CS 851: Assignment #3

Due on Thursday, April 2, 2015

DR NELSON 4:20PM

VICTOR NWALA

Contents

Problem 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/
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
For what classes of pages was it successful?
For what classes of pages was it unsuccessful?
Provide examples of both successful and unsuccessful removals and discuss at length.
To answer this question, I started with 10000 links, only 910 of them were unique links, after I used justext to extract my text files only 53 links were successful.
I answered my question in a different way, I extracted my text files with justext, I took note of the successful links and inturn extracted the html pages from those links using wget.
The classes of pages that were succesful 200 OK pages, html pages. The classes of unsucessful pages were, 404 response pages, pages with all the different kinds of redirects i.e. 301, 302, etc.
The text files have 4025 unique words and a total of 27870 words and a combined size of 174.8kb. The html files have a total of 22931 unique words and a total of 63253 words and a combined size of 985.5kb. Note this for html files, it includes non-english characters.

Listing 1: Python script to extract text files from URIs using justext

```
import requests
import justext
import hashlib
from hashlib import md5
import os

fh = open("NonDup.txt",'r')
for line in fh:
     def computeMD5hash(message):
          m = hashlib.md5()
          m.update(message)
          return m.hexdigest()
     hashMessage = computeMD5hash(line)
     try:
               url=line
               response = requests.get(line)
               code = str(response.status_code)
               if code == '200':
                    paragraphs = justext.justext(response.content, justext.
                        get_stoplist("English"))
                    for paragraph in paragraphs:
                         if not paragraph.is_boilerplate:
                              if len(paragraph.text) > 1:
                                   saveFile = open("successUrl.txt",'a')
                                   saveFile.write(str(line))
                                   saveFile.write('\n')
                                   saveFile.close()

                    for paragraph in paragraphs:
```

Problem 1 continued on next page...

```
                        if not paragraph.is_boilerplate:
                            print paragraph.text
                            saveFile = open("/home/vnwala/File/"+hashMessage+".
                                txt",'a')
                            saveFile.write(paragraph.text.encode('utf-8') + '\n'
                                )
                            saveFile.close()

    except BaseException, e:
            print 'failed because,',str(e)
```

Listing 2: Python script to extract html files from URIs using wget

```
import hashlib
from hashlib import md5
import os

fh = open("sample.txt",'r')

for line in fh:
    url=line
    url=url.replace('\n','')

    def computeMD5hash(message):
        m = hashlib.md5()
        m.update(message)
        return m.hexdigest()


    hashMessage = computeMD5hash(url)


    os.system("lynx -dump -force_html " + url+ "  > /home/vnwala/TEXT/" +
        hashMessage  +".processed"+ ".txt ")
```

Listing 3: Python script to eliminate duplicate URIs

```


lines_seen = set() # holds lines already seen
outfile = open("sample.txt", "w")
for line in open("successUrl.txt", "r"):
    if line not in lines_seen: # not a duplicate
        outfile.write(line)
        lines_seen.add(line)
outfile.close()
```

Listing 4: Python script to combine text files into one file

```
import glob
```

```
read_files = glob.glob("/home/vnwala/TEXT/*.txt")

with open("HtmlResults.txt", "wb") as outfile:
    for f in read_files:
        with open(f, "rb") as infile:
            outfile.write(infile.read())
```

Problem 2

Collection1: Extract all the unique terms and their frequency from the 10000 files*
Collection2: Extract all the unique terms and their frequency of the 10000 files* after running boilerpipe
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?
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.

Listing 5: Python script used to extract and count unique words and their frequencies for justext text files credits to Thiago Marzagao

```
### GENERATE WORD-FREQUENCY MATRICES
### author: Thiago Marzagao
### contact: marzagao ddott 1 at osu ddott edu

### supported encoding: UTF8
### supported character sets:
###     Basic Latin (Unicode 0-128)
###     Latin 1 Suplement (Unicode 129-255)
###     Latin Extended-A (Unicode 256-382)

import os
import re
import sys
import collections

ipath = '/home/vnwala/Result1/' # input folder
opath = '/home/vnwala/matrices/' # output folder

# identify files to process
done = set([file.replace('csv', 'txt') for file in os.listdir(opath)
            if file[-3:] == 'csv'])
filesToProcess = [file for file in os.listdir(ipath)
                  if file[-3:] == 'txt' if file not in done]
totalFiles = len(filesToProcess)

# quit if no files to process
if totalFiles == 0:
    sys.exit('No unprocessed txt files in {}'.format(ipath))

# map every uppercase character onto corresponding lowercase character
def upperToLower(obj):
```

```
    return(caseMap[obj.group('char')])
caseMap = {u'\u0041': u'\u0061', u'\u0042': u'\u0062', u'\u0043': u'\u0063',
           u'\u0044': u'\u0064', u'\u0045': u'\u0065', u'\u0046': u'\u0066',
           u'\u0047': u'\u0067', u'\u0048': u'\u0068', u'\u0049': u'\u0069',
           u'\u004A': u'\u006A', u'\u004B': u'\u006B', u'\u004C': u'\u006C',
           u'\u004D': u'\u006D', u'\u004E': u'\u006E', u'\u004F': u'\u006F',
           u'\u0050': u'\u0070', u'\u0051': u'\u0071', u'\u0052': u'\u0072',
           u'\u0053': u'\u0073', u'\u0054': u'\u0074', u'\u0055': u'\u0075',
           u'\u0056': u'\u0076', u'\u0057': u'\u0077', u'\u0058': u'\u0078',
           u'\u0059': u'\u0079', u'\u005A': u'\u007A', u'\u00C0': u'\u00E0',
           u'\u00C1': u'\u00E1', u'\u00C2': u'\u00E2', u'\u00C3': u'\u00E3',
           u'\u00C4': u'\u00E4', u'\u00C5': u'\u00E5', u'\u00C6': u'\u00E6',
           u'\u00C7': u'\u00E7', u'\u00C8': u'\u00E8', u'\u00C9': u'\u00E9',
           u'\u00CA': u'\u00EA', u'\u00CB': u'\u00EB', u'\u00CC': u'\u00EC',
           u'\u00CD': u'\u00ED', u'\u00CE': u'\u00EE', u'\u00CF': u'\u00EF',
           u'\u00D0': u'\u00F0', u'\u00D1': u'\u00F1', u'\u00D2': u'\u00F2',
           u'\u00D3': u'\u00F3', u'\u00D4': u'\u00F4', u'\u00D5': u'\u00F5',
           u'\u00D6': u'\u00F6', u'\u00D8': u'\u00F8', u'\u00D9': u'\u00F9',
           u'\u00DA': u'\u00FA', u'\u00DB': u'\u00FB', u'\u00DC': u'\u00FC',
           u'\u00DD': u'\u00FD', u'\u00DE': u'\u00FE', u'\u0100': u'\u0101',
           u'\u0102': u'\u0103', u'\u0104': u'\u0105', u'\u0106': u'\u0107',
           u'\u0108': u'\u0109', u'\u010A': u'\u010B', u'\u010C': u'\u010D',
           u'\u010E': u'\u010F', u'\u0110': u'\u0111', u'\u0112': u'\u0113',
           u'\u0114': u'\u0115', u'\u0116': u'\u0117', u'\u0118': u'\u0119',
           u'\u011A': u'\u011B', u'\u011C': u'\u011D', u'\u011E': u'\u011F',
           u'\u0120': u'\u0121', u'\u0122': u'\u0123', u'\u0124': u'\u0125',
           u'\u0126': u'\u0127', u'\u0128': u'\u0129', u'\u012A': u'\u012B',
           u'\u012C': u'\u012D', u'\u012E': u'\u012F', u'\u0130': u'\u0131',
           u'\u0132': u'\u0133', u'\u0134': u'\u0135', u'\u0136': u'\u0137',
           u'\u0139': u'\u013A', u'\u013B': u'\u013C', u'\u013D': u'\u013E',
           u'\u013F': u'\u0140', u'\u0141': u'\u0142', u'\u0143': u'\u0144',
           u'\u0145': u'\u0146', u'\u0147': u'\u0148', u'\u014A': u'\u014B',
           u'\u014C': u'\u014D', u'\u014E': u'\u014F', u'\u0150': u'\u0151',
           u'\u0152': u'\u0153', u'\u0154': u'\u0155', u'\u0156': u'\u0157',
           u'\u0158': u'\u0159', u'\u015A': u'\u015B', u'\u015C': u'\u015D',
           u'\u015E': u'\u015F', u'\u0160': u'\u0161', u'\u0162': u'\u0163',
           u'\u0164': u'\u0165', u'\u0166': u'\u0167', u'\u0168': u'\u0169',
           u'\u016A': u'\u016B', u'\u016C': u'\u016D', u'\u016E': u'\u016F',
           u'\u0170': u'\u0171', u'\u0172': u'\u0173', u'\u0174': u'\u0175',
           u'\u0176': u'\u0177', u'\u0178': u'\u00FF', u'\u0179': u'\u017A',
           u'\u017B': u'\u017C', u'\u017D': u'\u017E'}

# compile regular expressions
upperList = u'\u0041-\u005A\u00C0-\u00D6\u00D8-\u00DE\u0100\u0102\u0104\
              \u0106\u0108\u010A\u010C\u010E\u0110\u0112\u0114\u0116\u0118\
              \u011A\u011C\u011E\u0120\u0122\u0124\u0126\u0128\u012A\u012C\
              \u012E\u0130\u0132\u0134\u0136\u0139\u013B\u013D\u013F\u0141\
              \u0143\u0145\u0147\u014A\u014C\u014E\u0150\u0152\u0154\u0156\
              \u0158\u015A\u015C\u015E\u0160\u0162\u0164\u0166\u0168\u016A\
              \u016C\u016E\u0170\u0172\u0174\u0176\u0178\u0179\u017B\u017D'
regex1 = re.compile(u'[^\u0041-\u005A\u0061-\u007A\u00C0-\u00D6\u00D8-\u00F6\
                    \u00F8-\u00FF\u0100-\u017F\u0020\u002D]')
regex2 = re.compile(u'(^.{30,})')
```

```
regex3 = re.compile(u'(\A\u002D)|(\u002D\Z)')
regex4 = re.compile(ur'\A[{}]'.format(upperList))
regex5 = re.compile(ur'(?P<char>[{}])'.format(upperList))

# process each file
print ''
fileNumber = 0
for fileName in filesToProcess:
    if fileName[-3:] == 'txt': # discard non-txt files
        fileNumber += 1

        # monitor progress
        print 'Processing {} (file {} of {})'.format(fileName,
                                                     fileNumber,
                                                     totalFiles)

        # get file size and set chunk size
        chunkSize = 10000000 # number of bytes to process at a time
        fileSize = os.path.getsize(ipath + fileName)
        totalChunks = (fileSize / chunkSize) + 1

        # open file
        file = open(ipath + fileName, mode = 'r')

        # create dictionary to store word frequencies
        wordFreq = collections.Counter()

        # process each file chunk
        chunkNumber = 0
        while chunkNumber < totalChunks:

            # monitor progress
            chunkNumber += 1
            sys.stdout.write('Processing chunk {} of {} \r'
                             .format(chunkNumber, totalChunks))
            sys.stdout.flush()

            # read text
            rawText = file.read(chunkSize)

            # don't split last word
            separators = [' ', '\r', '\n']
            if chunkNumber < totalChunks:
                while rawText[-1] not in separators:
                    rawText = rawText + file.read(1)

            # decode text
            decodedText = rawText.decode('utf8')

            # remove special characters and anything beyond Unicode 382
            preCleanText = regex1.sub(' ', decodedText)

            # parse text
```

```
            parsedText = re.split(' |--', preCleanText)

            # clean up and count words
            uniques = set(parsedText)
            for word in parsedText:

                # if word > 30 characters, leave out
                if regex2.search(word):
                    continue

                # if word has trailing hyphens, fix
                while regex3.search(word):
                    word = regex3.sub('', word)

                # if word is empty string, leave out
                if word == '':
                    continue

                # if word == proper noun, leave out
                if regex4.search(word) and not regex5.search(word[1:]):
                    tempWord = regex4.sub(caseMap[word[0]], word)
                    if tempWord not in uniques:
                        continue

                # if word has uppercase, fix
                if regex5.search(word):
                    word = regex5.sub(upperToLower, word)

                # add word to count
                wordFreq[word] += 1

        # create output file
        output = fileName.replace('txt', 'csv')
        output = open(opath + output, mode = 'w')

        # write to output file
        totalWords = sum(wordFreq.values())
        for word, absFreq in wordFreq.items():
            relFreq = float(absFreq) / totalWords
            output.write(word.encode('utf8') + ','
                         + str(absFreq) + ','
                         + str(relFreq) + '\n')
        output.close()
        print '\n{} successfully processed'.format(fileName)
        print ''

# wrap up
print 'Done! All files successfully processed'
print 'Output saved to', opath
print ''
```

Listing 6: Python script used to extract and count unique words and their frequencies for html text files

```
file = open ( "HtmlResults.txt", "r" )
text = file.read()
file.close()
word_list = text.lower().split(' ')
word_freq = {}
saveFile = open('word.txt','a')
for word in word_list:
    word_freq[word] = word_freq.get(word, 0) + 1
keys = sorted(word_freq.keys())
for word in keys:
    print "%-10s %d" % (word, word_freq[word])
    saveFile.write(word+' '+str(word_freq[word])+'\n')
saveFile.close()
```

Figure 1: GRAPH OF WORDRANK VS FREQUENCY FOR TEXT

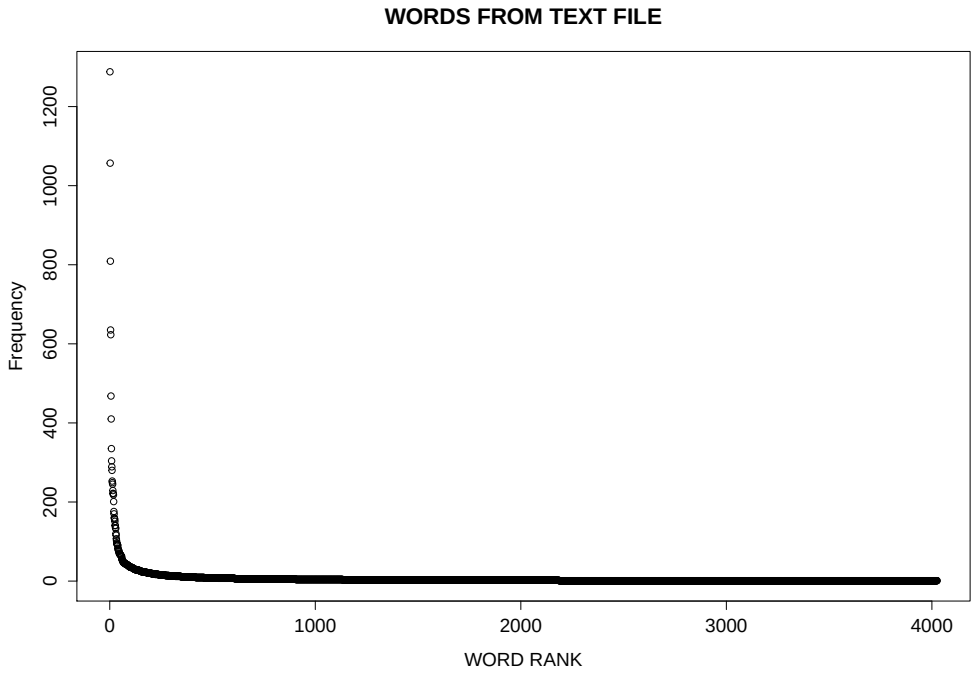
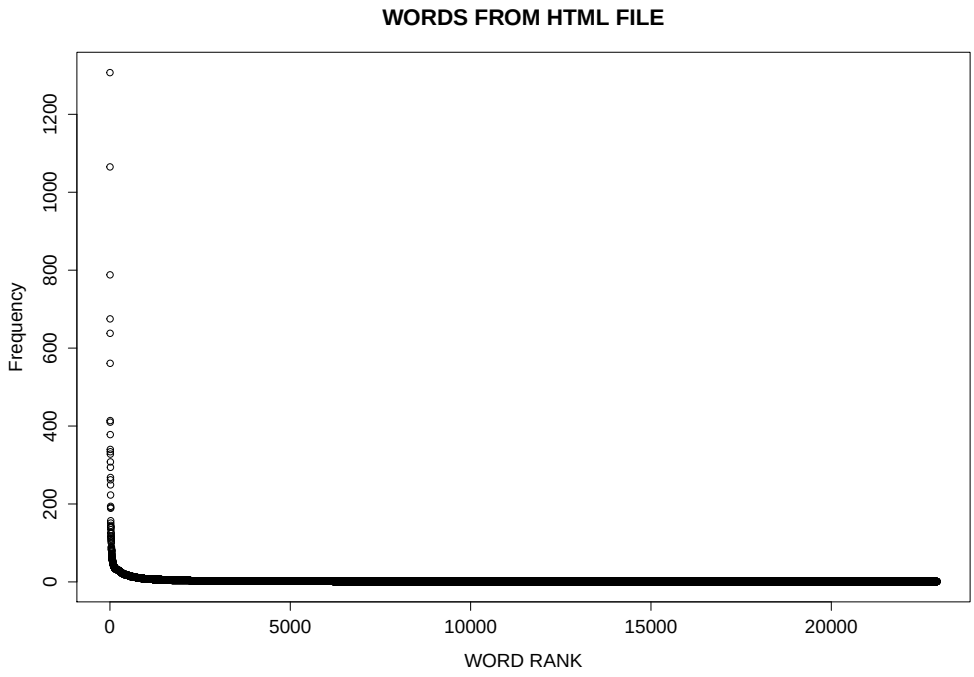


Figure 2: GRAPH OF WORDRANK VS FREQUENCY FOR HTML-TEXT



Problem 2 continued on next page...

Figure 3: GRAPH TO PROVE Zipfs LAW

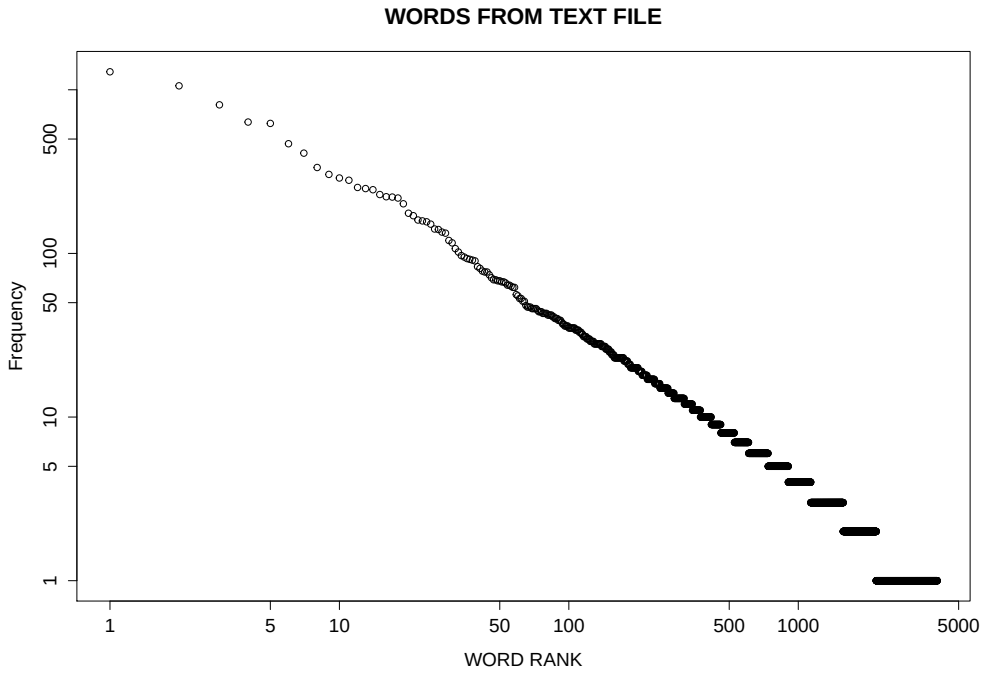
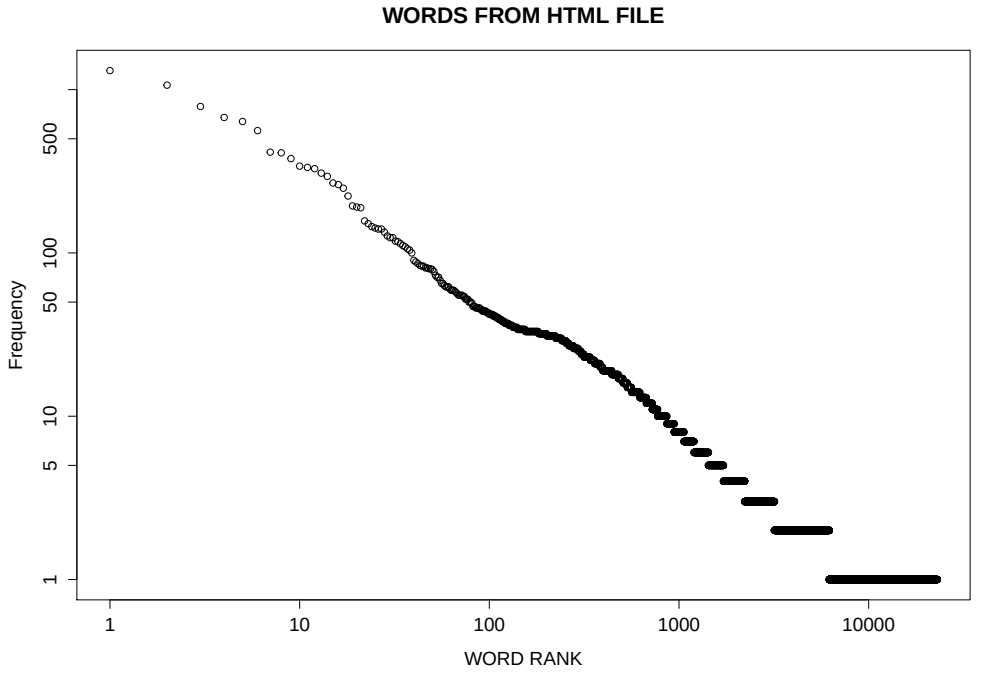


Figure 4: GRAPH TO PROVE Zipfs LAW



Using the stop word list I found that my first 50 words gotten form my text file had 47 stop words while the top 50 words gotten from HTML file had 42 stop words.

I also found that both graphs from the two collections follow the power law distribution . The classic almost perfect L-shape on the log-log scale is a dominant feature of power law graphs. I could not fully come to the absolute conclusion if the graph follows Zipf distribution, but this is what I discovered: Zipf's law is

generally understood to simply be a power-law distribution with integer values. But, power-law distributions have the special property that the complementary cumulative distribution function (ccdf) is also a power law form. This presents some ambiguity in interpreting what exactly people mean when they state that the estimated such-and-such graph distribution or parameter follows Zipf's Law.

References

[1] Onix Text Retrieval Toolkit API Reference. http://www.lextek.com/manuals/onix/stopwords1.html.

[2] Python Software Foundation. Creating Data Plots with R. "http://clasticdetritus.com/2013/01/10/creating-data-plots-with-r/", JANUARY 2013.

[3] Python Software Foundation. jusText 2.1.0. "https://pypi.python.org/pypi/jusText/2.1.0", 2014.

[4] Wikipedia. Zipf's law. "http://en.wikipedia.org/wiki/Zipf
[0]

Table 1: Word, Rank and Frequency From HTML File

RANK	WORD	FREQUENCY
1	the	1307
2	to	1065
3	and	788
4	of	675
5	a	638
6	in	561
7	at	414
8	for	410
9	by	378
10	is	340
11	i	334
12	on	328
13	with	308
14	you	294
15	that	268
16	your	262
17	this	249
18	posted	223
19	2015	194
20	it	191
21	o	189
22	20150204	157
23	from	151
24	as	145
25	or	142
26	are	140
27	—	140
28	have	134
29	my	127
30	all	124
31	be	118
32	not	114
33	can	111
34	new	109
35	published	106
36	out	104
37	more	100
38	will	90
39	an	88
40	get	86
41	has	84
42	one	83
43	was	83
44	like	81
45	about	80
46	her	80
47	but	79
48	up	71
49	when	71
50	march	68

Table 2: Word, Rank and Frequency From Justext File

RANK	WORD	FREQUENCY
1	the	1288
2	to	1057
3	and	809
4	of	635
5	a	623
6	you	468
7	in	410
8	is	335
9	that	304
10	i	289
11	for	280
12	it	253
13	with	249
14	or	245
15	on	229
16	this	222
17	we	221
18	s	218
19	your	201
20	as	176
21	from	170
22	information	160
23	will	158
24	are	156
25	not	151
26	by	141
27	be	140
28	at	135
29	have	133
30	plndr	120
31	can	116
32	my	107
33	t	102
34	an	97
35	if	95
36	when	93
37	our	92
38	use	91
39	so	90
40	more	83
41	but	81
42	do	78
43	any	77
44	about	77
45	her	74
46	new	71
47	us	69
48	may	69
49	time	68
50	their	68