CS 751: Assignment 3

Rohit Lambi

Spring 2015

Contents

1 Q1

1. For the text you saved for the 10000 URIs from A1, Q2: - Use the boilerpipe software to remove the HTML templates from all HTML pages (document how many pages link from the tweets were non-HTML and had to be skipped) - `https://code.google.com/p/boilerpipe/` - WSDM 2010 paper: `http://www.l3s.de/~kohlschuetter/boilerplate/`
2. For how many of the 10000 URIs was boilerpipe successful? - Compare the total words, unique words, and byte sizes before and after use of boilerpipe
3. For what classes of pages was it successful?
4. For what classes of pages was it unsuccessful?
5. Provide examples of both successful and unsuccessful removals and discuss at length.

1.1 Solution

1. I already had the HTML documents downloaded for the 10,000 URIs from A1, Q2.
2. I wrote a python program using boilerpipe library to fetch just the text from the 10,000 URIs.
3. At this point, I had two collections of documents which is stated above.
4. Out of 10000 URIs, boilerpipe worked for 5661 URIs.
5. It returned 404 for 1818 URIs and 2521 URIs had no content in the generated files.
6. It was successful in the cases where the webpage contain only HTML and no scripting embedded in it whereas it failed when some kind of scripting was present in the HTML page.
7. Examples of successful URIs:

   - `https://amp.twimg.com/v/24d799c3-48ba-4695-a2bb-9bafb4afc412`
   - `http://www.mirror.co.uk/news/uk-news/coalition-minister-centre-abuse-claims-5094966`
   - `http://exclaim.ca/Music/article/punch_brothers-phosphorescent_blues`

8. Examples of unsuccessful URIs:

   - `http://maneking.jp/index.html?ic=gn_twp_ltm2&af=75sf2xw12344558844`
   - `http://dorogobuzh.f-z-l.ru/dieta-babkinoy-nadezhdi-pevitsi-i-ee-menyu.php59`
   - `http://www.dunyabulteni.net/haberler/321299/sudanda-iki-rus-pilot-kacirildi?utm_source=dlvr.it&utm_medium=twitter62`

9. Observation on the number of words and byte size of both the document collections is that - the number of total words, unique words and byte size after boilerpipe is way too less as compared to the document collection which has HTML documents for all the URIs. The exact numbers are presented below.

1.2 Summary: Before boilerpipe/ HTML documents

Total Words: 35106673
Total Unique Words: 11041362
Byte Size: 914 MB

1.3 Summary: After boilerpipe/ Text from the webpages

Total Words: 1602010
Total Unique Words: 166391
Byte Size: 13 MB

1.4 Code Listing

fetchWebpages.py

```
'''
Created on Feb 8, 2015

@author: rlambi
'''
import subprocess
import os
import thread
import threading
import csv
import datetime

def fetchWebPage(url, fileName):
    print 'Fetching ', url
    #subprocess.Popen(["wget","-E","-H","-k","-K","-p", url])
    subprocess.Popen(["wget","--output-document=" + fileName, url]) # + ".html"

sites = 'sites'
if not os.path.exists(sites):
    os.makedirs(sites)

os.chdir(sites)

fieldNames = ['sno','seqNum','tcoUrl','url']

print datetime.datetime.now()
with open('../tweets-processed-1.txt') as csvfile:
    reader = csv.DictReader(csvfile, fieldnames=fieldNames, delimiter='\t')
    for row in reader:
        #fetchWebPage(row['url'])
        thread.start_new_thread(fetchWebPage, (row['url'], row['sno'], ))

print datetime.datetime.now()

print 'Waiting for all threads to complete'
while threading.activeCount() > 1:
    print str(threading.activeCount())
    pass

print 'Completed fetching all webpages'
print datetime.datetime.now()
```

Listing 1: Python program to fetch HTML documents for the 10,000 URIs of A1, Q2

extractTextWithBoilerpipe.py

```
import os
import thread
import threading
import csv
import datetime
from boilerpipe.extract import Extractor



iteration = '10'
BOILERPIPE_TEXT = 'boilerpipe_text/' + iteration
if not os.path.exists(BOILERPIPE_TEXT):
    print 'Creating folder - ' + BOILERPIPE_TEXT
    os.makedirs(BOILERPIPE_TEXT)


def fetchWebPage(URL):

    extractor = Extractor(url=URL)
    extracted_text = extractor.getText()
    return extracted_text


fieldNames = ['sno','seqNum','tcoUrl','url']

print datetime.datetime.now()

totalUrls = 0
skipCnt = 0
emptyContent = 0

skipped = open("skipped-" + iteration + ".txt", 'w')
with open(iteration + '.txt') as csvfile:
    reader = csv.DictReader(csvfile, fieldnames=fieldNames, delimiter='\t')

    for row in reader:

        totalUrls += 1
        try:
            print 'Fetching ', str(row['sno']), ' ', row['url']
            extracted_text = fetchWebPage(row['url'])

            if not extracted_text:
                                emptyContent += 1

            fil = open(BOILERPIPE_TEXT + '/' + row['sno'], 'w')
            fil.write(extracted_text.encode('UTF-8'))
            fil.close()

        except:
                        skipCnt += 1
                        skipped.write(row['sno'] + '\t' + row['url'] + '\n')

skipped.close()

print datetime.datetime.now()

summary = open("summary-" + iteration + ".txt", 'w')
summary.write("TotalUrls - " + str(totalUrls))
summary.write("\nSkipped - " + str(skipCnt))
summary.write("\nEmpty Content - " + str(emptyContent))
summary.close()

print 'Completed fetching all webpages'
print datetime.datetime.now()
```

Listing 2: Python program to fetch only text for the same URIs as above

2 Q2

1. Collection1: Extract all the unique terms and their frequency from the 10000 files*
2. Collection2: Extract all the unique terms and their frequency of the 10000 files* after running boilerpipe
3. Construct a table with the top 50 terms from each collection. - Find a common stop word list. How many of the 50 terms are on that stop word list?
4. For both collections, construct a graph with the x-axis as word rank, and y-axis as word frequency. - Do either follow a Zipf distribution? Support your answer.

2.1 Solution

1. I wrote two python programs to extract the unique terms and their frequency and saved it in text file.
2. In the collection1 there are 12 terms and in collection2 there are 41 terms that are from the stop word list. The stop word list is at the end of this document. The tables for both the collections with top 50 terms are shown below:

Table 1: Collection1: Word Frequency Table

Rank	Word	Word Frequency
1	div	1170684
2	a	538390
3	li	324226
4	span	307373
5	script	159971
6	ul	129377
7	span	120620
8	px	110074
9	lia	107868
10	the	103445
11	meta	100232
12	width	97620
13	var	97257
14	td	93394
15	to	86910
16	tr	86689
17	img	76837
18	height	74584
19	and	74201
20	p	66089
21	typetext	59865
22	javascript	59865
23	link	56247
24	in	55901
25	false	55637
26	of	54765
27	h	54099
28	if	52750
29	de	49659
30	button	42086
31	targetblank	42050
32	function	41691
33	class	41129
34	i	40408
35	for	39959
36	option	38933
37	typebutton	38543
38	input	37612
39	onclickreturn	33841
40	alt	32257
41	on	31724
42	is	29757
43	href	29362
44	with	28148
45	border	28143
46	br	27742
47	this	27409
48	value	25635
49	your	25497
50	color	25470

Table 2: Collection2: Word Frequency Table

Rank	Word	Word Frequency
1	the	41228
2	to	27058
3	a	23068
4	and	21496
5	of	17275
6	in	14449
7	is	11417
8	you	10493
9	for	8928
10	this	8494
11	play	8330
12	on	7955
13	your	7593
14	that	7545
15	it	7250
16	now	6905
17	que	6760
18	with	6335
19	i	5735
20	are	5187
21	y	5070
22	as	4928
23	be	4858
24	next	4632
25	by	4579
26	have	4355
27	or	4209
28	not	4119
29	from	3512
30	no	3486
31	more	3388
32	at	3301
33	has	3263
34	was	3191
35	but	3170
36	will	3101
37	an	3043
38	we	2961
39	can	2915
40	if	2764
41	all	2560
42	up	2402
43	they	2357
44	he	2284
45	new	2164
46	add	2151
47	been	2063
48	get	2049
49	about	2011
50	when	1991

2.2 Graphs

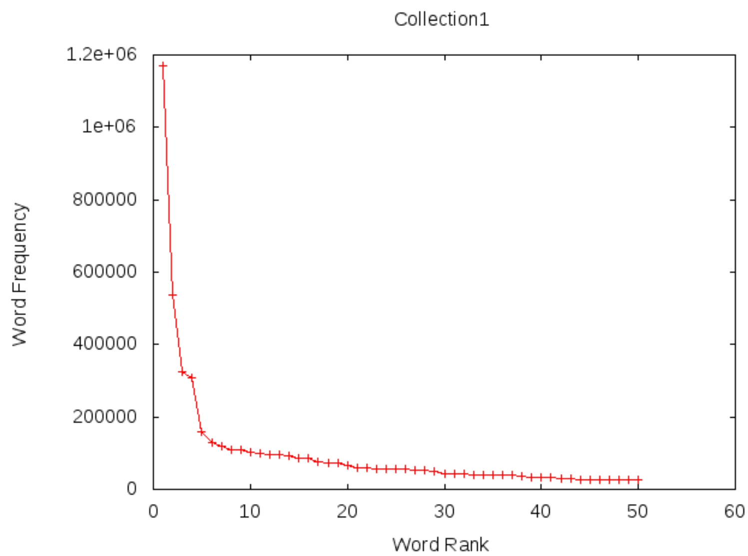

Figure 1: Graph for Collection1

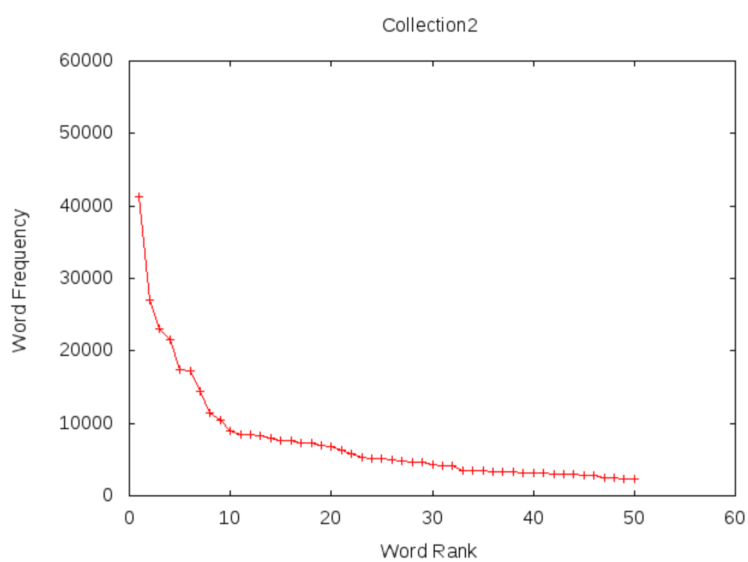

Figure 2: Graph for Collection2

Zipf law states that the frequency of any word is inversely proportional to its rank. In the above two graphs the value on Y axis decreases as the value on the X axis increases, which indicates that the frequency of the word is inversely proportional to its rank. Hence, both the collections follow Zipf distribution.

2.3 Code Listing

Collection 1: Python program to extract all the unique terms and their frequency from the 10,000 files generated by fetchWebpages.py

```
# Some of the code used is from http://code.google.com/edu/languages/google-python-class/

import sys
import re
import os
from os import path
from collections import OrderedDict

def sort_by_value(item):
    return item[-1]

def removePunctuation(word):

    word = re.sub('[\",():;?\\[\\].{}#$&_*+=%!<>~0-9]', '', word)
    return word


def build_dict(filename, count):
    f = open(filename, 'rU')
    words = f.read().split()

    for word in words:
        word = word.lower()
        word = word.strip()

        word = removePunctuation(word);

        if word not in count:
            count[word] = 1
        else:
            count[word] += 1

    f.close()

    return count

def write_words(PATH, dict):

    files = [f for f in os.listdir(PATH) if path.isfile(path.join(PATH, f))]
    #print files

    for f in files:
        filename = PATH + f
        dict = build_dict(filename, dict)




def main():

    totalWords = 0
    totalUniqueWords = 0
    dict = {}
    PATH = '/home/rlambi/rohit/Courses/Digital-Libraries/Assignments/A3/sites/'
    write_words(PATH, dict)

    odict = OrderedDict(sorted(dict.items(), key=lambda t: t[1]))

    fil = open('word-count.txt', 'w')
    odict.pop("\t", None)
    #for word in sorted(dict.keys()):
    for word in reversed(odict.keys()):

        freq = odict[word]

        if word is not None and freq is not None:
                if totalUniqueWords < 50000:
```

```
                        fil.write(word + '\t' + str(freq) + '\n')
                totalWords += freq
                totalUniqueWords += 1

    fil.close()

    print 'Total Words\t', str(totalWords), '\n'
    print 'Total Unique Words\t', str(totalUniqueWords)
    sys.exit(1)

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

Collection 2: Python program to extract all the unique terms and their frequency of the 10,000 files after running boilerpipe

```
# Some of the code used is from http://code.google.com/edu/languages/google-python-class/

import sys
import re
import os
from os import path

def sort_by_value(item):
    return item[-1]

def removePunctuation(word):

    word = re.sub('[\",():;?\\[\\].{}#$&_*+=%!<>~0-9]', '', word)
    return word


def build_dict(filename, count):
    f = open(filename, 'rU')
    words = f.read().split()

    for word in words:
        word = word.lower()
        word = word.strip()

        word = removePunctuation(word);

        if word not in count:
            count[word] = 1
        else:
            count[word] += 1

    f.close()

    return count

def write_words(PATH, dict):

    files = [f for f in os.listdir(PATH) if path.isfile(path.join(PATH, f))]
    #print files

    for f in files:
        filename = PATH + f
        dict = build_dict(filename, dict)




def main():

    dict = {}
    PATH = '/home/rlambi/rohit/Courses/Digital-Libraries/Assignments/A3/boilerpipe_text/1/'
    write_words(PATH, dict)

    PATH = '/home/rlambi/rohit/Courses/Digital-Libraries/Assignments/A3/boilerpipe_text/2/'
    write_words(PATH, dict)

    PATH = '/home/rlambi/rohit/Courses/Digital-Libraries/Assignments/A3/boilerpipe_text/3/'
    write_words(PATH, dict)

    PATH = '/home/rlambi/rohit/Courses/Digital-Libraries/Assignments/A3/boilerpipe_text/4/'
    write_words(PATH, dict)

    PATH = '/home/rlambi/rohit/Courses/Digital-Libraries/Assignments/A3/boilerpipe_text/5/'
    write_words(PATH, dict)

    PATH = '/home/rlambi/rohit/Courses/Digital-Libraries/Assignments/A3/boilerpipe_text/6/'
    write_words(PATH, dict)

    PATH = '/home/rlambi/rohit/Courses/Digital-Libraries/Assignments/A3/boilerpipe_text/7/'
```

```
    write_words(PATH, dict)

    PATH = '/home/rlambi/rohit/Courses/Digital-Libraries/Assignments/A3/boilerpipe_text/8/'
    write_words(PATH, dict)

    PATH = '/home/rlambi/rohit/Courses/Digital-Libraries/Assignments/A3/boilerpipe_text/9/'
    write_words(PATH, dict)

    PATH = '/home/rlambi/rohit/Courses/Digital-Libraries/Assignments/A3/boilerpipe_text/10/'
    write_words(PATH, dict)

    fil = open('word-count-boilerpipe.txt', 'w')
    dict.pop("\t", None)
    totalWords = 0
    totalUniqueWords = 0
    for word in sorted(dict.keys()):

        freq = dict[word]

        if word is not None and freq is not None:
                        fil.write(word + '\t' + str(dict[word]) + '\n')
                        totalWords += freq
                        totalUniqueWords += 1

    fil.close()

    print 'Total Words\t', str(totalWords), '\n'
    print 'Total Unique Words\t', str(totalUniqueWords)

    sys.exit(1)

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

Collection 1: Ruby program using gnuplot to plot the word frequency distribution graph

```
require "gnuplot"
require "csv"

Gnuplot.open do |gp|
  Gnuplot::Plot.new( gp ) do |plot|

    plot.terminal "png"
    plot.output File.expand_path("../plot_graph_c1.png", __FILE__)

    plot.xrange "[0:60]"
    plot.yrange "[0:1200000]"
    plot.title  "Collection1"
    plot.xlabel "Word Rank"
    plot.ylabel "Word Frequency"

    words = CSV.read('before-boilerpipe.csv')

    x,y = [], []
    words.each_with_index do |word, index|
      x += [word[0]]
      y += [word[1]]
    end
    plot.data << Gnuplot::DataSet.new( [x, y] ) do |ds|
      ds.with = "linespoints"
      ds.notitle
    end

  end
end
puts 'plot created'
```

Collection 2: Ruby program using gnuplot to plot the word frequency distribution graph

```ruby
require "gnuplot"
require "csv"

Gnuplot.open do |gp|
  Gnuplot::Plot.new( gp ) do |plot|

    plot.terminal "png"
    plot.output File.expand_path("../plot_graph_c2.png", __FILE__)

    plot.xrange "[0:60]"
    plot.yrange "[0:60000]"
    plot.title  "Collection2"
    plot.xlabel "Word Rank"
    plot.ylabel "Word Frequency"

    words = CSV.read('after-boilerpipe.csv')

    x,y = [], []
    words.each_with_index do |word, index|
      x += [word[0]]
      y += [word[1]]
    end
    plot.data << Gnuplot::DataSet.new( [x, y] ) do |ds|
      ds.with = "linespoints"
      ds.notitle
    end

  end
end
puts 'plot created'
```

2.4 Stopword List

a
able
about
above
abst
accordance
according
accordingly
across
act
actually
added
adj
affected
affecting
affects
after
afterwards
again
against
ah
all
almost
alone
along
already
also
although
always
am
among
amongst
an
and
announce
another
any
anybody
anyhow
anymore
anyone
anything
anyway
anyways
anywhere
apparently
approximately
are
aren
arent
arise
around
as
aside
ask
asking

at
auth
available
away
awfully
b
back
be
became
because
become
becomes
becoming
been
before
beforehand
begin
beginning
beginnings
begins
behind
being
believe
below
beside
besides
between
beyond
biol
both
brief
briefly
but
by
c
ca
came
can
cannot
can't
cause
causes
certain
certainly
co
com
come
comes
contain
containing
contains
could
couldnt
d
date
did
didn't

different
do
does
doesn't
doing
done
don't
down
downwards
due
during
e
each
ed
edu
effect
eg
eight
eighty
either
else
elsewhere
end
ending
enough
especially
et
et-al
etc
even
ever
every
everybody
everyone
everything
everywhere
ex
except
f
far
few
ff
fifth
first
five
fix
followed
following
follows
for
former
formerly
forth
found
four
from
further

furthermore
g
gave
get
gets
getting
give
given
gives
giving
go
goes
gone
got
gotten
h
had
happens
hardly
has
hasn't
have
haven't
having
he
hed
hence
her
here
hereafter
hereby
herein
heres
hereupon
hers
herself
hes
hi
hid
him
himself
his
hither
home
how
howbeit
however
hundred
i
id
ie
if
i'll
im
immediate
immediately
importance

important
in
inc
indeed
index
information
instead
into
invention
inward
is
isn't
it
itd
it'll
its
itself
i've
j
just
k
keep keeps
kept
kg
km
know
known
knows
l
largely
last
lately
later
latter
latterly
least
less
lest
let
lets
like
liked
likely
line
little
'll
look
looking
looks
ltd
m
made
mainly
make
makes
many
may

maybe
me
mean
means
meantime
meanwhile
merely
mg
might
million
miss
ml
more
moreover
most
mostly
mr
mrs
much
mug
must
my
myself
n
na
name
namely
nay
nd
near
nearly
necessarily
necessary
need
needs
neither
never
nevertheless
new
next
nine
ninety
no
nobody
non
none
nonetheless
noone
nor
normally
nos
not
noted
nothing
now
nowhere
o

obtain
obtained
obviously
of
off
often
oh
ok
okay
old
omitted
on
once
one
ones
only
onto
or
ord
other
others
otherwise
ought
our
ours
ourselves
out
outside
over
overall
owing
own
p
page
pages
part
particular
particularly
past
per
perhaps
placed
please
plus
poorly
possible
possibly
potentially
pp
predominantly
present
previously
primarily
probably
promptly
proud
provides

put
q
que
quickly
quite
qv
r
ran
rather
rd
re
readily
really
recent
recently
ref
refs
regarding
regardless
regards
related
relatively
research
respectively
resulted
resulting
results
right
run
s
said
same
saw
say
saying
says
sec
section
see
seeing
seem
seemed
seeming
seems
seen
self
selves
sent
seven
several
shall
she
shed
she'll
shes
should
shouldn't

show
showed
shown
showns
shows
significant
significantly
similar
similarly
since
six
slightly
so
some
somebody
somehow
someone
somethan
something
sometime
sometimes
somewhat
somewhere
soon
sorry
specifically
specified
specify
specifying
still
stop
strongly
sub
substantially
successfully
such
sufficiently
suggest
sup
sure

Bibliography

[1] Boilerpipe python library. https://pypi.python.org/pypi/boilerpipe.

[2] Draw line plot in gnuplot. https://github.com/rdp/ruby$_gnuplot$.

[3] Draw tables in LaTex. http://en.wikibooks.org/wiki/LaTeX/Tables.

[4] Stop word list. http://www.ranks.nl/stopwords.

[5] Zipf's law. http://en.wikipedia.org/wiki/Zipf's$_law$.