Homework #4: CMPT-413

Distributed on Mon, Feb 14; Due on Mon, Feb 21

Anoop Sarkar – anoop@cs.sfu.ca

For each question the filename to use for your submission is either mentioned within parentheses with each question, e.g. (`filename.pl`) or the filename is explicitly mentioned in the text of the question. Submit a `makefile` so that `make test` will run all the completed components of this homework.

(1) **Rank and Frequency**: use the file `hw4.txt` as input for these programs.

a. (`freq.pl`) Write a Perl program that collects the frequency of each word and prints it out sorted by descending frequency. The first few lines of the output should look like this:

```
3842 ,
3116 the
2310 plant
2244 .
1740 a
...
```

b. (`rankFreq.pl`) Modify your code from Q.1a to print out the line numbers for each word and its frequency as shown below:

```
1     3842 ,
2     3116 the
3     2310 plant
4     2244 .
5     1740 a
...
```

The largest number in the 1st column is the total number of **word types**; the total of all the numbers in the 2nd column is the total number of **word tokens**; and each number in the 2nd column is the number of word tokens for the corresponding word in the 3rd column, e.g. in this text corpus there are 3116 tokens for the word type *the*.

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

Save the entire output of your program `rankFreq.pl` to a file called `hw4-2.plot` as follows:

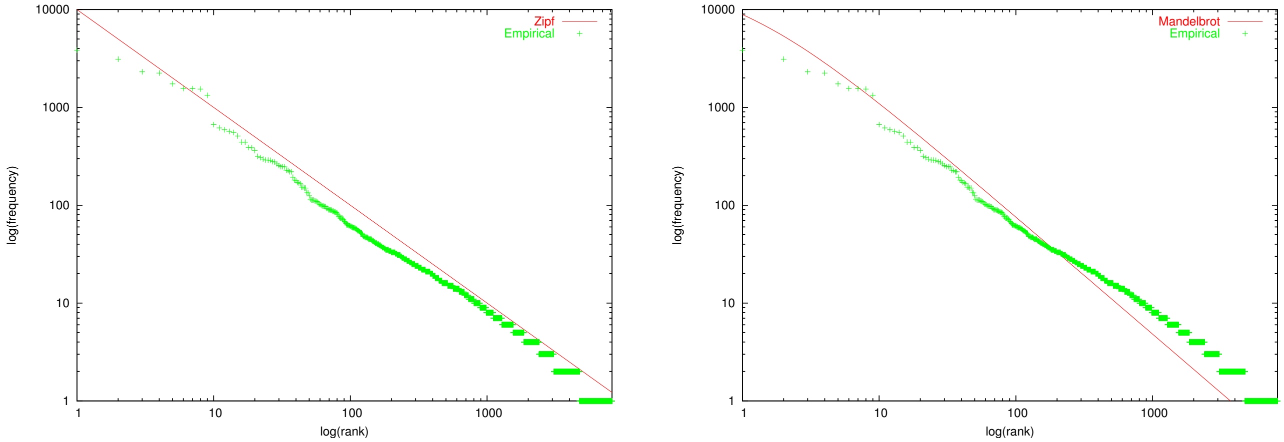
```
perl rankFreq.pl hw4.txt > hw4-2.plot
```

The line numbers correspond to the sorted rank r of each word based on its frequency f. Zipf's law states that

$$f \propto \frac{1}{r}$$

or, equivalently, that

$$f = \frac{k}{r}$$



(a) (b)

Figure 1: (a) Zipf's formula compared with the empirical distribution of word frequencies (b) Mandelbrot's formula compared with the empirical distribution.

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*.

In this question, we will use the GNU tool `gnuplot` to plot Zipf's formula against the empirical relationship between a word's rank and its frequency. At the unix shell, enter the command `gnuplot` and enter the following commands at the `gnuplot` shell (you will need to use an X11 terminal to view the graphs interactively, but you don't need one for saving the output to an `.eps` file). This will produce a plot that compares Zipf's formula with the empirical distribution you have stored in `hw4-2.plot`. We plot on the log scale, plotting $log(r)$ on the x-axis and $log(f)$ on the y-axis. Since there are 8200 tokens in our file `hw4.txt`, we vary $r = 1 \ldots 8200$ and set the value $k = 10000$. The `gnuplot` commands are:

```
set logscale
set xlabel "log(rank)"
set ylabel "log(frequency)"
plot [x=1:8200] 10000/x title "Zipf", 'hw4-2.plot' using 1:2 title "Empirical"
```

In order to get a better understanding of the log scale, try to view the graph without taking the log for each axis: `unset logscale` and then `plot` the graph again.

Now compare Zipf's formula with the Mandelbrot formula

$$f = P(r + \rho)^{-B}$$

or

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

This particular command to `gnuplot` uses the parameter settings: $P = 10000$, $\rho = 100$ and $B = 1.15$.

```
set logscale
set xlabel "log(rank)"
set ylabel "log(frequency)"
plot [x=1:8200] [1:] 10000*(x + 100)**(-1.15) title "Mandelbrot", \
  'hw4-2.plot' using 1:2 title "Empirical"
```

You can save your output as a postscript file by using the following commands *before* you enter the plot command:

```
set terminal postscript eps color
set output "filename.eps"
```

a. (`2a.eps`) Change the parameters P, B and ρ in the Mandelbrot function to match the observed distribution of word frequencies for `hw4.txt`.

b. (`2b.txt`) What happens to the Mandelbrot equation when $B = 1$ and $\rho = 0$?

c. (`zipfDictSearch.pl`) In this question we consider the implications of Zipf's law for natural language processing and the role of morphological processing. For this question you will need an additional file containing a list of English words called `dict.txt` (available in the usual course directory for this homework).

   Your perl program should take two files as input and print out 4 numbers:

   ```
   % perl zipfDictSearch.pl dict.txt hw4.txt
   68737 8200 5108 2077
   ```

   Each number is explained below (the first 3 numbers should match with your implementation):

   1. the total number of word tokens in `hw4.txt`,
   2. the total number of word types in `hw4.txt`,
   3. the total number of word types from `hw4.txt` that were **not** observed in `dict.txt` (notice that this number roughly corresponds to the number of single count words predicted by Zipf's law),
   4. the total number of word types from `hw4.txt` that were not observed in `dict.txt` after using Perl regular expression substitutions to find the **stem** of each word type in `hw4.txt`.
      For example, consider the following two Perl regex substitutions. The first removes the `im` prefix from the word in `$wd` only if the stem begins with `p`, `b` or `m`, and the second removes the `ly` suffix from any word:

      ```
      $wd =~ s/^im([pbm])/$1/;
      $wd =~ s/ly$//;
      ```

      As a result, the word `improperly` which occurs in `hw4.txt` but does not occur in `dict.txt` is converted into the stem `proper` which does occur in `dict.txt`. You should chain together an ordered list of such substitutions to get the lowest number you can for missing matches. Use the many examples of prefixes/suffixes in English from the lecture notes or J&M to create substitution rules. You should obtain a number that is ≤ 2554.
      You can print out the words that do not match to STDERR in order to improve your ad-hoc Perl stemmer and minimize the number of missing matches. You can optionally use a Perl implementation of the Porter stemmer from the web:

      `http://www.tartarus.org/~martin/PorterStemmer/perl.txt`

(3) **Minimum Edit Distance** (`edit-distance-align.pl`): Download the perl program `edit-distance.pl` from the location provided on the course web page. It implements a function `minEditDistance` which computes the minimum edit distance between two strings. Here are two examples of the output produced by `edit-distance.pl`:

```
% perl edit-distance.pl gamble gumbo
levenshtein distance = 5

% perl edit-distance.pl "recognize speech" "wreck a nice beach"
levenshtein distance = 14
```

Copy `edit-distance.pl` to a new file `edit-distance-align.pl` and then extend `edit-distance-align.pl` by modifying the function `minEditDistance` so that you can memorize or trace back your steps from the final score for the minimum distance alignment (for each substring record whether it was an insertion, deletion or substitution that provided the best alignment). Then, pass this information to a single new function `printAlignment` which produces the following visual display of the best (minimum distance) alignment:

```
% perl edit-distance-align.pl gamble gumbo
levenshtein distance = 5
g a m b l e
|   | |
g u m b _ o

% perl edit-distance-align.pl "recognize speech" "wreck a nice beach"
levenshtein distance = 14
_ r e c _ _ o g n i z e   s p e e c h
  | | |         | |   | |     |   | |
w r e c k   a   n i c e   _ b e a c h

% perl edit-distance-align.pl execution intention
levenshtein distance = 8
_ e x e c u t i o n
      |     | | | |
i n t e _ n t i o n
```

The 1st line of the visual display shows the *target* word and the 3rd line shows the *source* word. An insertion in the target word is represented as an underscore in the 3rd line aligned with the inserted letter in the 1st line. Deletion from the source word is represented as an underscore '_' in the 1st line aligned with the corresponding deleted character in the source on the 3rd line. Finally, if a letter is unchanged between target and source then a vertical bar (the pipe symbol '|') is printed aligned with the letter in the 2nd line. Producing a visual display with a different alignment but which has the same edit distance as the ones shown above is also acceptable.