

# NLP - Assignment 2

In this assignment you will...

- split text by paragraph.
- stem words.
- create a term-document matrix.
- apply latent semantic analysis.
- determine the cosine between word representations.

The goal of this assignment is to...

- practice regular expressions.
- learn about stemming.
- learn about latent semantic analysis.

## Prepare text

The first task of this assignment consists of again choosing a book and preparing the text for creating a term-document and running lsa.

- 1) Select a new book from Project Gutenberg or use the one of the previous assignment, extract its main text, and convert everything to lower case.
- 2) Split the text into paragraphs. To do this assess the first few hundred characters of the text using `str_sub()` (remember the `stringr`-package?). You will find that the paragraphs are separated by some sequence of `\r` and `\n`. Identify the string that separates the paragraphs and then split the main text using the string as the pattern with `str_split()`. Remember that `stringr`-functions return a list. Extract the relevant data using `[[]]`.
- 3) Iterate over the paragraphs using a loop and in each iteration (1) extract the words of the paragraph, (2) stem them using `wordStem()` from the `SnowballC`-package, and put the paragraph back together using `paste()` on the words and setting `collapse = ' '`. Extract the words using the same approach as in last week's assignment. The use of `wordStem()` is straightforward. Note that you can pass on vectors, i.e., you can stem all words of the paragraph at once. When having put the paragraph back together, overwrite the original one in the vector of paragraphs (you could of course also make a copy first and overwrite the elements in the copy). You should now have a vector of paragraphs consisting only of stemmed words (without any punctuation, etc.).

## Term document matrix

- 4) Now you have one ingredient for your term-document matrix, the documents (paragraphs in our case). What's missing is the set of words you want to evaluate across these documents. Ideally, one uses all words, however, their number can easily be too large to still be computationally efficient. Thus, I recommend to ignore words (1) that are stopwords, (2) that are short (e.g., fewer than 5 characters long), (3) that are too rare (e.g., fewer than 5 occurrences in the text). Extract all of these words from the original main text and then stem them using `wordStem()` so that they match the stemmed paragraphs. Finally, determine the unique set of stemmed words using `unique()`. Note: If you end up with more than 10,000 words then throw out a few more.
- 5) Create an empty term document matrix matching the number of paragraphs (columns) and **stemmed** words (rows) using `matrix()`. Then, loop over the stemmed words and count for each word how often they occur in each of the paragraphs using `str_count()`. Store the result in the word's row of the term-document matrix. Ready is your term-document matrix.

## Singular value decomposition

- 6) Run the singular value decomposition using `svds()` from the `RSpectra` package. Provide the term-document matrix as input and specify the number of dimensions. Since we are only interested in the word representations specify set `k` and `nu` to some non-zero number, e.g., 200, but `nv` to zero. This will speed up computation, which can take a few minutes depending on the length of your text and the number of words and paragraphs. The result is a list containing the singular values (`d`) and word's singular (`u`) vectors.
- 7) Extract the word representations by multiplying singular values (`d`) and word's singular (`u`). The result should be an  $m \times k$  matrix, where  $m$  is the number of stemmed words and  $k$  the number of singular values/vectors, which contains the representations for each word within the singular vector space.

## Cosine similarities

- 8) Determine the cosine similarities for each combination of words using the `cosine()`-function from the package `lsa`. Provide as the argument the word representatons, but note that it expects words to be represented by columns, not rows. Thus, first transpose the matrix containing the word representations using `t()`. Then pass the transposed matrix on to `cosine()`. The result should be a  $m \times m$  matrix containing the cosine similarities for pair of words.
- 9) Now you can study the meaning of words by evaluating the set of closest associates for each word. To do this useful to first name the rows and columns of the cosine matrix using `rownames()` and `colnames()`. Then, you can cut out, for example, the 10th row, which will correspond to the word in the 10th position in the stemmed words' vector, using `[10, ]` and inspect the cosine similarities to all of the other words.

## Report

- 10) Choose a word of your liking and produce a wordcloud (using `wordcloud()`) that shows the associates of that word according to their cosine similarity. Note: You want to remove the actual word from the cosines, as it necessarily has a cosine similarity of 1. Post the result on twitter.