Homework #1: CMPT-413

Due on Jan 25, 2011

Anoop Sarkar – http://www.cs.sfu.ca/~anoop/teaching/cmpt413

Only submit answers for questions marked with †. For the questions marked with a ††; choose one of them and submit its answer.

(1) Important! To solve this homework you must read the following chapters of the NLTK Book, available online at *http://www.nltk.org/book*

- Chapter 1. Language Processing and Python
- Chapter 2. Accessing Text Corpora and Lexical Resources

(2) 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.

(3) 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.

(4) 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.

(5) 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.

(6) 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
... """
```

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

(8) † Write a Python program to eliminate duplicate lines from the input file. This process is often called *deduplication* which is particularly useful in very large text data-sets. Provide the number of lines that remain after all duplicate lines are eliminated from `hw1.txt`. You should get 10 unique lines.

(9) † Read the Wikipedia entry on the Soundex Algorithm. Implement this algorithm in Python. Define a soundex class that contains 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. Using your defined `soundex` class, you must have the following test cases in your program:

```
if __name__ == "__main__":
    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("kkkkkkkwwwwkkkkkhhhhhhkkkkemmnmnhmn", "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")
```

(10) 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):

```
$ python2.6 urlwc.py | wc -l
6379 469 429
     429
```

(11) † 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)
```

(12) † Using `nltk.corpus.gutenberg.words('austen-sense.txt')`, collect the frequency of each word in the 'austen-sense.txt' 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:

```
$ python2.6 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.

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

The rank r of each word (see Q 12) 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*.

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

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

Use the following Python code as a template:

```
from __future__ import division
from nltk.probability import FreqDist
import pylab
import math
import random

text = """
What is the air-speed velocity of an unladen swallow ?
What do you mean ?
An African or European swallow ?
What ?
I don't know that !
Auuuuuuuugh !
How do know so much about swallows ?
"""

# collect word counts in nltk.probability.FreqDist
fd = FreqDist(word.lower() for word in text.split())
for w in fd.keys():
    print w, fd[w]
freqs = [fd[w] for w in fd.keys()]
pylab.grid(True)
pylab.plot(freqs, label='Frequency')
# for large data use pylab.loglog for a logscale plot

sz = fd.B()+1 # number of word types in text

randvals = [random.randrange(1,5,1) for r in range(1,sz)]
```

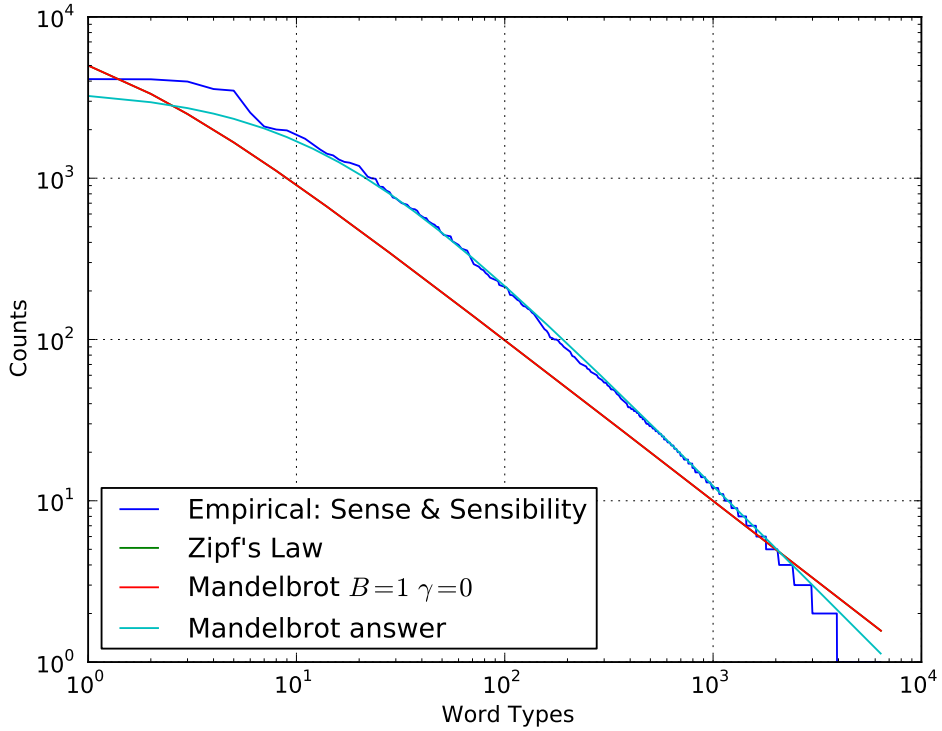

Figure 1: Log-log plot of word rank (x-axis) against frequency (y-axis), Zipf's formula and Mandelbrot's formula (using some arbitrary values for P, B and γ).

```
pylab.plot(sorted(randvals, reverse=True), label='Random')

# show all plots
pylab.xlabel("Word Types")
pylab.ylabel("Counts")
pylab.legend(loc='upper right')
pylab.show()
```

Save your graph as a PDF file using the Save to Disk menu icon.

For the following questions, use the text of Sense and Sensibility and collect the empirical word frequencies from the text as in Q (12).

a. Change the parameters P, B and γ 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. Submit the Python code and the PDF file of the plot. It should look like Fig. 1.

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

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). Use the Porter stemmer to plot the empirical distribution using stems instead of words. Submit the Python code that creates the plot and the PDF file of the plot. The following code prints the stem for the word *walking*:

```
from nltk import stem
p = stem.PorterStemmer()
print p.stem("walking")
```

(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('[^()]', '.', 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 and when 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.dispersion_plot(gutenberg.words('austen-sense.txt'), 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. Use the Porter stemmer to map from stems to inflected forms.

(16) ‡ Write a Python program to report the line numbers where each pair of adjacent words occurred in a text. The user can set a minimum frequency for each word in the pair, so the only word pairs that are reported are those in which each word in the pair occur with frequency ≥ the minimum frequency. On the file `hw1.txt` with a minimum frequency of 10 your program should print:

```
to $ 4
plant , 1 2 3 11 12 13 15
, a 5
, the 6 6 10 10 14
to a 9 17
the plant 4 15
```

(17) ‡ Pingala, a linguist who lived circa 5th century B.C., wrote a treatise on Sanskrit prosody called the Chandas Shastra. Virahanka extended this work around the 6th century A.D., providing the number of ways of combining short and long syllables to create a meter of length n. Meter is the recurrence of a pattern of short and long syllables (or stressed and unstressed syllables when applied to English). He found, for example, that there are five ways to construct a meter of length 4: V_4 = {LL, SSL, SLS, LSS, SSSS}. In general, we can split V_n into two subsets, those starting with L: {LL, LSS}, and those starting with S: {SSL, SLS, SSSS}. This provides a recursive definition for computing the number of meters of length n. Virahanka wrote down the following recursive definition on his trusty 6th century Python interpreter:

```
def virahanka(n, lookup = {0:[""], 1:["S"]} ):
    if (not lookup.has_key(n)):
        s = ["S" + prosody for prosody in virahanka(n-1)]
        l = ["L" + prosody for prosody in virahanka(n-2)]
        lookup.setdefault(n, s + l) # insert a new computed value as default
    return lookup[n]
```

Note that Virahanka has used top-down dynamic programming to ensure that if a user calls `virahanka(4)` then the two separate invocations of `virahanka(2)` will only be computed once: the first time `virahanka(2)` is computed the value will be stored in the `lookup` table and used for future invocations of `virahanka(2)`.

Write a *non-recursive* bottom-up dynamic programming implementation of the `virahanka` function. Provide the following output:

```
virahanka(4) = ['SSSS', 'SSL', 'SLS', 'LSS', 'LL']
```

(18) † 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 = 'python2.6 shuffle_array.py'
print 'exec:', cmd
os.system(cmd)
```

For some questions that produce a lot of output, e.g. Q 12, you can redirect your output to a file, e.g. `python2.6 freq.py > freq.out`. In these cases, `exec.py` should print out the filename and location of the output file.