content_5442.html	0.00050	Contents_22de.html	0.00130
Contents_b878.html	0.00050	Contents_b878.html	0.00130
Categories_101d.html	0.00049	Categories_101d.html	0.00129
Stub_72af.html	0.00017	Stub_72af.html	0.00044
SmackBot_cc7a.html	0.00012	SmackBot_cc7a.html	0.00032
Find_or_fix_a_stub_e7c5.html	0.00010	Find_or_fix_a_stub_e7c5.html	0.00024
Perfect_stub_article_2d8f.html	0.00009	Perfect_stub_article_2d8f.html	0.00023
Alaibot_de3d.html	0.00007	Alaibot_de3d.html	0.00019
Disambiguation_78fc.html	0.00005	United_States_09d4.html	0.00013
United_States_09d4.html	0.00005	Living_people_7259.html	0.00013
Living_people_7259.html	0.00005	Disambiguation_78fc.html	0.00013
Bluebot_e595.html	0.00004	Bluebot_e595.html	0.00011
Geographic_coordinate_system.html	0.00004	Geographic_coordinate_system.html	0.00011

Table 5: Wiki Top 20 Page Rank Method 3

Wiki Small Page	rank	Wiki Large Page	rank
Brazil.html	0.01739	Record_label.html	0.00855
Fernando_Collor_de_Mello_1c42.html	0.01497	Music_genre.html	0.00812
August_26.html	0.00963	London.html	0.00766
Kidney.html	0.00620	UTC-5_c184.html	0.00550
Diabetic_nephropathy.html	0.00537	Brazil.html	0.00495
Seinfeld.html	0.00201	Radio.html	0.00467
Ronald_Colman_8c1a.html	0.00184	Paris.html	0.00459
Manga.html	0.00172	UTC+2_8d21.html	0.00459
Magazine.html	0.00166	1966.html	0.00387
Broderick_Crawford_c49a.html	0.00166	Romania.html	0.00327
1150.html	0.00154	The_Byrds_72ba.html	0.00306
Pope_Innocent_VIII_650a.html	0.00149	1962.html	0.00303
Transjordan.html	0.00147	1946.html	0.00300
Gilbert_and_Ellice_Islands_8a4e.html	0.00147	July_1.html	0.00291
Mollusca.html	0.00145	UTC+7∼30_a9b0.html	0.00281
Pope_Paul_VI_4529.html	0.00143	Herb_Alpert_5b74.html	0.00281
List_of_Navarrese_monarchs_334f.html	0.00140	UTC+5_6f4a.html	0.00258
F-102_Delta_Dagger_058e.html	0.00136	UTC-10_755d.html	0.00254
Isthmian_League_1d48.html	0.00136	April_15.html	0.00252
Pope_Benedict_IV_b5de.html	0.00135	1930.html	0.00240

```
import re
import networkx as nx
from bs4 import BeautifulSoup
from urllib.parse import unquote
from util import *

local_wiki_re = re.compile('([.]{2}/){4}articles')

wiki_page_re = re.compile('(?:[.]{2}/){4}articles(?:/.){3}/(.+)')
wiki_http_page_re = re.compile('[a-z:/.]+(?:[.]{2}/){4}articles(?:/.){3}/(.+)')

nuke_tilda = re.compile('[A-Za-z]+~')


def no_image(a):
    return a.get('class') != ['image']


def get_edge_out(wfile, wfile_set):
    local_urls = []
    self_link = wfile[wfile.rfind('/') + 1:]
    with open(wfile, 'r') as wIn:
        wSoup = BeautifulSoup(wIn.read(), 'lxml')
        all_a = wSoup.find_all(href=local_wiki_re)
        for a in filter(no_image, all_a):
            furl = unquote(a['href'])
            rurl = nuke_tilda.sub('', furl[furl.rfind('/') + 1:])
            if rurl in wfile_set and rurl != self_link:
                local_urls.append((self_link, rurl))
    return self_link, local_urls


def get_edge_out2(wfile, nothing):
    local_urls = []
    self_link = wfile[wfile.rfind('/') + 1:]
    with open(wfile, 'r') as wIn:
        wSoup = BeautifulSoup(wIn.read(), 'lxml')
        all_a = wSoup.find_all(href=local_wiki_re)
        for a in filter(no_image, all_a):
            furl = unquote(a['href'])
            rurl = nuke_tilda.sub('', furl[furl.rfind('/') + 1:])
            if rurl != self_link:
                local_urls.append((self_link, rurl))
    return self_link, local_urls


def get_edge_out3(wfile, wfile_set):
    local_urls = []
    self_link = wfile[wfile.rfind('/') + 1:]
    with open(wfile, 'r') as wIn:
        wSoup = BeautifulSoup(wIn.read(), 'lxml')
        all_a = wSoup.find_all('a', href=True)
        for a in filter(no_image, all_a):
            href = unquote(a['href'])
            rurl = nuke_tilda.sub('', href[href.rfind('/') + 1:])
            if rurl in wfile_set and rurl != self_link:
                local_urls.append((self_link, rurl))
    return self_link, local_urls


def wiki_pagerank(w_list, w_set, edge_dump, graph_dump, edge_getter=get_edge_out):
    graph = nx.DiGraph()
    edge_list = []
```

```
    for wiki_file in w_list:
        self_link, out_links = edge_getter(wiki_file, w_set)
        graph.add_node(self_link)
        graph.add_edges_from(out_links)
        edge_list.append((self_link, out_links))
    dump_pickle(edge_list, edge_dump)
    w_pr = nx.pagerank_scipy(graph)
    dump_pickle((graph, w_pr), graph_dump)


def prout(wname):
    files = [('pickled/wiki-%s-graph.pickle' % wname, 'output_files/wiki-%s-pr20.csv' % wname),
             ('pickled/wiki-%s-graph2.pickle' % wname, 'output_files/wiki-%s-pr2-20.csv' % wname),
             ('pickled/wiki-%s-graph3.pickle' % wname, 'output_files/wiki-%s-pr3-20.csv' % wname)]
    for data, outfile in files:
        graph, pagerank = read_pickle(data)
        with open(outfile, 'w') as prOut:
            prOut.write('page,rank\n')
            for k, v in sorted(pagerank.items(), key=lambda x: x[1], reverse=True):
                prOut.write('%s,%.5f\n' % (k, v))


def wiki_small_pr():
    print('pr small')
    w_list, w_set = read_pickle(wsmall2)
    wiki_pagerank(w_list, w_set, wsmall_edges, wsmall_graph, get_edge_out)
    wiki_pagerank(w_list, w_set, wsmall_edges2, wsmall_graph2, get_edge_out2)
    wiki_pagerank(w_list, w_set, wsmall_edges3, wsmall_graph3, get_edge_out3)
    prout('small')


def wiki_large_pr():
    w_list, w_set = read_pickle(wlarge2)
    wiki_pagerank(w_list, w_set, wlarge_edges, wlarge_graph, get_edge_out)
    wiki_pagerank(w_list, w_set, wlarge_edges2, wlarge_graph2, get_edge_out2)
    wiki_pagerank(w_list, w_set, wlarge_edges3, wlarge_graph3, get_edge_out3)
    prout('large')


if __name__ == '__main__':
    wiki_small_pr()
    wiki_large_pr()
```

Source Code 2: Wiki PageRank Code

Q.3 Question 4.8

```
Find the 10 Wikipedia documents with the most inlinks. Show the collec-
tion of anchor text for those pages
```

Q.3 Answer

I used the graphs generated from filtering method one used in the answer to Q2 question 4.9. As seen in Table 6 the inlinks for the small set is much smaller in comparison to the large set. Due the vast size differences the counts for the number of anchor texts is shown and the full anchor texts is include in the output_files directory. The graphs were read in from the pickled format and in links for were gotten and sorted based on the length. The sorted list took the top ten and for each of the ten BeautifulSoup was used to extract the anchor text for all the links as seen in Source Code 4.

I will show a small section of the anchor texts for the top pages as a complete listing would not be presentable.

```
Brazil.html
Brazil (disambiguation)
Flag
Coat of arms
Motto
Anthem
Hino Nacional Brasileiro
National seal
Selo Nacional do Brasil
Capital
Brasília
Largest city
São Paulo
```

```
Music_genre.html
Electronic art music
Experimental music
Minimalist music
Jazz
Popular music
Traditional music
Folk music
Oral transmission
List of music genres
Art music
European classical music
List of classical music styles
```

Table 6: Wiki Top 10 In Links

Wiki Small Page	Anchor Text Count	Wiki Large Page	Anchor Text Count
Brazil.html	87	Music_genre.html	6030
August_26.html	25	Record_label.html	5844
Manga.html	14	London.html	2903
Magazine.html	13	UTC-5_c184.html	1543
Mollusca.html	12	Brazil.html	1535
Screenwriter.html	8	Paris.html	1516
Victoria_of_the_United_Kingdom_5e8e.html	8	UTC+2_8d21.html	1319
Kidney.html	6	Romania.html	1198
Tottenham_Hotspur_F.C._6bd2.html	5	1966.html	1188
Tuscany.html	4	1962.html	1127

```
import re
import networkx as nx
from bs4 import BeautifulSoup
from urllib.parse import unquote
from util import *

local_wiki_re = re.compile('([.]{2}/){4}articles')

wiki_page_re = re.compile('(?:[.]{2}/){4}articles(?:/.){3}/(.+)')
wiki_http_page_re = re.compile('[a-z:/.]+(?:[.]{2}/){4}articles(?:/.){3}/(.+)')

nuke_tilda = re.compile('[A-Za-z]+~')


def no_image(a):
    return a.get('class') != ['image']


def small():
    graph, pagerank = read_pickle(wsmall_graph)  # type: tuple[nx.DiGraph, dict]
    inlinks = []
    for n in graph.nodes():
        ins = graph.in_edges(n)
        if len(ins) > 0:
            inlinks.append((n, len(ins)))
    w_list, w_set = read_pickle(wsmall2)
    with open('output_files/wiki-small-inlinksc.csv', 'w+') as loc:
        with open('output_files/wiki-small-inlinks.csv', 'w+') as loc2:
            loc.write('page,count\n')
            loc2.write('page,atext\n')
            for n, c in sorted(inlinks, key=lambda x: x[1], reverse=True)[:10]:
                for fp in w_list:
                    wp = fp[fp.rfind('/') + 1:]
                    if n == wp:
                        print(fp, n, c)
                        loc.write('%s,%d\n' % (n, c))
                        at = []
                        with open(fp, 'r') as wIn:
                            wSoup = BeautifulSoup(wIn.read(), 'lxml')
                            all_a = wSoup.find_all(href=local_wiki_re)
                            for a in filter(no_image, all_a):
                                if a.string is not None:
                                    at.append(a.string)
                        loc2.write('%s,%s\n' % (n, ':'.join(at)))


def large():
    graph, pagerank = read_pickle(wlarge_graph)  # type: tuple[nx.DiGraph, dict]
    inlinks = []
    for n in graph.nodes():
        ins = graph.in_edges(n)
        if len(ins) > 0:
            inlinks.append((n, len(ins)))
    w_list, w_set = read_pickle(wlarge2)
    with open('output_files/wiki-large-inlinksc.csv', 'w+') as loc:
        with open('output_files/wiki-large-inlinks.csv', 'w+') as loc2:
            loc.write('page,count\n')
            loc2.write('page,atext\n')
            for n, c in sorted(inlinks, key=lambda x: x[1], reverse=True)[:10]:
                for fp in w_list:
                    wp = fp[fp.rfind('/') + 1:]
                    if n == wp:
                        print(fp, n, c)
```

```
                    loc.write('%s,%d\n' % (n, c))
                    at = []
                    with open(fp, 'r') as wIn:
                        wSoup = BeautifulSoup(wIn.read(), 'lxml')
                        all_a = wSoup.find_all(href=local_wiki_re)
                        for a in filter(no_image, all_a):
                            if a.string is not None:
                                at.append(a.string)
                    loc2.write('%s,%s\n' % (n, ':'.join(at)))


if __name__ == '__main__':
    small()
    print('----------------------------------------')
    large()
```

Source Code 4: Wiki Inlink Code

Q.4 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. (Dr. Nelson: use examples from the Wikipedia data set)
```

Q.4 Answer

As mentioned at the beginning of this report the size of the inverted index for the large set prohibits it from being included with this report and it follows the full contents would not be shown. The same goes for the small set but its file is included with this report as its size is only a mere 61.5mb. The size factor came into play as my initial implementation built the inverted index in memory using pythons defaultdict backed by a set. The initial implementation worked well for the small set but when I applied it to the large set, I found after letting it run for 45min+, that it used 11gb of memory and in combination with Chrome was about to cause kernel panic due to the swap space for Linux being used up entirely. Yes kernel panic due to memory consumption, I have had it happen before and it was not fun at all. To fix this issue the file and a single word were written to a file as they appeared and then was read from to build the inverted index. This brought down the memory usage for the large set to about 9gb. The tokenization method used for the Wiki vocabulary question was used here and the full code can be seen in Source Code 5. Table 7 shows the top 20 words and the number of documents found in.

Table 7: Wiki Inverted Index

Wiki Small Word	Number Of Article Found In	Wiki Large Word	Number Of Article Found In
wikimedia	6043	to	121818
articlediscussioncurrent	6043	by	121816
nonprofit	6043	revision	121816
gnu	6043	last	121816
contents	6043	wikipedia®	121816
help	6043	foundation	121816
if	6043	7emonobook	121816
wikipedia®	6043	of	121816
ismsie55	6043	skins	121816
gen	6043	under	121816
revision	6043	registered	121816
disclaimers	6043	s	121816
changes	6043	wikimedia	121816
search	6043	terms	121816
text	6043	7ecommon	121816
trademark	6043	contents	121816
registered	6043	featured	121816
encyclopedia	6043	this	121816
of	6043	gen	121816
inc	6043	encyclopedia	121816

```
import re
from collections import defaultdict
from util import wsmall2, wlarge2, read_pickle, dump_pickle
from bs4 import BeautifulSoup
from nltk.tokenize import WordPunctTokenizer

no_wspace_punk = re.compile('(?:\s+)|[%s]|•' % re.escape("""!"#$%&()*+,./:;<=>?@[\]^_{|}~"""))


def small_inverted_idx_helper():
    w_list, w_set = read_pickle(wsmall2)
    toke = WordPunctTokenizer()
    with open('output_files/wsmall-word-file.txt', 'w+') as out:
        for wf in w_list:
            fname = wf[wf.rfind('/') + 1:]
            with open(wf, 'r') as wIn:
                wSoup = BeautifulSoup(wIn.read(), 'lxml')
                for token in toke.tokenize(no_wspace_punk.sub(' ', wSoup.text)):
                    if len(token) > 1:
                        lt = token.lower()
                        out.write('%s %s\n' % (lt, fname))


def build_inverted_idx_small():
    small_inverted_idx_helper()
    inverted_index = defaultdict(set)
    with open('output_files/wsmall-word-file.txt', 'r') as wIn:
        for line in wIn:
            sline = line.rstrip().split(' ')
            inverted_index[sline[0]].add(sline[1])
    dump_pickle(inverted_index, 'pickled/wsmall-inverted-index.pickle')
    with open('output_files/wsmall-inverted-index-count.txt', 'w') as iidx2Out:
        with open('output_files/wsmall-inverted-index.txt', 'w') as iidxOut:
            for k, v in sorted(inverted_index.items(), key=lambda idxE: len(idxE[1]), reverse=True):
                files = ' '.join(v)
                iidxOut.write('%s\t%s\n' % (k, files))
                iidx2Out.write('%s\t%d\n' % (k, len(v)))


def large_inverted_idx_helper():
    w_list, w_set = read_pickle(wlarge2)
    toke = WordPunctTokenizer()
    with open('output_files/wlarge-word-file.txt', 'w+') as out:
        for wf in w_list:
            fname = wf[wf.rfind('/') + 1:]
            with open(wf, 'r') as wIn:
                wSoup = BeautifulSoup(wIn.read(), 'lxml')
                for token in toke.tokenize(no_wspace_punk.sub(' ', wSoup.text)):
                    if len(token) > 1:
                        lt = token.lower()
                        out.write('%s %s\n' % (lt, fname))


def build_inverted_idx_large():
    large_inverted_idx_helper()
    inverted_index = defaultdict(set)
    with open('output_files/wlarge-word-file.txt', 'r') as wIn:
        for line in wIn:
            sline = line.rstrip().split(' ')
            inverted_index[sline[0]].add(sline[1])
    with open('output_files/large-inverted-index-count.txt', 'w') as iidx2Out:
        with open('output_files/large-inverted-index.txt', 'w') as iidxOut:
            for k, v in sorted(inverted_index.items(), key=lambda idxE: len(idxE[1]), reverse=True):
```

```
                files = ' '.join(v)
                iidxOut.write('%s\t%s\n' % (k, files))
                iidx2Out.write('%s\t%d\n' % (k, len(v)))


if __name__ == '__main__':
    build_inverted_idx_small()
    build_inverted_idx_large()
```

Source Code 5: Wiki Inverted Index Code

Q.5 Question 5.14

```
In section 5.7.3, we saw that the optimal skip distance c can be determined
by minimizing the quantity kn/c + pc/2, where k is the skip pointer length, n
is the total inverted list size, c is the skip interval, and p is the number of postings
to find.
Plot this function using k = 4, n = 1,000,000, and p = 1,000, but varying c.
Then, plot the same function, but set p = 10,000. Notice how the optimal value
for c changes.
Finally, take the derivative of the function kn/c + pc/2 in terms of c to find
the optimum value for c for a given set of other parameters (k, n, and p).
```

Q.5 Answer

Figure 4 shows the plot for both p=1,000 and p=10,000. Please note that this plot has had as small amount of jitter applied to the points as both functions start out the same but diverge from each other quickly. This is due to the starting point for the function which is zero. The lower p value cause the function to grow rather slowly where as the large p value combined with the same c value the function grows faster.

The first derivative of the skip distance function in terms of c is $\frac{p}{2} - \frac{kn}{c^2}$ and was generated using the R code seen in Source Code 6. Finding the optimal c value for the function with p=1,000 and p=10,000 required me to do some thinking. After some thought I recalled what I had learned in the cs undergraduate numerical methods class here at ODU for root approximation of a function. The root approximation methods found the zeros of a function within a tolerance and the number which gave the zero is considered the functions root. This is much like what we want for this question. So I went looking for what would do this in R and discovered the grad function from the numDeriv package and from the documentation "The function grad calculates a numerical approximation of the first derivative of func at the point x". Using the grad function I generated the functions return values for a given c. The c values used for the two functions differ as after plotting various c values I found that for p=1,000 the c values from 10 to 200 showed the best results and for p=10,000 c values from 10 to 100. The optimal c value for the two functions was found by taking the maximum corresponding value whose approximation value was closest to or equal to zero. Figure 5 shows the plot for p=1,000 which had an optimal c value of 89 and Figure 6 shows the plot for p=10,000 which had an optimal c value of 28. The code used for the plots can be seen in Source Code 7.

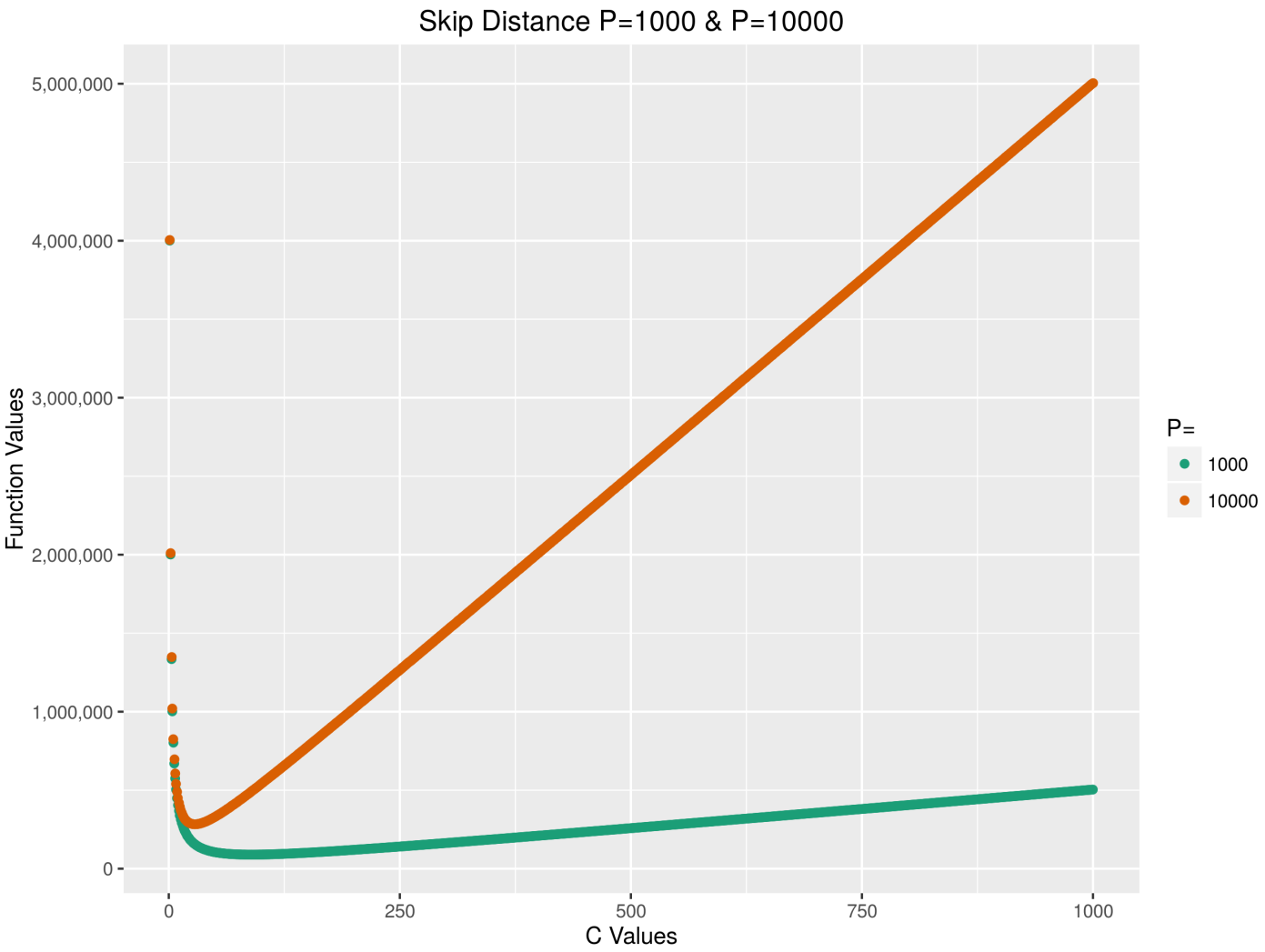

Figure 4: Skip Distance Optimal C Both P Plot

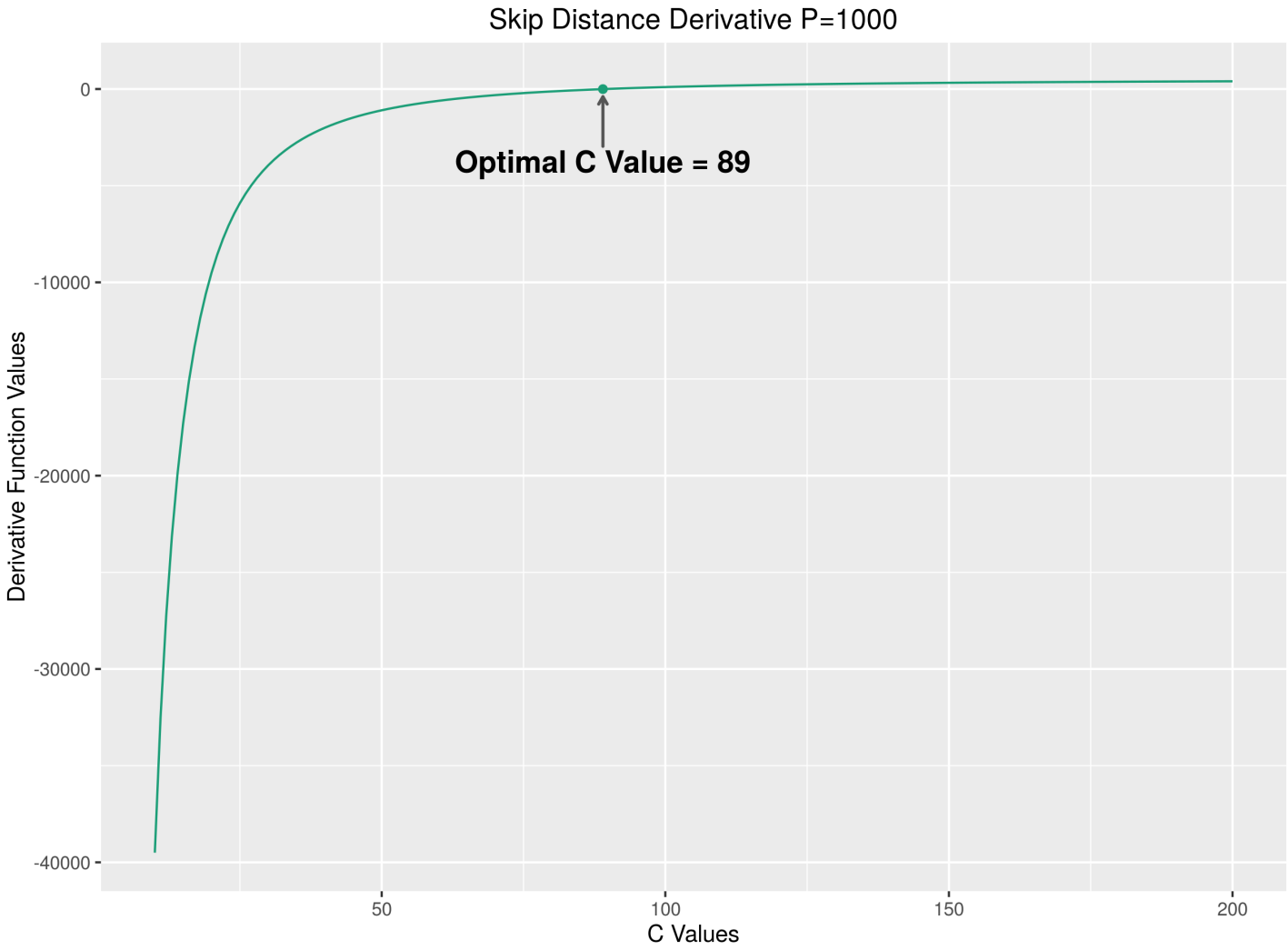

Figure 5: Skip Distance Derivative P=1000 Plot

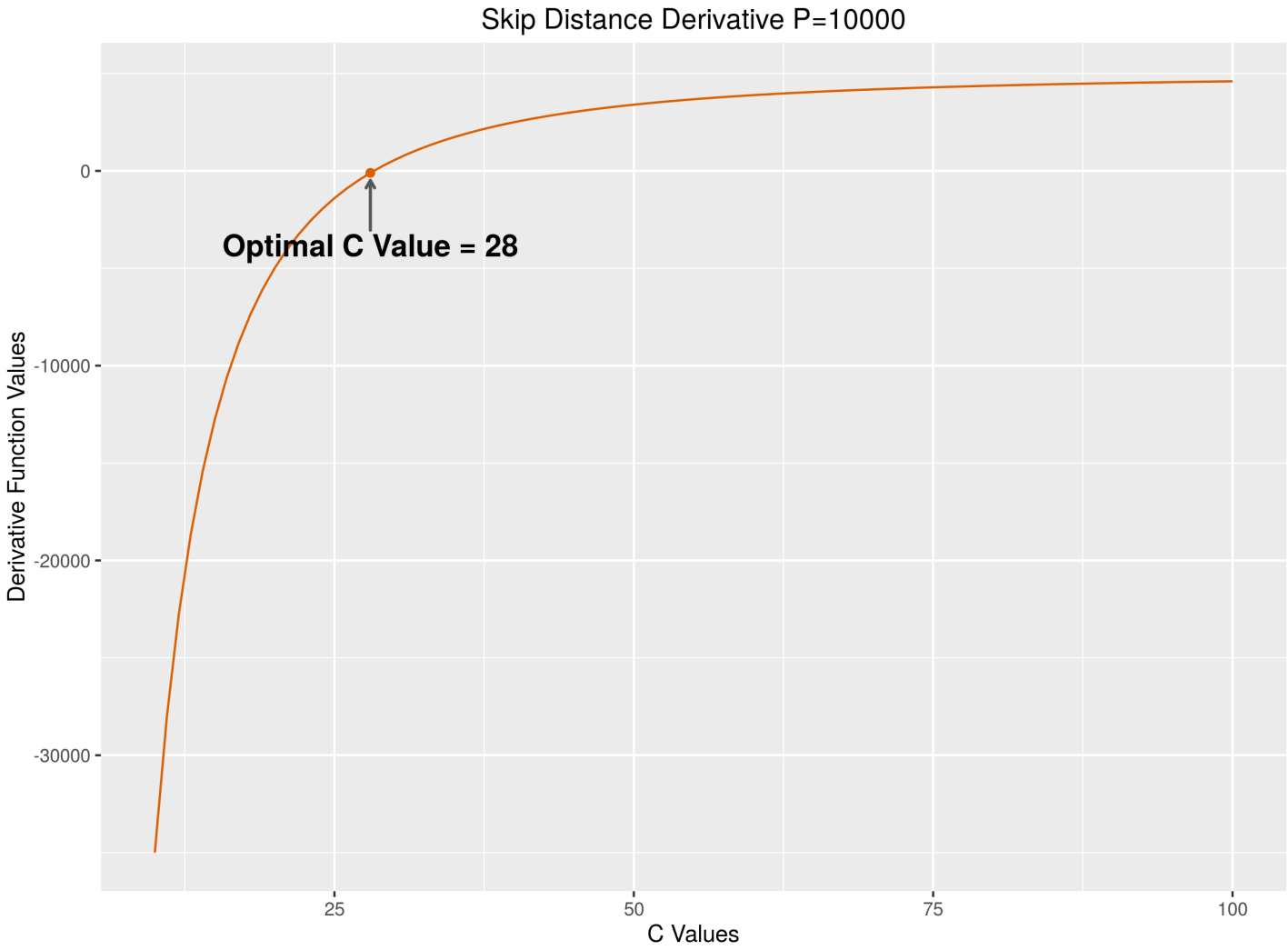

Figure 6: Skip Distance Derivative P=10000 Plot

```
library(Ryacas)
k <- Sym("k")
n <- Sym("n")
c <- Sym("c")
p <- Sym("p")

dc <- deriv((k*n/c)+(p*c/2),c)
PrettyForm(dc)
TeXForm(dc)
```

Source Code 6: First Derivative Of Skip Distance

```
library(ggplot2)
library(ggrepel)
library(ggthemes)
library(scales)
library(gridExtra)
library(RColorBrewer)
library(numDeriv)

colors <- brewer.pal(3, 'Dark2')

skip_distance1  <- function(c) { ((4 * 1000000) / c) + ((1000 * c) / 2) }
skip_distance2 <- function(c) {  ((4 * 1000000) / c) + ((10000 * c) / 2) }

x1 <- c(1:1000)
sd <- data.frame(
  x = c(x1,x1),
  y = c(
    sapply(x1,skip_distance1),
    sapply(x1,skip_distance2)
  ),
  P = c(
    rep(c('1000'),1000),
    rep(c('10000'),1000)
  )
)

ggplot(sd, aes(x,y)) +
  scale_y_continuous(labels = comma) +
  geom_point(aes(colour = P),position = position_jitter(width = 0.5, height = 0.5)) +
  scale_colour_brewer('P=',palette='Dark2') +
  labs(title='Skip Distance P=1000 & P=10000',x = 'C Values', y = 'Function Values')

ggsave('skipDistanceBoth.png')

skip_distanceD <- function(c) {(1000/2) - ((4 * 1000000)/c^2)}
skip_distanceD2 <- function(c) {(10000/2) - ((4 * 1000000)/c^2)}

xd <- c(10:200)
skipD <- data.frame(
  x = xd,
  y = grad(skip_distance1,xd)
)

xd2 <-c(10:100)
skipD2 <- data.frame(
  x = xd2,
  y = grad(skip_distance2,xd2)
)
```

```
skd_maxX_zero <- max(skipD[skipD$y<=0,])
skd2_maxX_zero <- max(skipD2[skipD2$y<=0,])

skd_oc <- skipD[skipD$x == skd_maxX_zero,]
skd_oc2 <- skipD2[skipD2$x == skd2_maxX_zero,]

ggplot(skipD, aes(x,y)) +
  geom_point( data=skd_oc,colour =colors[1]) +
  geom_line(colour =colors[1]) +
  geom_text_repel(data=skd_oc,aes(label = paste('Optimal C Value =',x)),
                  size = 5,
                  fontface = 'bold',
                  box.padding = unit(1.5, 'lines'),
                  point.padding = unit(0.5, 'lines'),
                  segment.color = '#555555',
                  segment.size = 0.7,
                  arrow = arrow(length = unit(0.01, 'npc')),
                  force = 1,
                  max.iter = 2e3) +
  labs(title='Skip Distance Derivative P=1000',x = 'C Values', y = 'Derivative Function Values')
ggsave('skipDistanceD.png')
ggplot(skipD2, aes(x,y)) +
  geom_point( data=skd_oc2,colour =colors[2]) +
  geom_line(colour =colors[2]) +
  geom_text_repel(data=skd_oc2,aes(label = paste('Optimal C Value =',x)),
                  size = 5,
                  fontface = 'bold',
                  box.padding = unit(1.5, 'lines'),
                  point.padding = unit(0.5, 'lines'),
                  segment.color = '#555555',
                  segment.size = 0.7,
                  arrow = arrow(length = unit(0.01, 'npc')),
                  force = 1,
                  max.iter = 2e3) +
  labs(title='Skip Distance Derivative P=10000',x = 'C Values', y = 'Derivative Function Values')
ggsave('skipDistanceD2.png')
```

Source Code 7: Skip Distance Plots

```
import pickle
from fs.osfs import OSFS

wlarge = 'pickled/wiki-large.pickle'
wlarge2 = 'pickled/wiki-large2.pickle'

wlarge_edges = 'pickled/wiki-large-sl.pickle'
wlarge_edges2 = 'pickled/wiki-large-edges2.pickle'
wlarge_edges3 = 'pickled/wiki-large-edges3.pickle'

wlarge_graph = 'pickled/wiki-large-graph.pickle'
wlarge_graph2 = 'pickled/wiki-large-graph2.pickle'
wlarge_graph3 = 'pickled/wiki-large-graph3.pickle'

wsmall = 'pickled/wiki-small.pickle'
wsmall2 = 'pickled/wiki-small2.pickle'

wsmall_edges = 'pickled/wiki-small-edges.pickle'
wsmall_edges2 = 'pickled/wiki-small-edges2.pickle'
wsmall_edges3 = 'pickled/wiki-small-edges3.pickle'

wsmall_graph = 'pickled/wiki-small-graph.pickle'
wsmall_graph2 = 'pickled/wiki-small-graph2.pickle'
wsmall_graph3 = 'pickled/wiki-small-graph3.pickle'


def dump_pickle(obj, file):
    with open(file, 'wb') as out:
        pickle.dump(obj, out)


def read_pickle(name):
    with open(name, "rb") as input_file:
        return pickle.load(input_file)


def pick_file_list():
    wl = OSFS('wiki-large')
    large_list = []
    large_set = set()
    for file in wl.walkfiles():
        large_list.append('wiki-large%s' % file)
        large_set.add(file[file.rfind('/') + 1:])
    dump_pickle((large_list, large_set), 'pickled/wiki-large2.pickle')
    wl.close()

    ws = OSFS('wiki-small')
    small_list = []
    small_set = set()
    for file in ws.walkfiles():
        small_list.append('wiki-small%s' % file)
        small_set.add(file[file.rfind('/') + 1:])
    dump_pickle((small_list, small_set), 'pickled/wiki-small2.pickle')
    ws.close()


if __name__ == '__main__':
    pick_file_list()
```

Source Code 8: Util File