Assignment 2
Introduction to Information Retrieval
CS734/834

John Berlin

October 11, 2016

Note

In order to make processing of the Wiki dataset timely and repeatable the util python file Source Code 3, 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 was necessary to ensure that self links were included when doing the calculations for section Q.2. The pickle format was used for the questions involving the Wiki dataset size allowing. Graphv3 generated by section Q.2 is not included with this report as Github does not like large files, 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

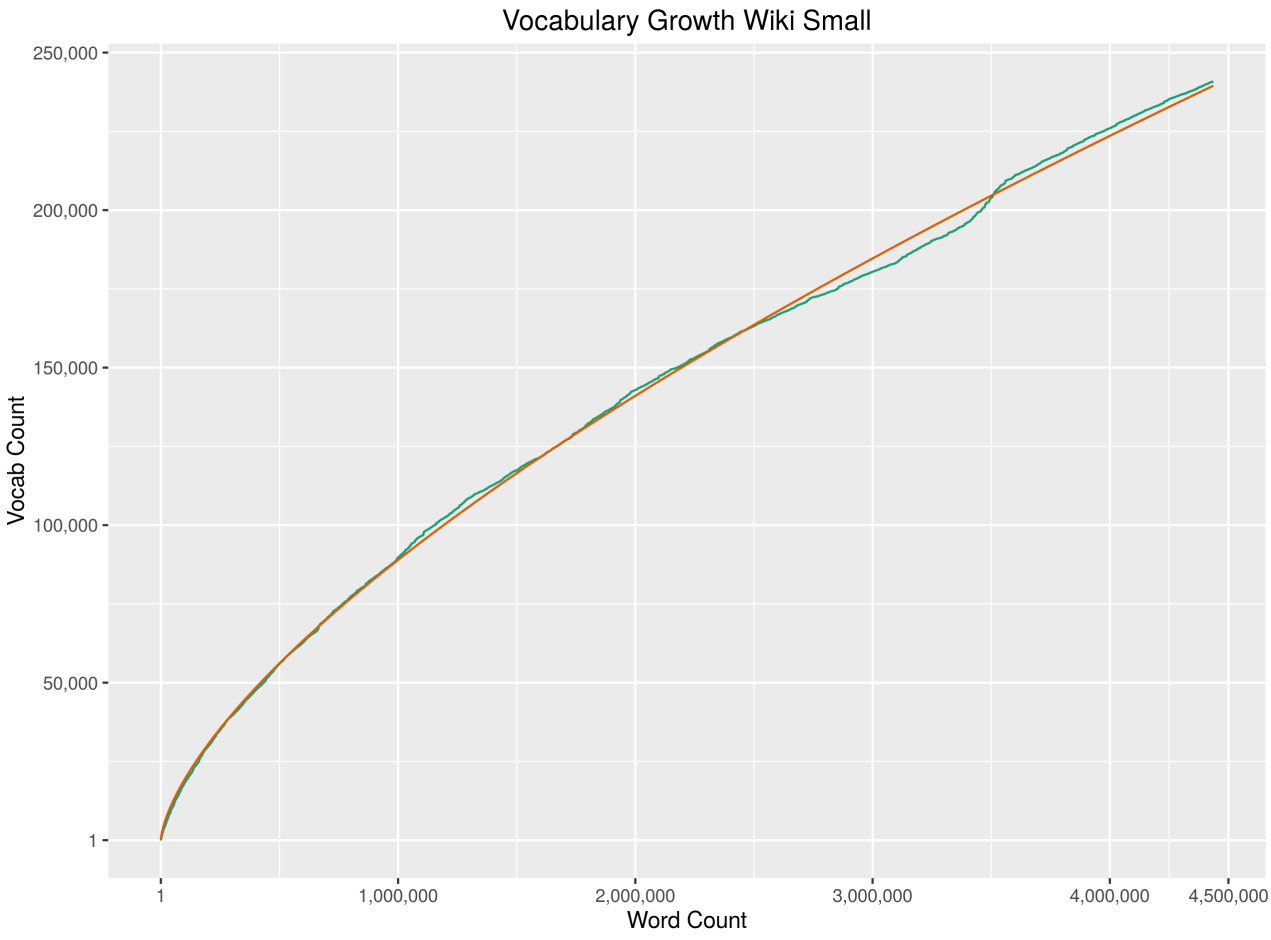

Figure 1: Wiki Small Vocab Growth

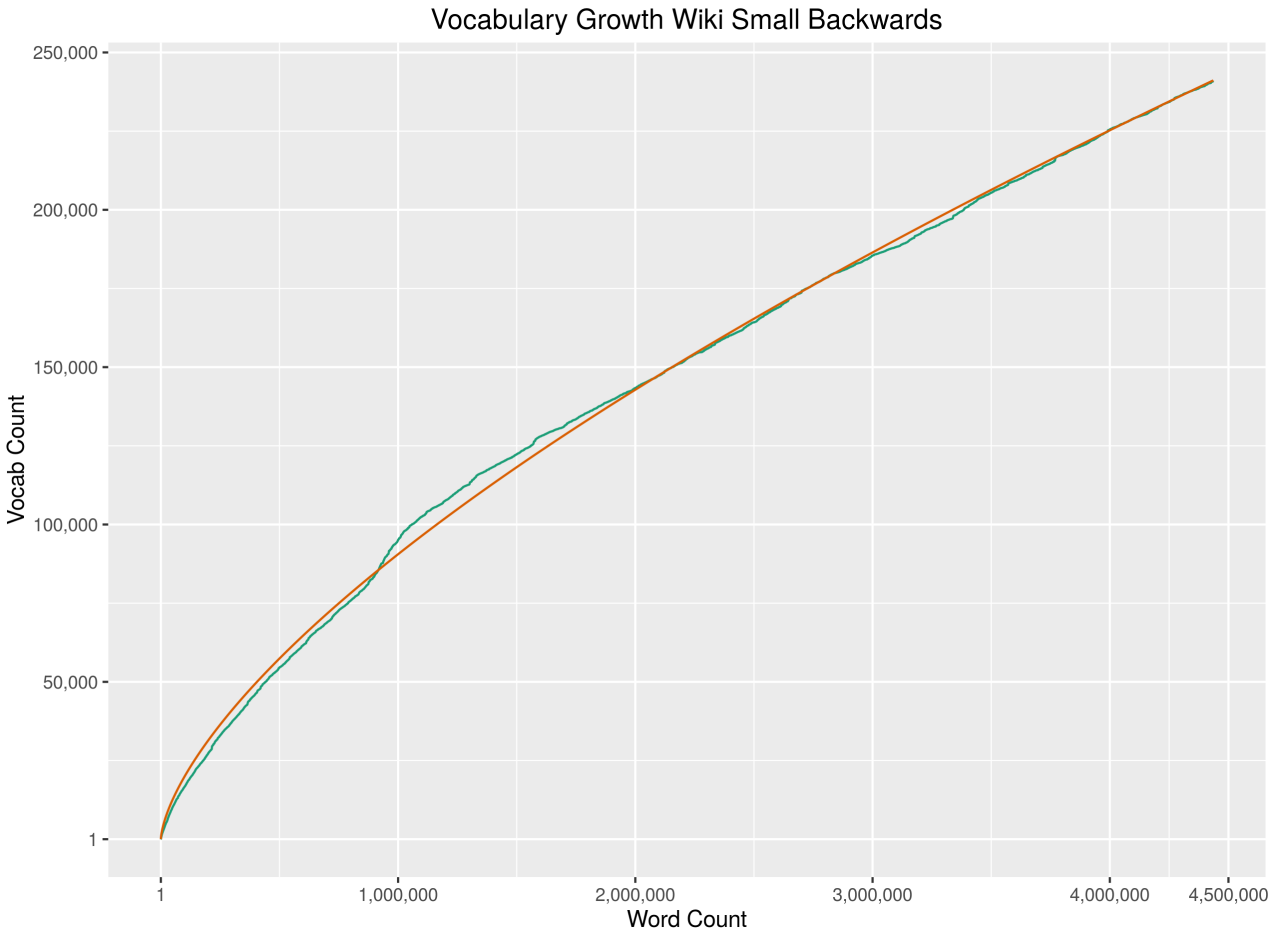

Figure 2: Wiki Small Vocab Growth Reverse Order

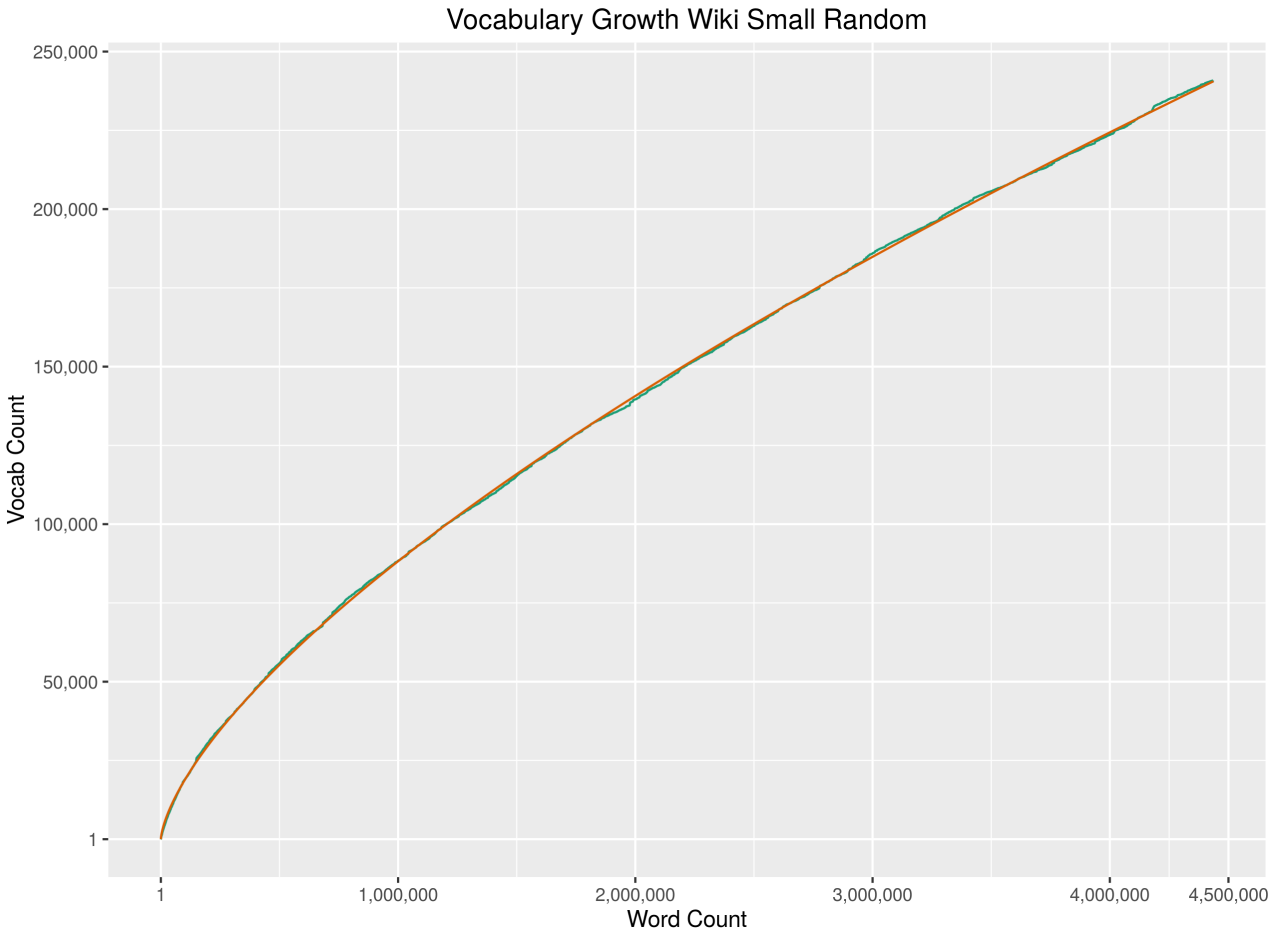

Figure 3: Wiki Small Vocab Growth Random Order

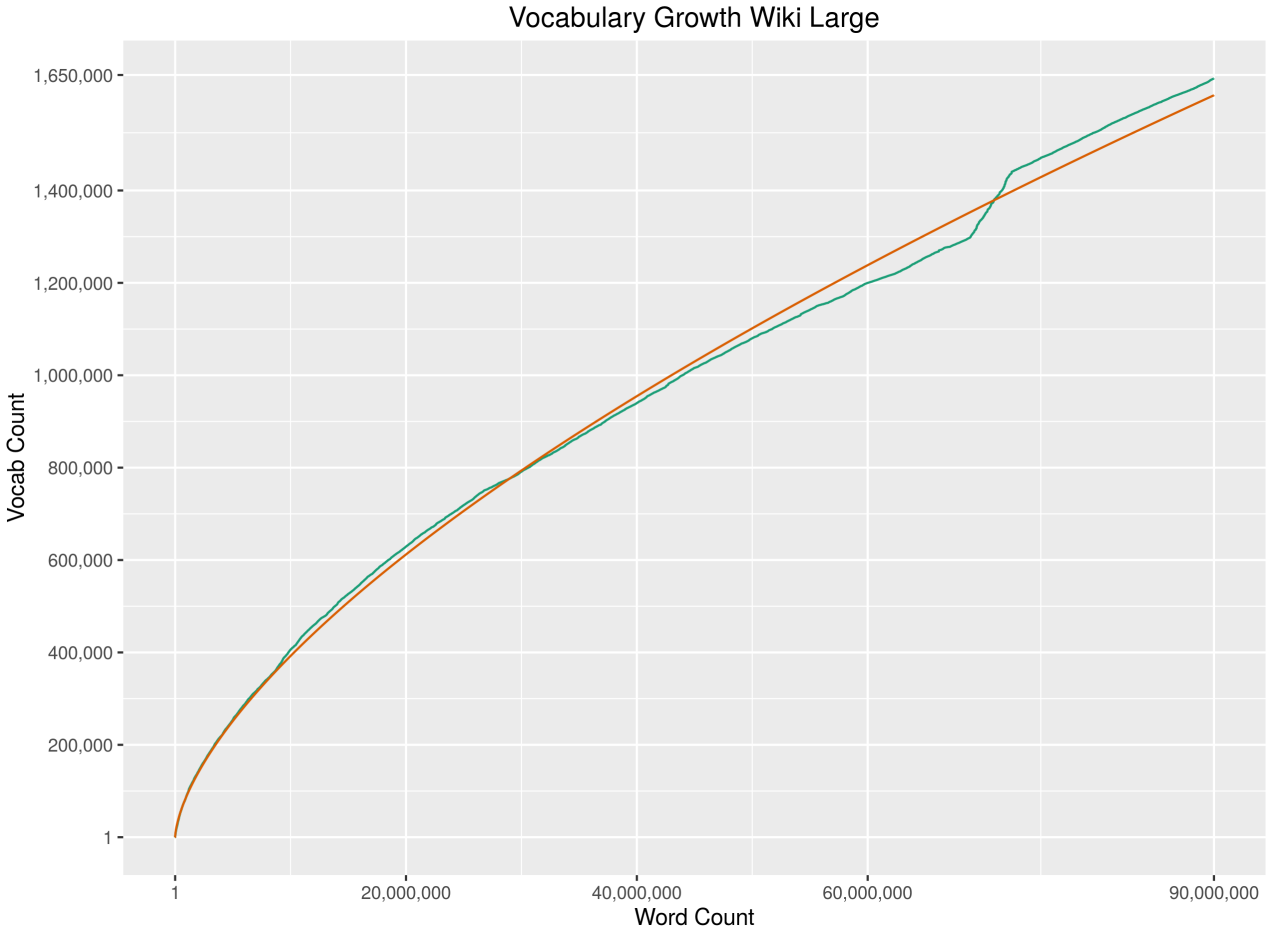

Figure 4: Wiki Large Vocab Growth

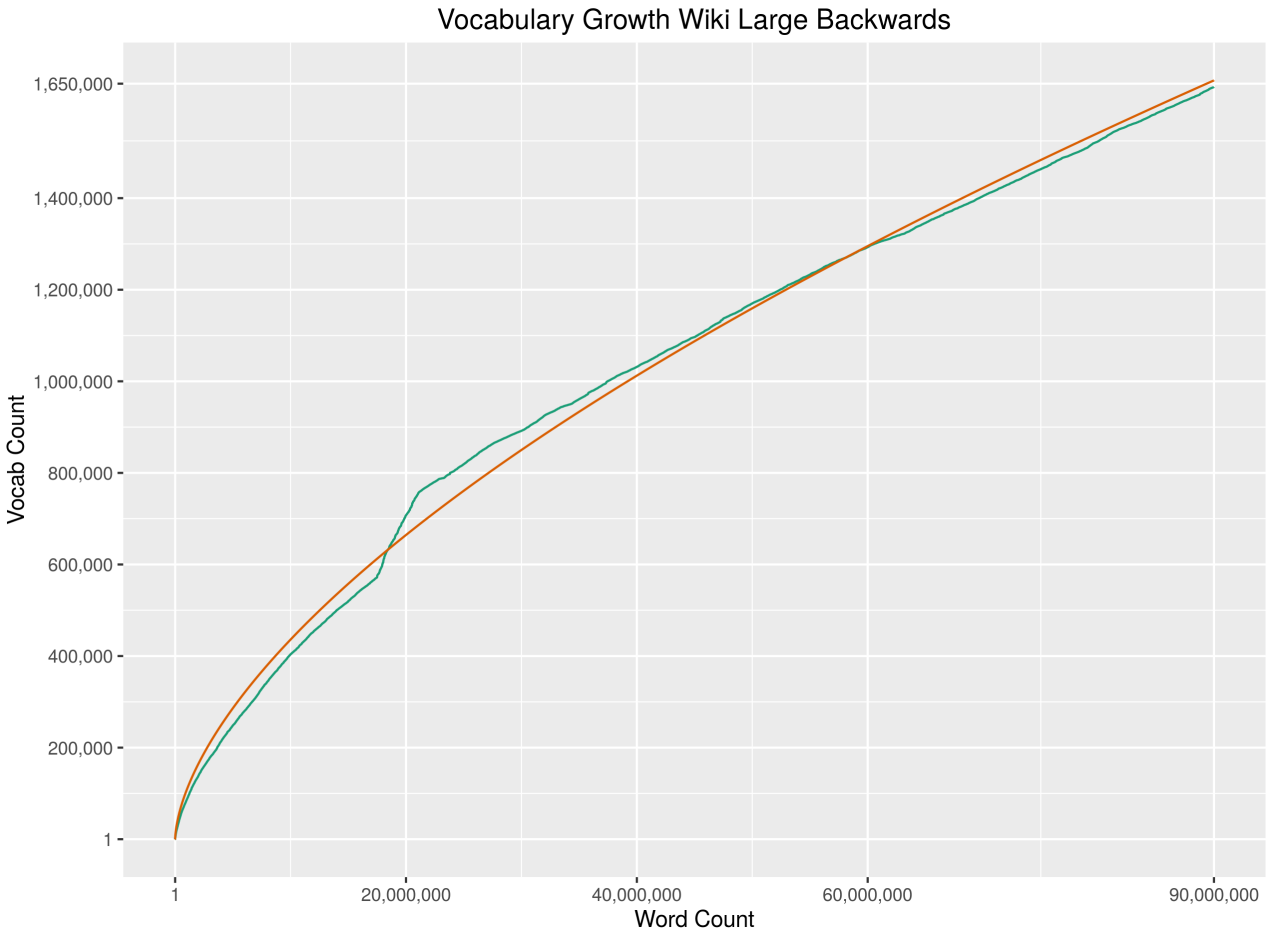

Figure 5: Wiki Large Vocab Growth Reverse Order

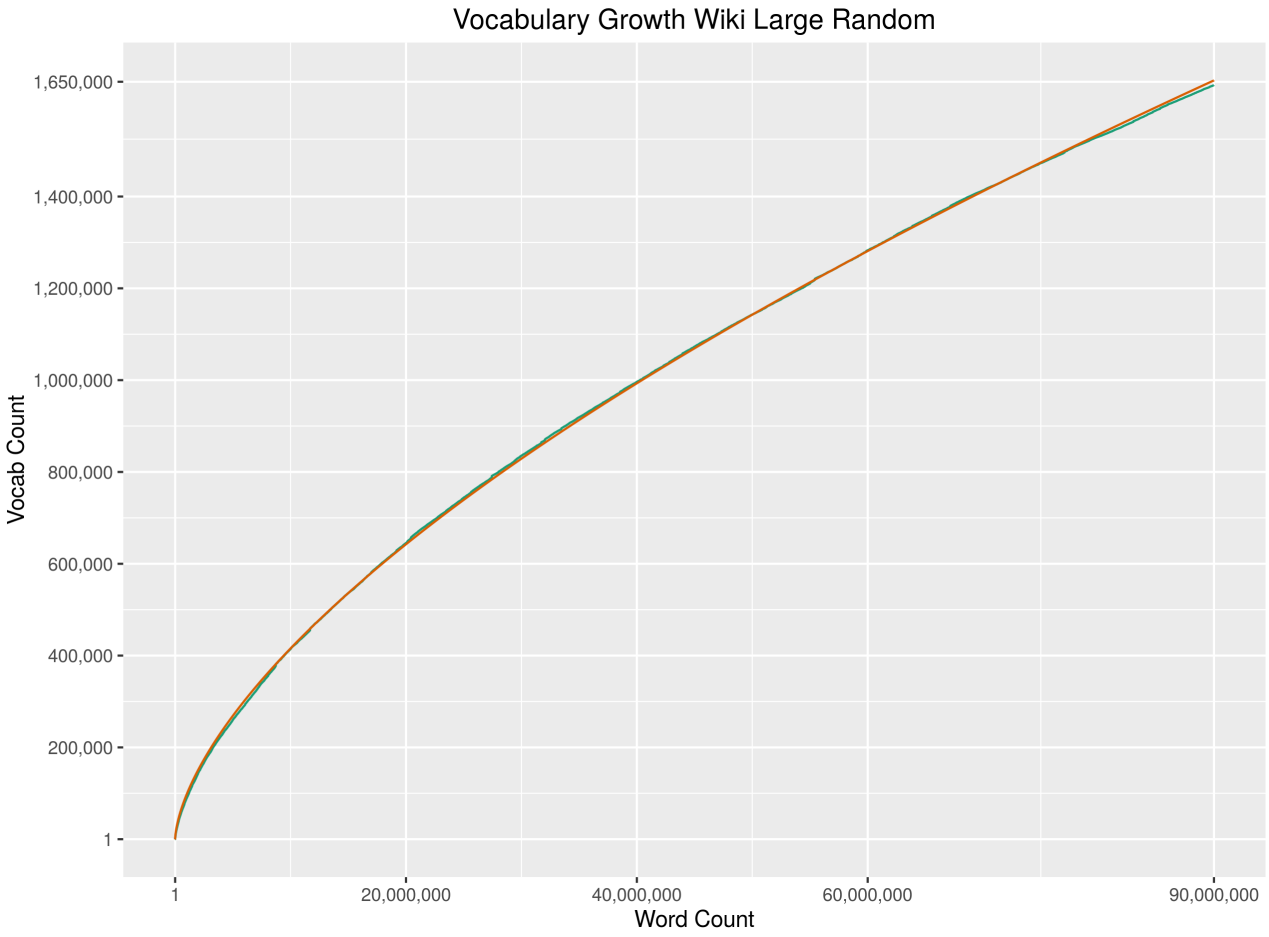

Figure 6: Wiki Large Vocab Growth Random Order

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

no_wspace_punk = re.compile('(?:\s+)|[%s]' % re.escape(punctuation))


def vocab(wfile, outfile):
    w_list, w_set = read_pickle(wfile)
    toke = TreebankWordTokenizer()
    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)):
                    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 = TreebankWordTokenizer()
    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)):
                    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 = TreebankWordTokenizer()
    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)):
```

```
                    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-vocab.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 1: Wiki Vocab Processing Code

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



small <- function() {
  colors <- brewer.pal(8, 'Dark2')
  wsv <- read.csv('output_files/wsmall-vocab.csv')
  wsvb  <- read.csv('output_files/wsmall-vocabB.csv')
  wsvr  <- read.csv('output_files/wsmall-vocabR.csv')

  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(color=colors[1]) +
    geom_line(data=data.frame(x=wsv$wc,y=predict(wsvNLS)),aes(x,y),color=colors[2]) +
    scale_x_continuous(breaks =c(1,1e+06,2e+06,3e+06,4e+06,4.5e+06),labels = comma) +
    scale_y_continuous(breaks = c(1,50000,100000,150000,200000,250000),labels = comma) +
    labs(title='Vocabulary Growth Wiki Small',x = 'Word Count', y = 'Vocab Count')
  ggsave('wikiSmallVG.png')
  ggplot(wsvb,aes(x = wc, y = vc)) +
    geom_line(color=colors[1]) +
    geom_line(data=data.frame(x=wsvb$wc,y=predict(wsvBNLS)),aes(x,y),color=colors[2]) +
    scale_x_continuous(breaks =c(1,1e+06,2e+06,3e+06,4e+06,4.5e+06),labels = comma) +
    scale_y_continuous(breaks = c(1,50000,100000,150000,200000,250000),labels = comma) +
    labs(title='Vocabulary Growth Wiki Small Backwards',x = 'Word Count', y = 'Vocab Count')
  ggsave('wikiSmallVGB.png')
  ggplot(wsvr,aes(x = wc, y = vc)) +
    geom_line(color=colors[1]) +
    geom_line(data=data.frame(x=wsvr$wc,y=predict(wsvRNLS)),aes(x,y),color=colors[2]) +
    scale_x_continuous(breaks =c(1,1e+06,2e+06,3e+06,4e+06,4.5e+06),labels = comma) +
    scale_y_continuous(breaks = c(1,50000,100000,150000,200000,250000),labels = comma) +
    labs(title='Vocabulary Growth Wiki Small Random',x = 'Word Count', y = 'Vocab Count')
  ggsave('wikiSmallVR.png')
}

large <- function() {
  colors <- brewer.pal(8, 'Dark2')
  wsv <- read.csv('output_files/wlarge-vocab.csv')
  wsvb  <- read.csv('output_files/wlarge-vocabB.csv')
  wsvr  <- read.csv('output_files/wlarge-vocabR.csv')

  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(color=colors[1]) +
    geom_line(data=data.frame(x=wsv$wc,y=predict(wsvNLS)),aes(x,y),color=colors[2]) +
    scale_x_continuous(breaks =c(1,2e+07, 4e+07, 6e+07,  9e+07),labels = comma) +
    scale_y_continuous(breaks = c(1,200000,  400000,  600000,  800000, 1000000, 1200000, 1400000,
    ↪  1650000),labels = comma) +
    labs(title='Vocabulary Growth Wiki Large',x = 'Word Count', y = 'Vocab Count')
  ggsave('wikiLargeVG.png')
  ggplot(wsvb,aes(x = wc, y = vc)) +
    geom_line(color=colors[1]) +
    geom_line(data=data.frame(x=wsvb$wc,y=predict(wsvBNLS)),aes(x,y),color=colors[2]) +
    scale_x_continuous(breaks =c(1,2e+07, 4e+07, 6e+07,  9e+07),labels = comma) +
    scale_y_continuous(breaks = c(1,200000,  400000,  600000,  800000, 1000000, 1200000, 1400000,
    ↪  1650000),labels = comma) +
    labs(title='Vocabulary Growth Wiki Large Backwards',x = 'Word Count', y = 'Vocab Count')
  ggsave('wikiLargeVGB.png')
  ggplot(wsvr,aes(x = wc, y = vc)) +
```

```
    geom_line(color=colors[1]) +
    geom_line(data=data.frame(x=wsvr$wc,y=predict(wsvRNLS)),aes(x,y),color=colors[2]) +
    scale_x_continuous(breaks =c(1,2e+07, 4e+07, 6e+07,  9e+07),labels = comma) +
    scale_y_continuous(breaks = c(1,200000,  400000,  600000,  800000, 1000000, 1200000, 1400000,
     ↪  1650000),labels = comma) +
    labs(title='Vocabulary Growth Wiki Large Random',x = 'Word Count', y = 'Vocab Count')
  ggsave('wikiLargeVGR.png')
}

small()
large()
```

Source Code 2: Wiki Vocab Plot Code

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

```
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 1: Wiki PageRank Code

Table 1: 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

```
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 collection of anchor text for those pages

Q.3 Answer

```
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 1: Wiki Inlink Code

Q.4 Question 5.8

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. (Dr. MLN: use examples from the Wikipedia data set)

Q.4 Answer

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

no_wspace_punk = re.compile('(?:\s+)|[%s]|•' % re.escape(punctuation))


def build_inverted_idx_small():
    inverted_index = defaultdict(set)
    w_list, w_set = read_pickle(wsmall2)
    toke = TreebankWordTokenizer()
    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)):
                lt = token.lower()
                inverted_index[lt].add(fname)
    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 = TreebankWordTokenizer()
    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)):
                    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])
    dump_pickle(inverted_index, 'pickled/wlarge-inverted-index.pickle')
    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 1: 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

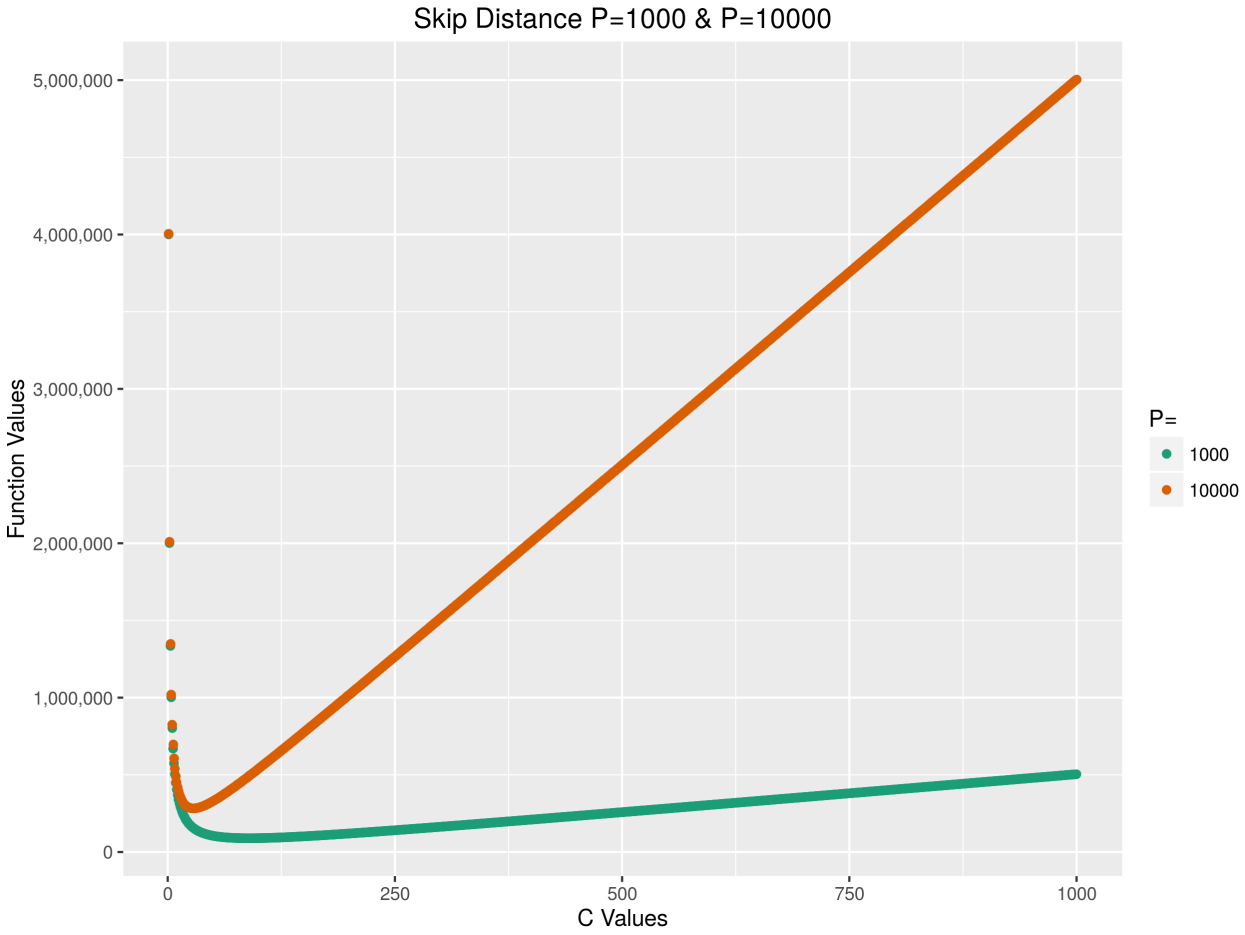

Figure 7: Skip Distance Optimal C Both P Plot

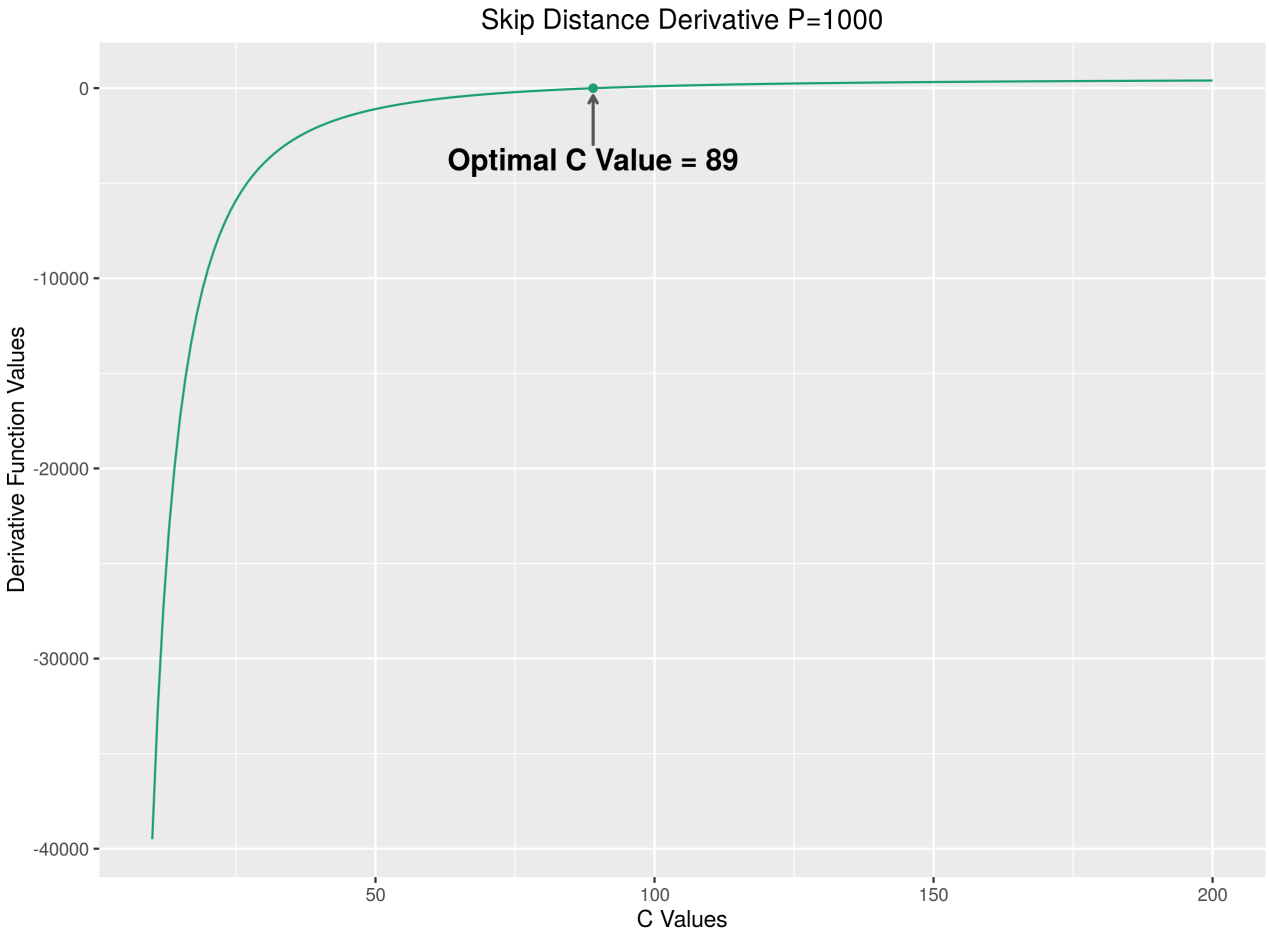

Figure 8: Skip Distance Derivative P=1000 Plot

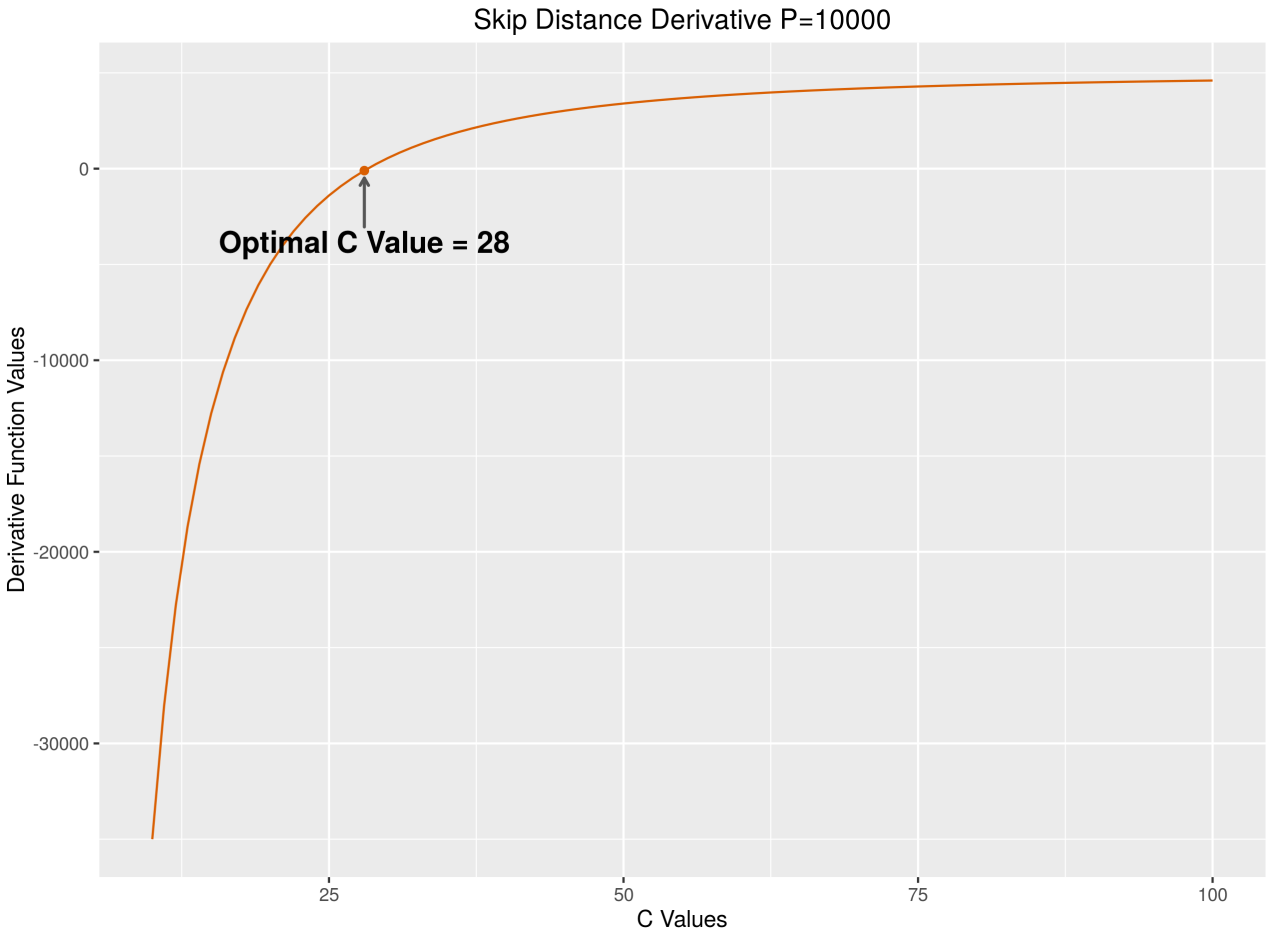

Figure 9: Skip Distance Derivative P=10000 Plot

```
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 1: 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 2: 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 3: Util File