Text Mining in R

Section: Exploratory Text Analysis

Nicolas Pröllochs
University of Giessen
nicolas.proellochs@wi.jlug.de
Agenda

1. **Exploratory text analysis:** Learn how to gain an initial understanding of text data

2. **Tidy text analysis:** Learn how to perform text analysis in a “tidy” way using `tidytext`

3. **Corpus analysis:** Understand how to explore text corpora and perform tf-idf document weighting in R
Exploratory text analysis

▶ Text mining
  ▶ Extracting relevant information or knowledge from text data
  ▶ Not always sure what we are looking for (until we find it)!

▶ Exploratory text analysis
  ▶ Gain an initial understanding of the text data
  ▶ Clean and preprocess the texts
  ▶ Identify patterns and data characteristics

💡 Exploratory text analysis serves as a first step towards further statistical analysis (e.g. sentiment analysis, text classification, ...)

Text Mining in R
Working with text

▶ Text data can come from **various sources**:  
  ▶ Websites  
  ▶ Books  
  ▶ Social media  
  ▶ Databases  
  ▶ Digital scans of printed materials  
  ▶ …  
  ▶ Typically in **unstructured format** (data without a pre-defined data model)

💡 Approximately 90% of the world’s data is held in unstructured formats (Source: Oracle)
The Economy is doing really well. The Federal Reserve can easily make it Record Setting! The question is being asked, why are we paying much more in interest than Germany and certain other countries? Be early (for a change), not late. Let America win big, rather than just win!
Text data

- **Texts** are stored as raw character strings
- Text string contains **tokens**, which is a semantically meaningful unit of text
- Tokens can be words, sentences, paragraphs, etc.
- Example: Peter Pan by J. M. Barrie

<table>
<thead>
<tr>
<th>Token type</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Documents</td>
<td>1</td>
</tr>
<tr>
<td>Paragraphs</td>
<td>4464</td>
</tr>
<tr>
<td>Sentences</td>
<td>6044</td>
</tr>
<tr>
<td>Words</td>
<td>47707</td>
</tr>
</tbody>
</table>

Need to transform the raw string into tokens to perform meaningful text analysis
Tidytext R-package

▶ Contains tidy tools for quantitative text analysis, including tokenization, basic text summarization, sentiment analysis, and text modeling

▶ Load necessary libraries tidyverse and tidytext to do text analysis

```r
library(tidyverse)
library(tidytext)
```
Tidy data

▶ Tidy format
  ▶ Each variable forms a column
  ▶ Each observation forms a row
  ▶ Each type of observational unit forms a table

▶ Why?
  ▶ **Standardized** consistent data structure
  ▶ Makes it **easier to manipulate, model and visualize data**

💡 We can easily switch between tidy format and other formats if needed
Tokenization

- Need to **organize text data around tokens**
  - If the data contains whole documents as a variable and the tokens are words, the data isn’t *tidy*

- Common steps before text analysis
  - Split on white space/punctuation
  - Make lower case
  - Handling abbreviations
  - Maybe put named entities together
  - …

Tokenization is the process of segmenting running text into a list of tokens (e.g. words or sentences)
Creating some text data

Creating a text corpus consisting of three documents

txt <- c("These are words", "so are these", "this is running on")
document <- c(1, 2, 3)
dat <- tibble(txt, document)
dat

```
# A tibble: 3 x 2
#  txt document
#  <chr>    <dbl>
#1 These are words   1
#2 so are these    2
#3 this is running on 3
```
The `unnest_tokens()` function

▶ The function `unnest_tokens(tbl, output, input, ...)` converts a text column of a dataframe into tokens

▶ Parameters:
  ▶ `tbl` is a dataframe
  ▶ `output` is the name of the column to be generated
  ▶ `input` is the column in the dataframe that gets split

```
unnest_tokens(tbl = dat, output = "tok", input = txt)
```

`unnest_tokens()` supports tokenization of words (default), sentences, ngrams, characters, and regular expressions
Example: `unnest_tokens()`

```r
## A tibble: 10 x 2
## document tok
## <dbl> <chr>
## 1 1 these
## 2 1 are
## 3 1 words
## 4 2 so
## 5 2 are
## # ... with 5 more rows
```

- One-token-per-row format
- Punctuation has been stripped
- Words have been converted to lowercase

Our text is **tidy** now!
Gathering more data

- **Project Gutenberg** provides access to the full text of many public domain works
- **Access the library via the gutenbergr R-package**

```r
library(gutenbergr)
gutenberg_metadata %>%
  filter(author == "Shakespeare, William")
```

```
## # A tibble: 317 x 8
## #  gutenberg_id title author gutenberg_author  language gutenberg_books
## #   <int> <chr> <chr> <chr> <chr>      <chr> <chr>
## 1     100  The ~ Shake~ Shakespeare, William en Plays
## 2     1041 Shak~ Shake~ Shakespeare, William en <NA>
## 3     1045 Venu~ Shake~ Shakespeare, William en <NA>
## 4     1100 The ~ Shake~ Shakespeare, William en <NA>
## 5     1101 The ~ Shake~ Shakespeare, William en <NA>
## # ... with 312 more rows, and 2 more variables: rights <chr>,
## #  has_text <lgl>
```
Gathering more data

- Download book 5314 (“Household Tales by Brothers Grimm”)

```r
full_text <- gutenberg_download(5314)
```

- Take a glimpse at the book via `slice(rows)`

```r
full_text %>% slice(1000:1005)
```

```r
# A tibble: 6 x 2
#  gutenberg_id text                       
#   <int>     <chr>
# 1  5314 "What rumbles and tumbles"
# 2  5314 "Against my poor bones?"
# 3  5314 "I thought 't was six kids,"
# 4  5314 "But it's naught but big stones.""
# 5  5314 ""
# ... with 1 more row
```
Time to tidy your text!

▶ Word tokenization using `unnest_tokens()`

```r
# tidy_book <- full_text %>%
#   unnest_tokens(word, text)
# tidy_book

## # A tibble: 287,073 x 2
##   gutenberg_id word
##       <int> <chr>
## 1       5314  the
## 2       5314   e
## 3       5314  book
## 4       5314  was
## 5       5314 prepared
## # ... with 2.87e+05 more rows
```

▶ The book contains 287,073 words
What are the most common words?

- Calculate word counts via `count(var)`
- Output is a new column `n`
- If `sort = TRUE`, the output is sorted in descending order

```r
tidy_book %>%
  count(word, sort = TRUE)
```

```
## # A tibble: 8,288 x 2
## #  word   n
## <chr> <int>
## 1 the   20176
## 2 and   14740
## 3 to    7454
## 4 he    5954
## 5 a     5436
## # ... with 8,283 more rows
```

Most common words are non-characteristic terms without a deeper meaning.
Stopwords

- **Stopwords** are short function words occurring frequently but with no deep meaning.
- Removal of stopwords in order to concentrate on more important words (that are specific to the text).
- Common approach is to use predefined list of stopwords (Examples: the, is, at, which, and).
- Get such a built-in list via `get_stopwords()`.

```r
get_stopwords()
```

```
## # A tibble: 175 x 2
## #  word   lexicon
## #  <chr> <chr>
## 1 i     snowball
## 2 me    snowball
## 3 my    snowball
## 4 myself snowball
## 5 we    snowball
## # ... with 170 more rows
```
Filtering stopwords

- Filter stopwords via `anti_join()`

```r
 tidy_book %>%
    anti_join(get_stopwords()) %>%
    count(word, sort = TRUE)
```

```r
# A tibble: 8,141 x 2
#  word n
#  <chr> <int>
# 1 said 3025
# 2 thou 1525
# 3 one 1369
# 4 went 1181
# 5 came 1044
# ... with 8,136 more rows
```
Handling contractions

- The function `unnest_tokens()` does not replace contractions

```r
tidy_book %>% filter(word == "can't")
```

```
# A tibble: 26 x 2
## gutenberg_id word
## <int> <chr>
## 1 5314 can't
## 2 5314 can't
## 3 5314 can't
## 4 5314 can't
## 5 5314 can't
## # ... with 21 more rows
```

- Load package `textclean`

```r
library(textclean)
```
Example: Handling contractions

Example: Replace contractions using `replace_contraction()`

```r
text <- "I'll go home"
replace_contraction(text, contraction.key = lexicon::key_contractions)
```

```r
## [1] "I will go home"
```

`replace_contraction()` uses a predefined list of contractions

```r
head(lexicon::key_contractions, 5)
```

```r
## contraction expanded
## 1 'cause because
## 2 'tis it is
## 3 'twas it was
## 4 ain't am not
## 5 aren't are not
```
Handling contractions

- Replace contractions in our book

```r
tidy_book <- full_text %>%
  mutate(text = replace_contraction(text)) %>%
  unnest_tokens(word, text)

tidy_book %>%
  filter(word == "can't")
```

```r
## # A tibble: 0 x 2
## # ... with 2 variables: gutenberg_id <int>, word <chr>
```

💡 The `textclean` package provides additional functions for replacing dates, emojis, emoticons, etc.
Word clouds

The `wordcloud` R-package allows one to easily visualize the most common words in a word cloud.

```r
library(wordcloud)

wc_data <- tidy_book %>%
  anti_join(stop_words) %>%
  count(word)
wordcloud(wc_data$word, wc_data$n, max.words = 100)
```
Zipf’s law

Zipf’s law states that the frequency that a word appears is inversely proportional to its rank.

term_freq <- tidy_book %>%
  count(word, sort = TRUE) %>%
  mutate(TotalWords = sum(n),
         rank = row_number(),
         tf = n / TotalWords)

term_freq

## # A tibble: 8,267 x 5
## word     n TotalWords  rank    tf
## <chr> <int>     <int> <int> <dbl>
## 1 the   20176    287331    1 0.0702
## 2 and   14740    287331    2 0.0513
## 3 to    7454     287331    3 0.0259
## 4 he    5955     287331    4 0.0207
## 5 a     5436     287331    5 0.0189
## # ... with 8,262 more rows
Zipf’s law

- We indeed observe a constant, negative slope indicating an inversely proportional relationship

```r
term_freq %>%
  ggplot(aes(rank, tf)) +
  geom_line(size = 1.1, show.legend = FALSE) +
  scale_x_log10() +
  scale_y_log10()
```
Analysis text corpora

- A text corpus is a **structured set of texts** (e.g. a collection of articles)

- Example: Loading a collection of physics classics

```r
# load a few classics:

## # A tibble: 4 x 8
## #  gutenberg_id title author gutenberg_author__ language gutenberg_books~
## #<int>     <chr> <chr>     <int>     <chr> <chr>           
##     1 7333  Side~ Einst~     1630    en <NA>
##     2 13476 "Exp~ Tesla~     5067    en <NA>
##     3 14725 "Tre~ Huyge~     5648    en <NA>
##     4 37729 A Di~ Galil~     39014   en <NA>
## ... # ... with 2 more variables: rights <chr>, has_text <lgl>

# load:

physics <- gutenberg_download(c(37729, 14725, 13476, 7333), meta_fields = "author")
```
Show the most frequent words

Show the most frequent words

```r
physics_words <- physics %>%
  unnest_tokens(word, text) %>%
  count(author, word, sort = TRUE) %>% print()
```

```
# A tibble: 11,219 x 3
#  author       word n
#  <chr>        <chr> <int>
# 1 Galilei, Galileo the 3760
# 2 Tesla, Nikola the 3604
# 3 Huygens, Christiaan the 3553
# 4 Galilei, Galileo of 2049
# 5 Tesla, Nikola of 1737
# ... with 1.12e+04 more rows
```

How can we find the words that are most characteristic for each document?
How can we find characteristic words?

- Example: *relativity* and *the* in the document from Albert Einstein

<table>
<thead>
<tr>
<th>Word</th>
<th>Document Frequency</th>
<th>Corpus Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>relativity</td>
<td>31</td>
<td>31</td>
</tr>
<tr>
<td>the</td>
<td>694</td>
<td>11611</td>
</tr>
</tbody>
</table>

- Rare word *relativity*
  - Document containing this term is very likely to be relevant to Albert Einstein
  - High weight for rare terms like *relativity*

- Common word *the*
  - Document containing this term can be about anything
  - Very low weight for common terms like *the*

- Idea: Need a numerical **measure** that reflects how **important** a word is to a **document** in a **corpus**
Tf-idf weighting

- Best known weighting scheme in information retrieval
- Increases with the number of occurrences of a term in a document
- Increases with the rarity of the term in the collection

Calculation

1. Term Frequency (tf): Number of times a term $t$ occurs in a document $d$
2. Document Frequency (df): Number of documents $d$ that contain each term $t$
3. Inverse Document Frequency $idf = \log(N/df)$, where $N$ is the total number of documents
4. Term frequency–inverse document frequency $tf – idf = tf \times idf$

Tf-idf weighting is used frequently by search engines
The `bind_tf_idf` function

- `bind_tf_idf(tbl, term, document, n)` adds tf-idf values to a tidy text dataset.

**Parameters:**

- `tbl` is a tidy text dataset with one-row-per-term-per-document.
- `term` is the column containing the terms (`word` in this case).
- `document` is the column containing the document IDs (`author` in this case).
- `n` is the column containing document-term counts (`n` in this case).

**Add a column tf-idf using the `bind_tf_idf()` function**

```r
physics_words %>%
  bind_tf_idf(word, author, n) %>%
  arrange(desc(tf_idf))
```
### The bind_tf_idf function

```r
# A tibble: 11,219 x 6
#  author       word n  tf  idf tf_idf
#  <chr>        <chr> <int> <dbl> <dbl> <dbl>
#1  Huygens, Christiaan  refraction  218 0.00569 1.39  0.00789
#2  Tesla, Nikola      bulb    171 0.00433 1.39  0.00600
#3  Galilei, Galileo   water   828 0.0206  0.288 0.00593
#4  Tesla, Nikola      coil    166 0.00420 1.39  0.00583
#5  Einstein, Albert   theory  67  0.00774  0.693 0.00536
# ... with 1.12e+04 more rows
```

- Top tf-idf values are intuitively very characteristic to the authors.

💡 Top tf-idf values are not affected by stop words as IDF values of such stop words are very small (due to their presence in almost every document).
Visualizing the results

- Plotting the 5 most characteristic words per author using `ggplot`

```
plot_physics <- physics_words %>%
  bind_tf_idf(word, author, n) %>%
  group_by(author) %>%
  top_n(5, tf_idf) %>%
  ungroup()

ggplot(plot_physics, aes(reorder(word, tf_idf), tf_idf, fill = author)) +
geom_col(show.legend = FALSE) + labs(x = NULL, y = "tf-idf") +
facet_wrap(~ author, ncol = 4, scales = "free") + coord_flip()
```
Wrap-up

► Key takeaways
  ► **Text data** typically comes in **unstructured format**
  ► **Exploratory text analysis** allows one to gain an **initial understanding** of the data
  ► The **tidytext R-Package** provides tools to perform **exploratory text analysis** in a “tidy” way

► Advanced topics
  ► Sentiment analysis
  ► Topic modeling
  ► Text classification & text-based forecasting

► Further reading
  ► Book: Text Mining with R (O’Reilly, 2017, by J. Silge & D. Robinson)