Homework #1: CMPT-413

Reading: NLTK Tutorial Chp 1, 2, 3;
http://nltk.org/doc/en/{introduction,programming,words}.html;
Distributed on Jan 14; due on Jan 28

Anoop Sarkar – anoop@cs.sfu.ca

Only submit answers for questions marked with †.

(1) Define a variable silly to contain the string: 'newly formed bland ideas are unexpressible in an infuriating way'. Now write code to perform the following tasks:

a. Split silly into a list of strings, one per word, using Python's split() operation.

b. Extract the second letter of each word in silly and join them into a string, to get 'eoldrnnnna'.

c. Build a list phrase4 consisting of all the words up to (but not including) 'an' in silly. Hint: use the index() function in combination with list slicing.

d. Combine the words in phrase4 back into a single string, using join(). Make sure the words in the resulting string are separated with whitespace.

e. Print the words of silly in alphabetical order, one per line.

(2) Write code to abbreviate text by removing all the vowels. Define a variable sentence to hold any string you like, then initialize a new string result to hold the empty string ''. Now write a for loop to process the string, one character at a time, and append any non-vowel characters to the result string.

(3) Write a program that takes a sentence expressed as a single string, splits it and counts up the words. Get it to print out each word and the word's frequency, one per line, in alphabetical order.

(4) Write regular expressions to match the following classes of strings:

a. A single determiner (assume that a, an, and the are the only determiners).

b. An arithmetic expression using integers, addition, and multiplication, such as 2*3+8.

(5) Using re.findall(), write a regular expression which will extract pairs of values of the form login name, email domain from the following string:

```
>>> str = """
... austen-emma.txt:hart@vmd.cso.uiuc.edu  (internet)  hart@uiucvmd (bitnet)
... austen-emma.txt:Internet (72600.2026@compuserve.com); TEL: (212-254-5093)
... austen-persuasion.txt:Editing by Martin Ward (Martin.Ward@uk.ac.durham)
... blake-songs.txt:Prepared by David Price, email ccx074@coventry.ac.uk
... """
```

(6) Write code to read a file and print it in reverse, so that the last line is listed first.

(7) † Write a Python program to eliminate duplicate lines from the input file. Provide the number of lines that remain after all duplicate lines are eliminated from `hw1.txt`. You should get 10 unique lines.

(8) † Read the Wikipedia entry on the Soundex Algorithm. Implement this algorithm in Python. Write a function `check` that accepts a string and the expected Soundex value, and prints out if your implementation of Soundex does, in fact, return that value. Check your implementation using the following checks:

```
s = soundex()
s.check("Euler", "E460")
s.check("Ellery", "E460")
s.check("guass", "G200")
s.check("gauss", "G200")
s.check("Ghosh", "G200")
s.check("HILBERT", "H416")
s.check("Heilbronn", "H416")
```

```
  s.check("Knuth", "K530")
  s.check("K ** n  U123t9247h   ", "K530")
  s.check("Kant", "K530")
  s.check("Lloyd", "L300")
  s.check("Liddy", "L300")
  s.check("Lukasiewicz", "L222")
  s.check("Lissajous", "L222")
  s.check("Wachs", "W200")
  s.check("Waugh", "W200")
  s.check("HYHYH", "H000")
  s.check("kkkkkkkwwwwkkkkkhhhhhkkkkemmnmnhmn", "K500")
  s.check("Rogers", "R262")
  s.check("Rodgers", "R326")
  s.check("Sinclair", "S524")
  s.check("St. Clair", "S324")
  s.check("Tchebysheff", "T212")
  s.check("Chebyshev", "C121")
  s.check("Bib", "B100")
  s.check("Bilbo", "B410")
  s.check("Bibby", "B100")
  s.check("Bibbster", "B123")
  s.check("Losh-shkan", "L225")
  s.check("Los-qam", "L225")
```

(9) Download the front page from *http://www.cbc.ca* print out to sys.stderr the number of characters, words and lines on the page. You can optionally strip out the html tags. Here's what the output should look like (number of chars, number of words, number of lines):

```
$ python2.5 urlwc.py | wc -l
6379 469 429
  429
```

(10) † Use the function `fisher_yates_shuffle` given below in a Python program which reads in a file and prints out each line with the words randomly shuffled. Each line of text should appear in the same order as the input but the words in each line should be randomly permuted. Test your program on `hw1.txt`.

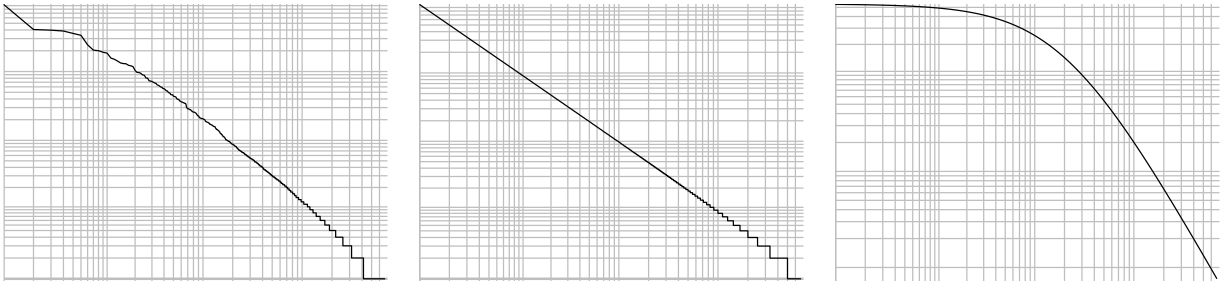
```
from random import *

# generator for a reverse iterator over an array
def reverse_iterator(data):
  for index in range(len(data)-1, -1, -1):
    yield index

# fisher yates random shuffle of an array
def fisher_yates_shuffle(array):
  for i in reverse_iterator(array):
    j = int(randrange(i+1))
    (array[i],array[j]) = (array[j],array[i])

if __name__ == "__main__":
  x = range(1,51)
  fisher_yates_shuffle(x)
  print " ".join(str(i) for i in x)
```

(11) † Using `nltk.corpus.gutenberg.words(’austen-sense’)`, collect the frequency of each word in the ’austen-sense’ corpus (Sense and Sensibility by Jane Austen) and print it out sorted by descending frequency. Print out the rank of the word, the frequency and the word itself. The first few lines of the output should look like this:



(a) (b) (c)

Figure 1: (a) Log plot of word rank (x-axis) against frequency (y-axis). (b) Log plot of Zipf's formula. (b) Log plot of Mandelbrot's formula using some arbitrary values for P, B and ρ.

```
$ python2.5 freq.py | head
1 9397  ,
2 4063  to
3 3975  .
4 3861  the
```

We say that ',' has rank of 1 and 'to' has rank of 2, and so on.

(12) **Zipf's Law**:

The rank r of each word (see Q 11) is based on its frequency f. Zipf's law states that $f \propto \frac{1}{r}$ or, equivalently, that $f = \frac{k}{r}$ for some constant factor k. For example, if Zipf's law holds then the 50th most common word should occur with three times the frequency of the 150th most common word. This relationship between frequency and rank was first noticed by J. B. Estoup in 1916, but was widely publicized by Zipf in his 1949 book: *Human Behaviour and the Principle of Least Effort*.

Use the Plot function defined in *nltk.draw* (or alternatively you can use the `pylab` module, see Listing 3.1 in the nltk docs) to plot the empirical relationship between a word's rank and its frequency for the 'austen-sense' corpus.

```
from nltk.draw import *
# sz should be set to the total number of word types in the corpus
freqplot = Plot(sortedfreq, range(1,sz), scale='log')
```

The following commands will produce a plot for Zipf's formula. We plot on the log scale, plotting $log(r)$ on the x-axis and $log(f)$ on the y-axis. We vary $r = 1 \ldots sz$ where there are sz words in the corpus and set the value $k = 10000$.

```
# sz should be set to the total number of word types in the corpus
zipfplot = Plot(lambda x: 10000/x, range(1,sz), scale='log')
```

Now compare Zipf's formula with the Mandelbrot formula $f = P(r + \rho)^{-B})$ which can be written as:

$$log(f) = log(P) - B \cdot log(r + \rho)$$

You can save your output as a postscript file by using the following command; Python will prompt you for a filename.

```
# sz should be set to the total number of word types in the corpus
zipfplot = Plot(lambda x: 10000/x, range(1,sz), scale='log')
zipfplot.postscript()
```

a. Change the parameters P, B and ρ in the Mandelbrot formula to match the empirical distribution of word frequencies.

b. † Submit the Python code that plots the empirical rank vs. frequency plot, the Zipf’s formula plot and the Mandelbrot’s formula plot which has been modified to match the empirical distribution. Also submit the postscript files generated by your code. Compare with Fig. 1.

c. What happens to the Mandelbrot formula when $B = 1$ and $\rho = 0$?

d. † A stemmer is a program that uses heuristic rules to convert a word into a stem (the word minus any characters that belong to the prefix or suffix). Download a Python implementation of the Porter stemmer from the web and plot the empirical distribution using stems instead of words. Submit the Python code for the plot and the postscript file.

(13) † Write a Python program called `exec.py` which will execute all the programs you submit in this assignment. For every homework you must submit a program called `exec.py` to enable us to run all your submitted programs. For example, the following `exec.py` runs the shuffle array program listed above which was saved as a file called `shuffle_array.py`:

```
import os
cmd = ’python2.5 shuffle_array.py’
print ’exec:’, cmd
os.system(cmd)
```

For some questions that produce a lot of output, e.g. Q 11, you can redirect your output to a file, e.g. `python2.5 freq.py > freq.out`. In these cases, `exec.py` should print out the filename and location of the output file.

(14) What does the following Python program do? What would be an example input file what would it print out?

```
import sys
import re
try: infile = open(sys.argv[1])
except: infile = sys.stdin
for line in infile:
    line = line.strip()
    if not line or line[0] == ’#’: break
    parens = re.sub(’[ˆ()]’, ’.’, line.replace(’(’, ’(.’).replace(’)’, ’.)’))
    try: matchparens = re.compile(parens)
    except:
        print >> sys.stderr, "Error in input"
        continue
    for t in matchparens.match(line).groups(): print t
```

(15) The following Python program prints out a dispersion plot: visually depicting how often a certain character occurs in the Jane Austen novel ’Sense and Sensibility’.

```
from nltk.corpus import gutenberg
from nltk.draw import dispersion
words = [’Elinor’, ’Marianne’, ’Edward’, ’Willoughby’]
dispersion.plot(gutenberg.words(’austen-sense’), words)
```

Compare the dispersion plot for the words *walking, talking, hunting* compared to all of the various inflected forms of the stems *walk, talk, hunt* that actually occur in the novel.