# CSC 121, Spring 2017 — Large Assignment #2

*Worth 10% of the course grade. Due at 1:10pm April 4, to be handed in using MarkUs. This assignment may be handed in late, with a 20% penalty, by 1:10pm on April 7. Assignments will not usually be accepted after that. Contact the instructor as soon as possible if you have a legitimate excuse (such as documented illness) for handing in the assignment late (without penalty).*

*This assignment is to be done by each student individually. You may discuss it in general terms with other students, but the work you hand in should be your own. In particular, you shouldn't leave a discussion with someone else with any written notes (either paper or electronic).*

In this assignment, you will look at the text of the novel "Pride and Prejudice", by Jane Austin. You will investigate two questions — how well "Zipf's Law" works for this text, and how well you can predict the next word of the text based on the previous word. For each part of the assignment, you will produce an HTML document using `knitr`.

I have converted the text of "Pride and Prejudice" so that all words consist of upper case letters only, with one space between words, and with no punctuation, and no newlines. You can read this text in R with the following call of `scan`:

```
scan("http://www.cs.utoronto.ca/~radford/csc121/pride-and-prejudice.txt","")
```

The `""` as the second argument informs `scan` that the text consists of strings (not numbers). The result returned by `scan` is a vector of these strings; the length of this vector should be 122416.

Zipf's Law is not really a mathematically certain Law, but is just an observation that in many large documents the frequency with which a word occurs is close to being inversely proportional to the rank of the word (by frequency). The rank is defined to be 1 for the most common word, 2 for the second-most common word, etc. If $N_w$ is the number of times that word $w$ occurs in the document, and $R_w$ is the rank of word $w$, then Zipf's Law can be written as

$$N_w \;=\; c/R_w$$

where $c$ is some constant (the same for all words, but perhaps different for different documents). It's not expected that this equation will hold exactly, only approximately.

A generalization of Zipf's Law allows for the power of $R_w$ in this equation to be something other than $-1$. In this form,

$$N_w \;=\; c\,[R_w]^p$$

Here, $p$ would be $-1$ to give the original form of Zipf's Law. We might sometimes find that values of $p$ other than $-1$ work better.

We can take the logarithm of both sides of the equation above, getting

$$\log(N_w) \;=\; \log(c) \;+\; p\log(R_w)$$

We can therefore check whether Zipf's Law seems to hold for some document by looking at a scatterplot of $\log(N_w)$ versus $\log(R_w)$, in which a point is plotted for each word, $w$, in the document, with horizontal coordinate $\log(R_w)$ and vertical coordinate $\log(N_w)$. If Zipf's Law holds for some power $p$, these points should lie close to some straight line with slope $p$.

You should produce such a scatterplot, and add to it the line that best fits the points plotted, as fit by R's `lm` function. You should also show the slope and intercept of this line in your `knitr` output, and comment briefly on how close Zipf's Law seems to be to being correct.

You should then consider whether applying Zipf's Law separately to high-rank, middle-rank, and low-rank words might work better. By examining the plot you produced by eye, choose two values, $r_1$ and $r_2$, with which to separate words into three groups — one group with rank less than $r_1$, one group with rank between $r_1$ and $r_2$, and one group with rank greater than $r_2$. Use `lm` to fit a line to the points corresponding to the words in each group. On a new plot (separate from the one mentioned above), plot the points in all the groups, in different colours, and the lines fit to each group (in the same colour as the points for that group). Also, show the slopes and intercepts of these lines in your `knitr` output. Comment briefly on whether dividing the words into these groups seems to produce a substantial improvement in how well the lines fit.

Note: In these comments, you do not have to do any formal statistical tests.

Predicting what comes next in a document is useful for many purposes. It is the basis of methods for compressing text documents so that they occupy less space. It is also a useful device for speeding up user input, especially for disabled users who have very limited motor capabilities (eg, see the Dasher Project at `http://www.inference.phy.cam.ac.uk/dasher/`). Predictions are most useful when they take the form of probability distributions over the next word. However, in this assignment, we will consider only "best guess" predictions, that take the form of a guess at what the next word will be. We would like for this guess to be correct as often as we can manage.

You should investigate two methods of predicting each successive word in the second half of "Pride and Prejudice". The predictions must be based only on the first half of the novel, and on the words in the second half before the one being predicted. (Basing a prediction for a word on words that come after it would not be possible in a real scenario.)

The first method is very simple — for *every* word in the second half of "Pride and Prejudice", predict that it will be the word that occurred most often in the first half. (So this method always makes the same guess.) You should implement this method, and show (in your `knitr` output) for what fraction of words its guess is correct.

Rather than making the same guess for every word, we might hope to do better by having the guess depend on what the previous word was. The second method you implement should do this by guessing that a word in the second half that follows word $w$ will be the word that most commonly follows $w$ in the first half. If word $w$ does not appear in the first half (excluding the last word of the first half), the second method should guess the most common word in the first half (same as for the first method). In your knitr output, you should show what fraction of the time the second method's guess is correct.

You should implement the second method as follows: First, tentatively set the guess for every word to be the guess found with the first method above. For every word in the second half of "Pride and Prejudice", find the word before it, and then extract from the first half all the words that follow that word. If it turns out that there are no such words, you should leave the guess from the first method unchanged. If the word before the word being predicted does occur in the first half, find which is the most common word that follows it, and change the guess for this word to be that word.

You will likely find that this method is rather slow, taking a minute or more. You can find out exactly how slow by running the method using `system.time`, whose use is illustrated below:

```
> system.time (for (i in 1:10000000) x <- sqrt(i))
   user  system elapsed
  1.982   0.019   2.002
```

The number under "elapsed" is the actual time, in seconds (which should match your watch). The number under "user" will usually be just slightly smaller, but might be much smaller if the computer is doing lots of other things too. You can ignore the number under "system". You should run your second method with `system.time` and show the results in your `knitr` output.

**Bonus:** For up to 10 bonus marks, try to implement the second method in some way other than as described above, which takes signficantly less time, and show how much time it takes.

For this assignment, you may find uses for the following R functions, which have been or will be covered in lectures:

`table, sort, which, which.min, which.max, rank, lm, abline, names`

The `%in%` operator may also be useful. However, depending on how you write your program, you may not need to use all of these. And of course you will need to use various other R functions that we've covered as well.

You should hand in two R script files, one for the Zipf's Law part of this assignment, and one for the prediction part, both of which you run with `knitr`. These should be called `lga2-script1.R` and `lga2-script2.R`; their output files will be `lga2-script1.html` and `lga2-script2.html`. You should also hand in files with definitions of functions that you write for the two parts, called `lga2-defs1.R` and `lga2-defs2.R`. (You should not put function definitions in your `knitr` script files, unless perhaps they are very short.) You will need to decide yourself what functions you should write — but you should certainly write functions for the two methods of predicting the next word. One guide to when to define a function is that if you find that you are doing the same operations several times, you should put those operations in a function (if they aren't trivial).

You should do this assignment in a way so that it could easily be changed to look at some document other than "Pride and Prejudice".

You should properly indent your functions, and include comments where useful in both the function definitions and the `knitr` script.