NLP - Assignment 3

In this assignment you will…

- create a term-document matrix.
- determine co-occurrence vectors using ppmi.
- calculate cosine similarities and page ranks.
- analyze fluency data.

Preparations

1) Complete all of the steps of the previous assignment, to obtain a tibble that loos like this.

```
# load text
text <- read_file('grimm.txt')

# define regex
regex <- '\\*{3}[:print:]*\\*{3}'

# cut text into sections
text_split = str_split(text, '\\*{3}[:print:]*\\*{3}')

# get sections
sections <- text_split[[1]]

# select main text
main_text <- sections[2]

# create tibble
text_tbl <- tibble(text = main_text)

# define regex
token_tbl <- text_tbl %>%
  unnest_tokens(sentence, text, token = "sentences") %>%
  mutate(sentence_ind = as.character(1:n())) %>%
  unnest_tokens(word, "sentence")

token_tbl
```

```
## # A tibble: 101,660 x 2
##    sentence_ind word
##    <chr>        <chr>
##  1 1            produced
##  2 1            by
##  3 1            emma
##  4 1            dudding
##  5 1            john
##  6 1            bickers
##  7 1            and
##  8 1            dagny
##  9 1            fairy
```

```
## 10 1             tales
## # ... with 101,650 more rows
```

2) Then remove stopwords using the code below.

```
# remove stopwords
token_tbl <- token_tbl %>%
  anti_join(get_stopwords('en'))
```

Term-Document Matrix

2) There are many powerful packages to determine Term-Document Matrices, such as the `tm` package. The same job can, however, also use be achieved using the basic R function `table()`. When the `table()` function is applied to a tibble consisting of more than one variable, then the function computes a full cross-table off all the entries in each variable. That is, for a tibble with a variable containing a sentence indicator and a variable containing words, it will tabulate words against sentences, which is exactly what we want. Apply `table()` to your tokenized tibble and call the object `tdm`. Make sure that the first variable contains the words.

```
# create term document matrix
tdm <- token_tbl %>%
  select(word, sentence_ind) %>%
  table()
```

3) Explore the object's dimensions using `dim()`. How many rows and columns are there?

```
dim(tdm)
```

```
## [1] 4751 4508
```

4) Use `mean(tdm == 0)` to count the proportion of zeros in the matrix. What proportion of cells are zero?

```
mean(tdm == 0)
```

```
## [1] 0.9979584
```

Positive PMI transformation

1) To calculate the positive PMI, first create a new matrix that contains the joint probability of words and sentences by diving `tdm` by `sum(tdm)`. Call the new matrix `p_tdm`.

```
p_tdm <- tdm / sum(tdm)
```

2) Next determine marginal probability distributions of words and sentences by calculating `rowSums()` and `colSums()` (applied to `p_tdm`). Save the results as `p_words` and `p_sentences`.

```r
p_words <- rowSums(p_tdm)
p_sentences <- colSums(p_tdm)
```

3) Now you have all you need to calculate the point-wise mutual information (pmi). That is you now the joint distribution $p(sentence, word)$ and the marginal distributions of words $p(word)$ and sentences $p(sentence)$. Compare to the formula in Bullinaria and Levy (2017, p. 514). The only thing missing is to divide each cell of `p_tdm` by the according values of `p_words` and `p_sentences`. One way to achieve this is by the outer product of $p(word)$ and $p(sentence)$. Run the code below.

```r
outer_mat <- outer(p_words, p_sentences)
```

4) Explore `outer_mat`. How many rows and columns does it have? How do the first few values, e.g., `outer_mat[1:10, 1:10]` correspond to `p_words[1:10]` and `p_sentences[1:10]`? Check out the **outer product** on Wikipedia.

```r
dim(outer_mat)
```

```
## [1] 4751 4508
```

```r
outer_mat[1:10, 1:10]
```

```
##                1           10          100         1000         1001
## _jug 1.13434e-08 9.370634e-09 1.479574e-09 2.959148e-09 8.877443e-09
## _my_ 1.13434e-08 9.370634e-09 1.479574e-09 2.959148e-09 8.877443e-09
## 1    2.26868e-08 1.874127e-08 2.959148e-09 5.918295e-09 1.775489e-08
## 1785 1.13434e-08 9.370634e-09 1.479574e-09 2.959148e-09 8.877443e-09
## 1786 1.13434e-08 9.370634e-09 1.479574e-09 2.959148e-09 8.877443e-09
## 1812 1.13434e-08 9.370634e-09 1.479574e-09 2.959148e-09 8.877443e-09
## 1814 1.13434e-08 9.370634e-09 1.479574e-09 2.959148e-09 8.877443e-09
## 1823 1.13434e-08 9.370634e-09 1.479574e-09 2.959148e-09 8.877443e-09
## 1859 1.13434e-08 9.370634e-09 1.479574e-09 2.959148e-09 8.877443e-09
## 1863 1.13434e-08 9.370634e-09 1.479574e-09 2.959148e-09 8.877443e-09
##              1002         1003         1004         1005         1006
## _jug 2.959148e-09 1.183659e-08 8.877443e-09 2.959148e-09 5.918295e-09
## _my_ 2.959148e-09 1.183659e-08 8.877443e-09 2.959148e-09 5.918295e-09
## 1    5.918295e-09 2.367318e-08 1.775489e-08 5.918295e-09 1.183659e-08
## 1785 2.959148e-09 1.183659e-08 8.877443e-09 2.959148e-09 5.918295e-09
## 1786 2.959148e-09 1.183659e-08 8.877443e-09 2.959148e-09 5.918295e-09
## 1812 2.959148e-09 1.183659e-08 8.877443e-09 2.959148e-09 5.918295e-09
## 1814 2.959148e-09 1.183659e-08 8.877443e-09 2.959148e-09 5.918295e-09
## 1823 2.959148e-09 1.183659e-08 8.877443e-09 2.959148e-09 5.918295e-09
## 1859 2.959148e-09 1.183659e-08 8.877443e-09 2.959148e-09 5.918295e-09
## 1863 2.959148e-09 1.183659e-08 8.877443e-09 2.959148e-09 5.918295e-09
```

```r
p_words[1:10]
```

```
##         _jug         _my_            1         1785         1786
## 2.220791e-05 2.220791e-05 4.441582e-05 2.220791e-05 2.220791e-05
##         1812         1814         1823         1859         1863
## 2.220791e-05 2.220791e-05 2.220791e-05 2.220791e-05 2.220791e-05
```

```r
p_sentences[1:10]
```

```
##            1           10          100         1000         1001
## 5.107819e-04 4.219503e-04 6.662373e-05 1.332475e-04 3.997424e-04
##         1002         1003         1004         1005         1006
## 1.332475e-04 5.329899e-04 3.997424e-04 1.332475e-04 2.664949e-04
```

5) Using `outer_mat`, you can now conveniently compute $\frac{p(sentence,word)}{p(word)*p(sentence)}$ by dividing `p_tdm` by `outer_mat`. Do so. Name the result `pmp`, for point-wise mutual probability.

```r
pmp <- p_tdm / outer_mat
```

6) Calculate the `pmi` from `pmp` by taking the logarithm to the base 2 (`log2()`) of `pmp`.

```r
pmi <- log2(pmp)
```

7) Finally, change all negative values in `pmi` to `0` using the code below.

```r
ppmi <- pmi
ppmi[ppmi < 0] <- 0
```

Cosine similarity

Next, we want to construct a matrix containing the cosine similarities between the word vectors of each pair of words. To do this, we can again make use of some matrix algebra. The general form of the cosine similarity is $cos = \frac{A \cdot B}{\|A\| \|B\|}$, where $\|A\| = \sqrt{\sum_i A_i^2} = \sqrt{A \cdot A}$. This means that we need to determine (1) the **dot products** of all pairs of word vectors and (2) the square of the dot-product of each word vector with itself. Turns out we can do both in one step.

1) Compute the **matrix product** of `ppmi` and its transpose `t(ppmi)`. To compute the matrix product use `%*%` rather than `*`. Name the resulting object `dotprod`. Be aware that the calculation may take a moment.

```r
dotprod <- ppmi %*% t(ppmi)
```

2) Explore `dotprod`. How many rows and columns? Do the numbers match the number of words?

```r
dim(dotprod)
```

```
## [1] 4751 4751
```

3) Next, we need to normalize (aka divide) the dot product by the square root of the dot products of the words with themselves, i.e., $\sqrt{A \cdot A}$ and $\sqrt{B \cdot B}$. Turns out, we already calculated $A \cdot A$ and $B \cdot B$. In the previous step we calculated the dot products of all possible pairs of words, such that `dotprod[1, 2]` contains the dot product of words `1` and `2`, `dotprod[3, 8]` contains the dot product of words `3` and `8`, and so on. Correspondingly, `dotprod[1, 1]` contains the dot product of word `1` with itself. Thus, the diagonal of `dotprod` contains the $A \cdot A$ and $B \cdot B$. Use `diag()` to select the diagonal of `dotprod` and store it in `dotprod_diag`.

```
dotprod_diag <- diag(dotprod)
```

4) Now calculate the outer product of the square root of `dotprod_diag`, i.e., `sqrt(dotprod_diag)` using `outer()` and store it as `dotprod_diag_outer`.

```
dotprod_diag_outer <- outer(sqrt(dotprod_diag), sqrt(dotprod_diag))
```

5) Finally, divide `dotprod` by `dotprod_diag_outer` to obtain a matrix of cosine similarities. Name the result `cosines`.

```
cosines <- dotprod/dotprod_diag_outer
```

6) Explore your cosines a bit. For instance, to examine the 20 closest associates to a word you can use `sort(cosines[word,], decreasing = T)[1:20]`. In my case, using the brothers Grimm works, the 20 closest associates to `gretel` are ...

```
##     gretel     hansel   presents       hans      comes    goodbye
## 1.00000000 0.27321563 0.23655510 0.15694120 0.15410732 0.11514618
##   pinafore      guest       give      binds      leads      ferry
## 0.11125674 0.09863558 0.09232053 0.08947632 0.08947632 0.08708551
##       good       crab     master     cutter      leant    nibbled
## 0.08544284 0.08492275 0.08338411 0.08294832 0.08294832 0.08294832
##      panes   daintily
## 0.08294832 0.07945037
```

Page rank

1) Use the code below to determine the *page rank* of each of the words in your cosine matrix. First, however, make sure to install the `igraph` package using `install.packages('igraph')` (only once).

```
# import to igraph
net = igraph::graph_from_adjacency_matrix(cosines,
                                          mode = "undirected",
                                          weighted = TRUE)

# determine page rank
pagerank = igraph::page_rank(net)$vector
```

2) Explore which words have highest page rank values using `sort(pagerank,decreasing = T)[1:50]`. The 50 highest page ranks in the works of the brothers Grimm are...

```
##         came         said          one         went       little
## 0.0011853102 0.0011644919 0.0011390171 0.0011147856 0.0010871668
##         upon         away         took        great          old
## 0.0009574306 0.0009414044 0.0008955111 0.0008809835 0.0008289580
##         time         king          two          saw         made
## 0.0008208702 0.0008163071 0.0008158298 0.0008025343 0.0007913428
##          man         last         soon           go         home
## 0.0007827924 0.0007808577 0.0007713826 0.0007695734 0.0007633816
```

```
##          way         till          sat        began          day 
## 0.0007531015 0.0007525183 0.0007473171 0.0007309039 0.0007305330 
##          now         fell         like          put    beautiful 
## 0.0007296186 0.0007263456 0.0007217672 0.0007213177 0.0007088929 
##         long        heard        water         tree         back 
## 0.0007081556 0.0007065166 0.0007006529 0.0006934865 0.0006921714 
##         must        still      thought          ran        found 
## 0.0006864377 0.0006849015 0.0006839727 0.0006770969 0.0006768850 
##         come          got        house         door          see 
## 0.0006768354 0.0006765443 0.0006701150 0.0006654688 0.0006652371 
##          set        three         gold         head         take 
## 0.0006630519 0.0006626328 0.0006625790 0.0006576520 0.0006545981
```

Explore letter fluency data

1) Load the letter fluency data sets with the following code. NOTE: You need to be connected to the internet to load the data.

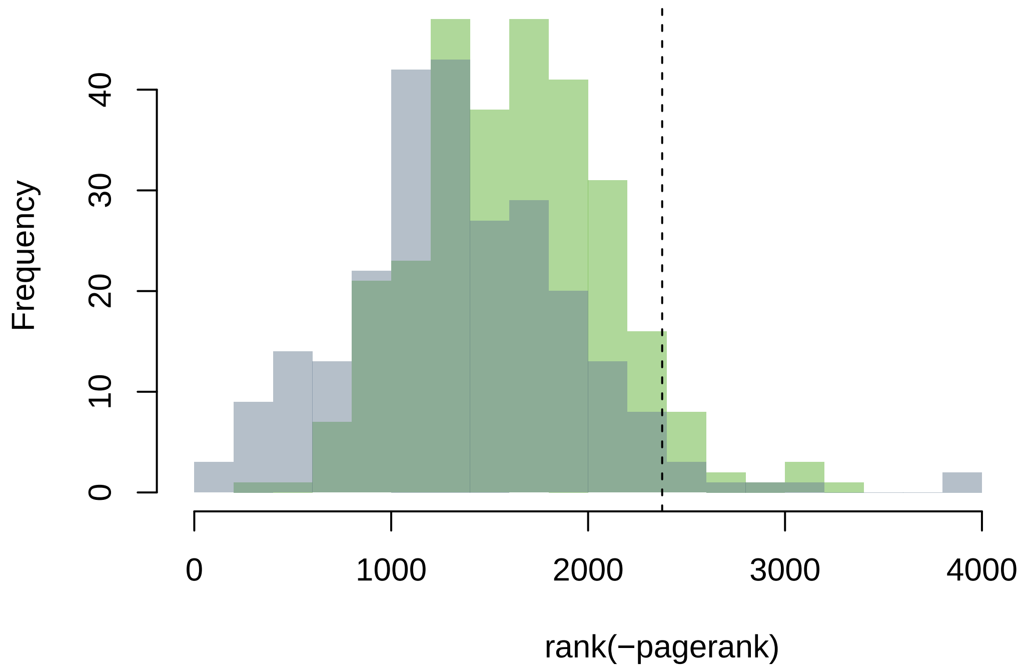
```r
# fluency data links
link_m <- 'https://www.dirkwulff.org/courses/2019_naturallanguage/data/letter_m.RDS'
link_s <- 'https://www.dirkwulff.org/courses/2019_naturallanguage/data/letter_s.RDS'

# read fluency data
letter_m <- readRDS(url(link_m))
letter_s <- readRDS(url(link_s))
```

2) Using the code below, determine the average rank of the page rank values for a given list in each of the fluency data vectors. Remember `rank(-pagerank)` means that small ranks have higher page ranks.

```r
# get average ranks
pg_letter_m <- sapply(letter_m, function(x) mean(rank(-pagerank)[x], na.rm = T))
pg_letter_s <- sapply(letter_s, function(x) mean(rank(-pagerank)[x], na.rm = T))
```

3) Using the code below, plot the average ranks for both letters and compare against the overall average rank. Are the average ranks smaller than the overall average rank?

```r
# plot average rank pageranks of s and m
cols = c("#5FB2337F", "#6A7F937F")
hist(pg_letter_m, xlim = c(0, length(pagerank)),
     breaks = 20, col = cols[1], border=NA,
     main = '', xlab = 'rank(-pagerank)')
hist(pg_letter_s, add = TRUE, col = cols[2],border=NA,
      xlim = c(0, length(pagerank)), breaks = 20)
abline(v = mean(rank(-pagerank)), lwd=1, lty=2)
```



Explore category fluency data

1) Load the category fluency data sets with the following code. NOTE: You need to be connected to the internet to load the data.

```
# fluency data links
link_ani <- 'https://www.dirkwulff.org/courses/2019_naturallanguage/data/animals.RDS'
link_veg <- 'https://www.dirkwulff.org/courses/2019_naturallanguage/data/veggies.RDS'

# read fluency data
animals <- readRDS(url(link_ani))
veggies <- readRDS(url(link_veg))
```

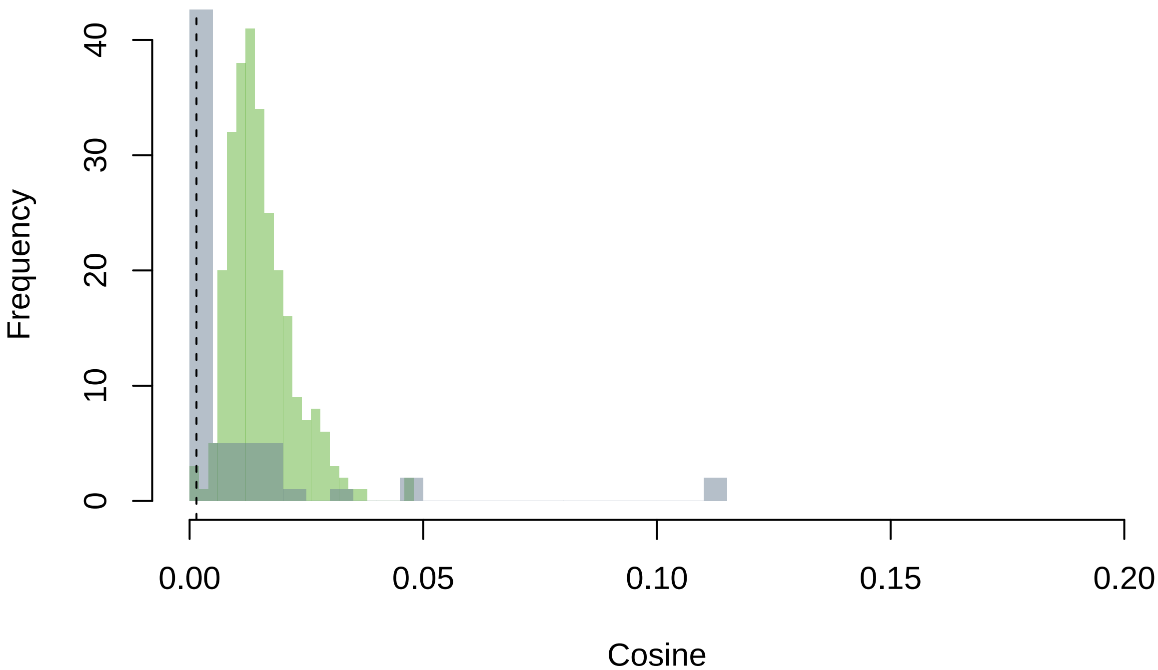
2) Using the code below, determine the average cosine similarity for the words within each fluency list.

```
# set the cosine diagonal to 0
cosines_noloop <- cosines
diag(cosines_noloop) <- 0

# words in cosine mat
nam <- rownames(cosines_noloop)

# extract cosines fun
extract_cosines <- function(words){
  words <- unique(words)
  words <- words[words %in% nam]
  cosines_noloop[words, words]
  }

# get average ranks
```

```r
cos_animals <- sapply(animals, function(x) mean(extract_cosines(x)))
cos_veggies <- sapply(veggies, function(x) mean(extract_cosines(x)))
```

3) Using the code below, plot the average cosines for both categories and compare against the overall average cosine. Are the average cosines larger than the overall average cosine? What do you think, what do the Brothers Grimm talk more about, animals or veggies?

```r
# plot average cosines of animals and veggies
cols = c("#5FB2337F", "#6A7F937F")
hist(cos_animals, xlim = c(0, .2),
     breaks = 20, col = cols[1], border=NA,
     main = '', xlab = 'Cosine')
hist(cos_veggies, add = TRUE, col = cols[2],border=NA,
      xlim = c(0, .2), breaks = 20)
abline(v = mean(cosines_noloop), lwd=1, lty=2)
```



BONUS: Category fluency transitions

Another interesting analysis of the category fluency data evaluates whether the word-to-word transitions in the fluency lists have higher cosine similarity than the average. Analyze this for both categories.

```r
# extract cosines fun
compute_cosine_diff <- function(words){

  # transition cosine
  transitions <- cbind(words[-length(words)],words[-1])
  transitions <- transitions[transitions[,1] %in% nam &
                             transitions[,2] %in% nam, ]
  transition_cos <- mean(cosines_noloop[transitions])

```

```r
  # overall cosine
  words <- unique(words)
  words <- words[words %in% nam]
  average_cos <- mean(cosines_noloop[words, words])

  transition_cos - average_cos

  }

# get average ranks
cos_diff_animals <- sapply(animals, compute_cosine_diff)
cos_diff_veggies <- sapply(veggies, compute_cosine_diff)

# plot results
cols = c("#5FB2337F", "#6A7F937F")
hist(cos_diff_animals, xlim = c(-.2, .2),
     breaks = 20, col = cols[1], border=NA,
     main = '', xlab = 'cos(Transitions) - cos(Average)')
hist(cos_veggies, add = TRUE, col = cols[2],border=NA,
      xlim = c(-.2, .2), breaks = 20)
abline(v = 0, lwd=1, lty=2)
```