# Eksploracja tekstu i wyszukiwanie informacji w mediach społecznościowych

## LABORATORIUM 3

- Ważenie termów
- Korpusy
- Macierz termów-dokumentów

### Ważenie termów

Na poprzednich zajęciach rozpatrywaliśmy przykłady pojedynczych tekstów, na których wykonywaliśmy proste operacje związane z ich eksploracją, takie jak rozdział na słowa, usuwanie słów funkcyjnych. W wielu przypadkach mamy jednak do czynienia z grupą dokumentów - poniżej (zabawkowy) przykład 3 "dokumentów"

```
# PRZYKŁAD 3.1
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(tidytext)

data <- tibble(doc_id = 1:3, text =
c("Once upon a time there was a king who wished to have a child.",
"Once upon a time, in a faraway land there lived a beautiful princess.",
"In the olden time, when wishing was having, there used to live a King."))

data
```

```
## # A tibble: 3 x 2
##   doc_id text
##    <int> <chr>
## 1      1 Once upon a time there was a king who wished to have a child.
## 2      2 Once upon a time, in a faraway land there lived a beautiful prin..
## 3      3 In the olden time, when wishing was having, there used to live a..
```

To, do czego będziemy zmierzać, to wyznaczenie omawianych na wykładzie wielkości związanych z

popularnością słów, czyli **TF** oraz **TF-IDF**. Zacznijmy od tego, że potrzebne będą nam zliczenia słów w poszczególnych dokumantach

```
# PRZYKŁAD 3.2
doc_words <- data %>%
  unnest_tokens(word, text) %>%
  count(doc_id, word, sort = TRUE)

doc_words
```

```
## # A tibble: 37 x 3
##    doc_id word      n
##     <int> <chr> <int>
##  1      1 a         3
##  2      2 a         3
##  3      1 child     1
##  4      1 have      1
##  5      1 king      1
##  6      1 once      1
##  7      1 there     1
##  8      1 time      1
##  9      1 to        1
## 10      1 upon      1
## # ... with 27 more rows
```

Z drugiej strony, chcemy też mieć całkowitą liczbę słów w dokumentach: wykorzystamy tu funkcje **group_by()** oraz **summarize()**:

```
# PRZYKŁAD 3.3
total_words <- doc_words %>%
  group_by(doc_id) %>%
  summarize(total = sum(n))

total_words
```

```
## # A tibble: 3 x 2
##   doc_id total
##    <int> <int>
## 1      1    14
## 2      2    13
## 3      3    14
```

Na koniec powiążemy te dwie tabele za pomocą funkcji **left_join()**:

```
# PRZYKŁAD 3.4

doc_words <- doc_words %>%
  left_join(total_words)
```

```
## Joining, by = "doc_id"
```

```
doc_words
```

```
## # A tibble: 37 x 4
##    doc_id word      n total
##     <int> <chr> <int> <int>
##  1      1 a         3    14
##  2      2 a         3    13
##  3      1 child     1    14
##  4      1 have      1    14
##  5      1 king      1    14
##  6      1 once      1    14
##  7      1 there     1    14
##  8      1 time      1    14
##  9      1 to        1    14
## 10      1 upon      1    14
## # ... with 27 more rows
```

Mając taką strukturę, możemy już bez problemu policzyć np *term frequency*, czyli czestość termów daną wzorem

$$TF_{ij} = \frac{n_{ij}}{\sum_k n_{kj}}$$

gdzie $n_{ij}$ to liczba wystąpień słowa $i$ w dokumencie $j$. Wykorzystamy do tego funkcję **mutate()**

```
# PRZYKŁAD 3.5

doc_words %>%
  mutate(tf = n / total)
```

```
## # A tibble: 37 x 5
##    doc_id word      n total     tf
##     <int> <chr> <int> <int>  <dbl>
##  1      1 a         3    14 0.214
##  2      2 a         3    13 0.231
##  3      1 child     1    14 0.0714
##  4      1 have      1    14 0.0714
##  5      1 king      1    14 0.0714
##  6      1 once      1    14 0.0714
##  7      1 there     1    14 0.0714
##  8      1 time      1    14 0.0714
##  9      1 to        1    14 0.0714
## 10      1 upon      1    14 0.0714
## # ... with 27 more rows
```

Jednak łatwiej jest od razu otrzymać również wartości **TF-IDF**. Oczywiście możemy to policzyć "na piechotę", ale łatwiej jest skorzystać z funkcji **bind_tf_idf()**. Dodatkowo skierujemy strumień poprzez funkcję **print()** i każdemy jej wypisać pełną ramkę:

```
# PRZYKŁAD 3.6

doc_words %>%
  bind_tf_idf(word, doc_id, n) %>%
  arrange(desc(tf_idf)) %>%
  print(n = Inf)
```

```
## # A tibble: 37 x 7
##    doc_id word          n total     tf   idf tf_idf
##     <int> <chr>     <int> <int>  <dbl> <dbl>  <dbl>
##  1      2 beautiful     1    13 0.0769 1.10  0.0845
##  2      2 faraway       1    13 0.0769 1.10  0.0845
##  3      2 land          1    13 0.0769 1.10  0.0845
##  4      2 lived         1    13 0.0769 1.10  0.0845
##  5      2 princess      1    13 0.0769 1.10  0.0845
##  6      1 child         1    14 0.0714 1.10  0.0785
##  7      1 have          1    14 0.0714 1.10  0.0785
##  8      1 who           1    14 0.0714 1.10  0.0785
##  9      1 wished        1    14 0.0714 1.10  0.0785
## 10      3 having        1    14 0.0714 1.10  0.0785
## 11      3 live          1    14 0.0714 1.10  0.0785
## 12      3 olden         1    14 0.0714 1.10  0.0785
## 13      3 the           1    14 0.0714 1.10  0.0785
## 14      3 used          1    14 0.0714 1.10  0.0785
## 15      3 when          1    14 0.0714 1.10  0.0785
## 16      3 wishing       1    14 0.0714 1.10  0.0785
## 17      2 in            1    13 0.0769 0.405 0.0312
## 18      2 once          1    13 0.0769 0.405 0.0312
## 19      2 upon          1    13 0.0769 0.405 0.0312
## 20      1 king          1    14 0.0714 0.405 0.0290
## 21      1 once          1    14 0.0714 0.405 0.0290
## 22      1 to            1    14 0.0714 0.405 0.0290
## 23      1 upon          1    14 0.0714 0.405 0.0290
## 24      1 was           1    14 0.0714 0.405 0.0290
## 25      3 in            1    14 0.0714 0.405 0.0290
## 26      3 king          1    14 0.0714 0.405 0.0290
## 27      3 to            1    14 0.0714 0.405 0.0290
## 28      3 was           1    14 0.0714 0.405 0.0290
## 29      1 a             3    14 0.214  0     0
## 30      2 a             3    13 0.231  0     0
## 31      1 there         1    14 0.0714 0     0
## 32      1 time          1    14 0.0714 0     0
## 33      2 there         1    13 0.0769 0     0
## 34      2 time          1    13 0.0769 0     0
## 35      3 a             1    14 0.0714 0     0
## 36      3 there         1    14 0.0714 0     0
## 37      3 time          1    14 0.0714 0     0
```

## Korpusy

Inny sposobem pracy na wielu dokumentach jest wykorzystanie idei **korpusu**, czyli po prostu zbioru

dokumentów. Taką możliwość daje pakiet **tm**. Aby stworzyć korpus, musimy podać źródło - w naszym przypadku będzie to ramka danych, więc skorzystamy z funkcji **DataframeSource()** (musimy jeszcze dokonac rzutowania tibble na data frame). Gdyby nasze dokumenty miały formę kolejnych wektorów, użylibysmy **VectorSource**. Zawartość korpusu obejrzymy przy pomocy funkcji **inspect()**.

```
# PRZYKŁAD 3.7
library(tm)
```

```
## Loading required package: NLP
```

```
df.source <- DataframeSource(as.data.frame(data))
corpus <- VCorpus(df.source)

inspect(corpus)
```

```
## <<VCorpus>>
## Metadata:  corpus specific: 0, document level (indexed): 0
## Content:  documents: 3
##
## [[1]]
## <<PlainTextDocument>>
## Metadata:  7
## Content:  chars: 61
##
## [[2]]
## <<PlainTextDocument>>
## Metadata:  7
## Content:  chars: 69
##
## [[3]]
## <<PlainTextDocument>>
## Metadata:  7
## Content:  chars: 70
```

Do pojedynczego dokumentu możemy odwoływać się za pomocą indeksu listy oraz odpowiednich pól.

```
# PRZYKŁAD 3.8
corpus[[2]]
```

```
## <<PlainTextDocument>>
## Metadata:  7
## Content:  chars: 69
```

```
corpus[[2]]$meta
```

```
##   author       : character(0)
##   datetimestamp: 2018-11-05 09:08:51
##   description  : character(0)
##   heading      : character(0)
##   id           : 2
##   language     : en
##   origin       : character(0)
```

```
corpus[[2]]$content
```

```
## [1] "Once upon a time, in a faraway land there lived a beautiful princess."
```

Biblioteka **tm** jest wyposażona w kilka przydatnych funkcji, w tym **tm_filter()**, która umozliwia np. wyszukanie określonych słów w poszczególnych dokumentach

```
# PRZYKŁAD 3.9
corpus %>%
  tm_index(FUN = function(x) any(grep("king", content(x))))
```

```
## [1]  TRUE FALSE FALSE
```

Inną wygodną funkcją jest **tm_map()**, która zastosuje wybrane transformcje do poszczególnych dokumentów, przy czym w przypadku dedykowanych funkcji (np. **reomvePunctation**) wystarczy podać je w treści funkcji **tm_map()**, natomiast dla własnych tworów potrzebujemy "ubrać" je w tzw **content_transformer()**.

```
# PRZYKŁAD 3.10
corpus1 <- corpus %>%
  tm_map(removePunctuation)

content(corpus1[[3]])
```

```
## [1] "In the olden time when wishing was having there used to live a King"
```

```
# PRZYKŁAD 3.11
corpus1 <- corpus %>%
  tm_map(content_transformer(tolower))

content(corpus1[[3]])
```

```
## [1] "in the olden time, when wishing was having, there used to live a king."
```

## Macierz termów-dokumentów

Bardzo wygodną reprezentacją dokumentu stanowi **macierz termów-dokumentów**. Jest to w pewnym sensie implementacja modelu *bag-of-words* lub (w zalażności od wykorzystania) modelu przestrzeni wektorowej. Każda kolumna macierzy odpowiada dokumentowi, natomiast każdy rząd - wyrazowi, otrzymanemu po procesie

tokenizacji.

```
# PRZYKŁAD 3.12
tdm <- TermDocumentMatrix(corpus)
tdm
```

```
## <<TermDocumentMatrix (terms: 24, documents: 3)>>
## Non-/sparse entries: 30/42
## Sparsity           : 58%
## Maximal term length: 9
## Weighting          : term frequency (tf)
```

```
inspect(tdm)
```

```
## <<TermDocumentMatrix (terms: 24, documents: 3)>>
## Non-/sparse entries: 30/42
## Sparsity           : 58%
## Maximal term length: 9
## Weighting          : term frequency (tf)
## Sample             :
##            Docs
## Terms       1 2 3
##   beautiful 0 1 0
##   child.    1 0 0
##   faraway   0 1 0
##   have      1 0 0
##   having,   0 0 1
##   once      1 1 0
##   there     1 1 1
##   time,     0 1 1
##   upon      1 1 0
##   was       1 0 1
```

Powyższe instrukcje pokazują jedynie część macierzy $tdm$, poza tym widać, że tokanizacja została przeprowadzona przy użyciu znaków interpunkcyjnych. Zanki usuwamy przekazując opcję **removePunctuation = TRUE**, natomiast obiekt rzutujemy na zwykłą macierz za pomocą **as.matrix()**.

```
# PRZYKŁAD 3.13
tdm <- TermDocumentMatrix(corpus, control = list(removePunctuation = TRUE))
tdm <- as.matrix(tdm)
tdm
```

```
##            Docs
## Terms       1 2 3
##   beautiful 0 1 0
##   child     1 0 0
##   faraway   0 1 0
##   have      1 0 0
##   having    0 0 1
##   king      1 0 1
##   land      0 1 0
##   live      0 0 1
##   lived     0 1 0
##   olden     0 0 1
##   once      1 1 0
##   princess  0 1 0
##   the       0 0 1
##   there     1 1 1
##   time      1 1 1
##   upon      1 1 0
##   used      0 0 1
##   was       1 0 1
##   when      0 0 1
##   who       1 0 0
##   wished    1 0 0
##   wishing   0 0 1
```

Aby jeszcze bardziej "obrobić" dokument, pozbywamy się słów kluczowych, a poza tym korzystamy z opcji do stemingu, co sprowadzi część wyrazów do wspólnych form.

```
# PRZYKŁAD 3.14
tdm <- TermDocumentMatrix(corpus, control = list(
  removePunctuation = TRUE, stopwords = TRUE, stemming = TRUE))
tdm <- as.matrix(tdm)
tdm
```

```
##           Docs
## Terms      1 2 3
##   beauti   0 1 0
##   child    1 0 0
##   faraway  0 1 0
##   king     1 0 1
##   land     0 1 0
##   live     0 1 1
##   olden    0 0 1
##   princess 0 1 0
##   time     1 1 1
##   upon     1 1 0
##   use      0 0 1
##   wish     1 0 1
```

I wreszcie, możemy również zastosowac transformację TF-IDF do naszej macierzy:

```
# PRZYKŁAD 3.15
tdm <- TermDocumentMatrix(corpus, control = list(weighting = weightTfIdf))
tdm
```

```
## <<TermDocumentMatrix (terms: 24, documents: 3)>>
## Non-/sparse entries: 27/45
## Sparsity           : 62%
## Maximal term length: 9
## Weighting          : term frequency - inverse document frequency (normalized) (tf-i
df)
```

```
as.matrix(tdm)
```

```
##            Docs
## Terms                 1          2          3
##   beautiful 0.00000000 0.17610694 0.00000000
##   child.    0.15849625 0.00000000 0.00000000
##   faraway   0.00000000 0.17610694 0.00000000
##   have      0.15849625 0.00000000 0.00000000
##   having,   0.00000000 0.00000000 0.14408750
##   king      0.15849625 0.00000000 0.00000000
##   king.     0.00000000 0.00000000 0.14408750
##   land      0.00000000 0.17610694 0.00000000
##   live      0.00000000 0.00000000 0.14408750
##   lived     0.00000000 0.17610694 0.00000000
##   olden     0.00000000 0.00000000 0.14408750
##   once      0.05849625 0.06499583 0.00000000
##   princess. 0.00000000 0.17610694 0.00000000
##   the       0.00000000 0.00000000 0.14408750
##   there     0.00000000 0.00000000 0.00000000
##   time      0.15849625 0.00000000 0.00000000
##   time,     0.00000000 0.06499583 0.05317841
##   upon      0.05849625 0.06499583 0.00000000
##   used      0.00000000 0.00000000 0.14408750
##   was       0.05849625 0.00000000 0.05317841
##   when      0.00000000 0.00000000 0.14408750
##   who       0.15849625 0.00000000 0.00000000
##   wished    0.15849625 0.00000000 0.00000000
##   wishing   0.00000000 0.00000000 0.14408750
```

W przypadku, gdybyśmy chcieli wyznaczyć iloczyn skalarny, związany wystepowaniem słów w poszczególnych dokumentach, wystarczy wykonać odpowiednie mnożenie macierzowe

```
# PRZYKŁAD 3.14
tdm <- TermDocumentMatrix(corpus, control = list(
  removePunctuation = TRUE, stopwords = TRUE, stemming = TRUE))
tdm <- as.matrix(tdm)

tdm[,1] %*% tdm
```

```
##       Docs
##        1 2 3
##   [1,] 5 2 3
```