NLP - Assignment 1

In this assignment you will... - download a book from the Gutenbrg project & read it into R. - perform simple processing steps and tokenize the text. - analyze the frequency distribution of tokens.

The goal of this assignment is to become familiar with... - importing of text into R. - regular expressions. - tokenization. - word frequency distributions.

Load text into R

The first task of this assignment consists of choosing a book of your liking and loading it into R.

1) Visit the website of Project Gutenberg and select a book that you like. Tipp: check out Project Gutenberg's Top 100 list. To download the book visit the book's website and select 'Textdatei UTF-8'. Depending on your browser (and your browser settings) this will either download the file directly or open up the text in the browser tab. In the latter case, you can use right-click on the text and select 'save as' (or comparable) to download the text as a text-file to your hard-drive (preferably your project folder).

2) Load text into R using the `read_file()` function from the `readr` package. To use it, first load (and, if necessary, also install) the package using the library `library()` (`install.packages()`). Then, use the `read_file()` function with the path to the text file as the (only) argument and assign the result to an object called, e.g., `text`. Note: Instead of taking the detour via your hard-drive, you can also provide the url to the text to load the text straight from the internet. Now you should have a text object containing a `vector` of type `character` and `length` one. Confirm this using `typeof()` and `length()`. You may also want to evaluate the number of characters using `nchar()` and inspect the first, say, 1000 characters using the `str_sub()` function from the `stringr` package (remember to install & load the package). The `str_sub()` function takes three arguments: 1) the text, 2) the starting character index (e.g., 1), 3) and the ending character index (e.g., 1000).

Tokenize text

3) Before you can move on to tokenize the text, you must remove several text sections added by the Project Gutenberg containing information on the text and the license of use. The sections are separated by header lines with leading and trailing star symbols, e.g., `*** START OF THIS PROJECT GUTENBERG EBOOK THE GAMBLER ***`. Build a regular expression that captures such lines using the escaped star symbol `\\*`, curly brackets `{}` to indicate the number of symbol repetitions, the print class `[:print:]` for every letter in between, and the plus `+` to indicate the number of print repetitions. Then, use the regular expression to split the text into its sections. To do this, use the `str_split()` function from the `stringr` package. This will return a list of length one, with the only element being a character vector containing the individual sections. Store the list in an object called, e.g., `text_split`. Next, you can use the fact that the main text will be in all likelihood the section with the largest number of characters. Count the number of characters in each section using `nchar()` by passing to the function the character vector of sections (i.e., the first element in the list, selected using `[[]]`). Now you can creat a `main_text`-object by assigning to it the largest element in the section vector.

4) Now you can tokenize the text, i.e, split the text into its individual words. This can conveniently be done using the `str_extract_all()` function from the `stringr` package. To do this, use the `[:alpha:]` class and the plus `+` symbol to extract all sequences of alphabetic characters of length one and larger. This will automatically disregard all non-alphabetic characters. The result is again a list of length one. Select the first element and store it as, e.g., `tokens()`.

Analyze frequency distribution

5) Now that you have a vector of tokens, you can begin analyzing the frequency distribution by counting the occurences of each word using `table()`. The `table()`-function will return a table object that you can treat, for the most part, like a named integer vector. Next, to inspect the token frequencies sort them using `sort()` with `decreasing = TRUE` so that the first items in the table become the most frequent ones. This allows you to access the, say, 100 most frequent words using `[1:100]` Note: `1:100` simply creates an integer vector with all values from 1 to 100. What are the most frequent words and do they reveal much about the content?

6) In the last task, you will have noticed that the most frequent tokens are words that are used in any context and thus are not very uninformative with regard to the contents of your specific book. To focus on words that are more relevant for the current context, try to remove 1) overly short words (e.g., words with less than five characters) and 2) stopwords. To prepare the former create a logical vector that indicates whether the token character lengths (tokens can be accessed via `names(tokens_frequencies)`) have less than five characters (use `nchar()` and `<`). To prepare the latter test which tokens are included in a common stopword lists. A suitable stopword list can be obtained using the `stopwords()`-function from `tm` package. If you evaluating a German book pass on `de` as the `kind` argument to access the German stopword lists. Now create a logical vector indicating whether tokens are included in the stopword list use the is-element-of operator `%in%`. E.g., `c(1, 7) %in% c(1, 2, 3, 4, 5)` tests for each element in the left-hand-side vector whether they are included in the right-hand-side vector, returning in this case `c(TRUE, FALSE)`. Now you can combine the two logical vectors using the pipe `|` operator (represents logical OR) to select cases where one or the other exclusion criterion is true and use the resulting logical vector to extract relevant tokens. To do this negate the logical vector using `!`, which flips all `TRUE` to `FALSE` and vice versa, and use it inside single brackets `[]`. Assign the result to a new object and inspect again the first 100 elements. Do the most frequent words look more relevant now?

7) One way to represent frequency distributions is via word clouds. Plot a word cloud using the `wordcloud()`-function from the wordcloud package (install & load). The function takes two arguments: 1) the token names (i.e., `names(tokens_frequencies)`) and 2) the token frequencies. For the second argument the function expects a numeric vector. Thus, you cannot provide the table of frequencies directly, but you have to transform it first to `numeric` using `as.numeric()`. You will find that not all words will easily fit into the plot. Try plotting only, say, the first few hundred words and play around with the `scale` argument (see `?wordcloud`). You will also want to set the plot margins to zero prior to using the function using `par(mar = c(0, 0, 0, 0)`. When happy with the plot, save the image using the *Export* function atop your plot window and post the plot on **twitter** using *#nlpbasel*.

8) As the final task, evaluate Zipf's law by determining the frequencies of individual token frequencies, e.g., how many words have frequency 5, using again `table()` and plotting their frequencies of individual token frequencies (y-axis) against the token frequencies (x-axis). To do this use `ggplot2`.

9) BONUS: Evaluate the relationship between word length and frequency. Is this in line with the information theoretic account of Zipf's law?