Assignment Report

Apoorv Ingle
BT10CSE03

Problem Statement (Question 9):

- Generate character 5 grams from the tokens extracted out of the corpus
- Generate a log-log plot of frequency vs rank order
- Do the 5 grams follow Zipf's law, if so what is the approximate value of alpha

Program Architecture

The program has 3 main parts:

1. Generating 5-gram tokens from the corpus
2. Calculating the frequency of the 5-grams
3. Plotting a log-log plot of frequency vs rank order

Part 1 Generating 5-gram tokens from the corpus:

The corpus is a collection of 19997 text documents of various types such as essays, emails, articles etc. All these files are stored in the folder called "corpus".

The token generation is done via parallel processing, where multiple process are used to generate the tokens. This helps in reducing the overall time requrired to analyse the documents.

Each file generatres a gram-dictonary file in the folder "ngramsDatacorpus". This file is a text document which has the grams and its frequency in a python dictonary form.

The function digestAllFiles generates the 5 gram tokens from the corpus.

Part 2 Calculating the frequency of the 5-grams

All the files in ngramsDatacorpus are merged to form a master dictonary in which the 5-grams. This master dictonary contains all the 5-grams and their respective frequencies.

An inverted-freqency 5-gram list is then created, which stores all the frequency and 5-grams in a tuple form, in the highest fequency first order. This list is then stored in master_dict file in the parent folder.

The function getTopGrams generates the file master_dict.

Part 3 Plotting the log-log frequency vs rank order

The function plot plots the graph using the list of tuples generated in part 2.

Appendix A

Program Code

```
import os
import glob
import multiprocessing
from nltk.util import ngrams
from nltk.stem.porter import PorterStemmer
import ast
import sys
from matplotlib import pyplot
from math import log

#Stemmer to stem the words
ps = PorterStemmer()

def digestData(rawData):
“”“generates the ngrams of the given text
| input : raw data read from the file stream
| output: ngrams of the given input
“”“
#print “in digestData"
#strip the punctuations
finerText = str(rawData).lower().translate(None, “1234567890\/!?*+@#$%^&*()<>\"\’:;.,-|_\n\r\t”)
#stem the data
stemmedText = ps.stem(finerText)
#generate character 5-grams of the stemmed text
return ngrams(list(stemmedText),5)
```

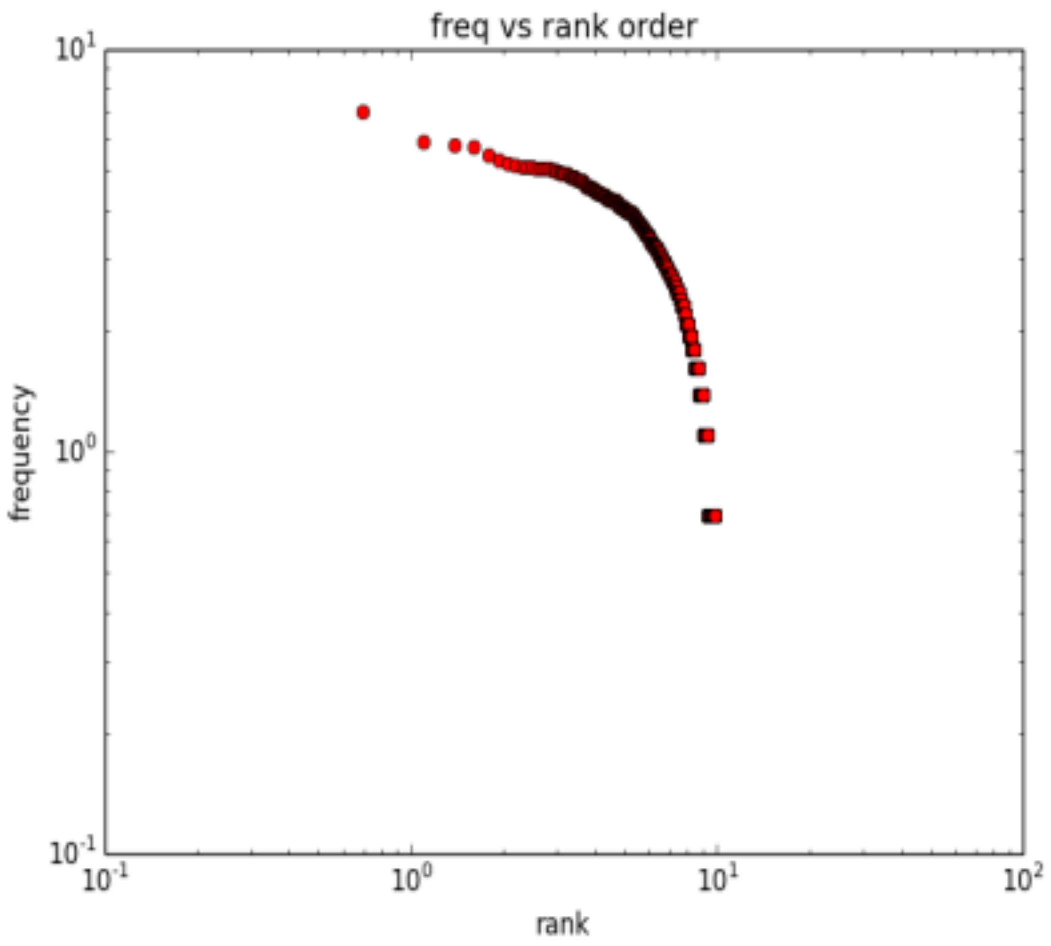
```
def digestFile(inFilePath, outFilePath):
    """reads file from inFilePath digests data and outputs
      the data in the outFilePath folder
      | input : path to the input file to read data inFilePath,
              path to the output file to write data outFilePath
      | output: None
    """
    try:
        #open input stream
        #print inFilePath
        d = dict()
        inFile = open(inFilePath)
        #open output stream
        outFile = open(outFilePath, "w")
        #read inputstream
        rawData = inFile.read()
        #print rawData
        digestedData = digestData(rawData)

        #get frequency of ngrams
        for gram in digestedData:
            if gram in d.keys():
                d[gram] = d[gram] + 1
            else:
                d[gram] = 1

        #print digestedData
        outFile.write(str(d))
    except IOError as e:
        print "**STATUS: I/O error({0}): {1}".format(e.errno, e.strerror)
        raise
    except:
        print "**STATUS: Unexpected error:", sys.exc_info()[0]
        raise

def digestMultipleFiles(inFileList, outFilePath):
    """digests multiple files given in the inFileList
      and generates corresponding data in the outFilePath
      | input : digest the files given in the inFileList
      | output: None
    """
    print "digesting partition " + inFileList[0]
    for fileName in inFileList:
        digestFile(fileName, outFilePath+fileName)
```

```
def digestAllFiles(inDirectory, outDirectory):
    """uses 17 parallel process to read
       all the files in the inDirectory
       digests data and writes in the outDirectory
       each file in the outDirectory contains the respective
       ngrams of each file
       | input : directory name in which the corpus is situated
       | output: None
    """

    process_list=[]

    for i in range(0,10):
        print "partition: " + inDirectory+"/9"+str(i)+"*"
        listOfFiles = glob.glob(inDirectory+"/9"+str(i)+"*")
        #print "list:" + str(listOfFiles)
        p = multiprocessing.Process(target=digestMultipleFiles, args=(listOfFiles, outDirectory))
        process_list.append(p)

    print "**STATUS: starting process execution"
    #start the thread execution
    for p in process_list:
        p.start()

    #wait for the threads to stop
    for p in process_list:
        p.join()

    print "**STATUS: all files data digested"
    print "input directory:" + inDirectory + " output Directory: " +outDirectory + "corpus"
```

```
def getTopGrams():
    """gets the dict from the files from ngramDatacorupus
       converts the string dict into dictonary
       each dict and their frequencies
       finds cumulative freq
       | input : None
       | output: None
    """
    print "**STATUS: creating master dictonary"
    master_dict = dict()
    file_list = glob.glob("ngramsDatacorpus/91*")
    for f in file_list:
        fp =  open(f)
        d = ast.literal_eval(fp.read())
        for gram in d.keys():
            if gram in master_dict.keys():
                master_dict[gram] = master_dict[gram] + d[gram]
            else:
                master_dict[gram] = d[gram]

    #generate rank order and frequency tuple
    valueSortedArray = []
    for key in master_dict.keys():
        valueSortedArray.append((master_dict[key],key))

    valueSortedArray.sort(reverse=True)
    op = open(r'master_dict', "w")
    op.write(str(valueSortedArray))

def plot(listOfTuples):
    """inputs list of tuples as (x,y) co-ordinates
       plot x vs y in log log scale
    """
    print "**STATUS: plotting"
    x = []
    y = []
    count = 1
    for tup in listOfTuples:
        x.append(log(count))
        count = count + 1
        y.append(log(tup[0]))
    pyplot.clf()
    pyplot.xscale('log')
    pyplot.yscale('log')
    pyplot.title('freq vs rank order')
    pyplot.xlabel('rank')
    pyplot.ylabel('frequency')
    pyplot.plot(x,y,'ro')
    pyplot.show()
```

```
digestAllFiles("corpus", "ngramsData")
getTopGrams()
f = open(r'master_dict')
listOfTuples = ast.literal_eval(f.read())
plot(listOfTuples)
```

Log-log plot of freqency vs rank order:



Appendix B

Python third party libraries

NLTK (www.nltk.org)
This library has an extensive functionality for text analysis

matplotlib (http://matplotlib.org/)
This is an open source library for plotting requirements like matlab plot functions.

References

- Extention to zipfs law to Words and Phrases (cl.ldc.upenn.edu/coling2002/proceedings/data/area-04/co-305.pdf)
- Prof. William D Lewis presentation on N grams (http://faculty.washington.edu/wlewis2/courses/570/570-Day-11-slides.pdf)

Future Amendments

- The functions can be encapusated in to a class for a better API.
- in getTopGrams function use parallel processes to generate the master_dict faster. Use a modified external sort to get the inverted frequency-Ngram tuple list