Text Analysis Using R

Kenneth Benoit Stefan Müller Kohei Watanabe

Table of Contents

About the Authors

Kenneth Benoit is currently Professor of Computational Social Science in the Department of Methodology at the London School of Economics and Political Science. He has previously held positions in the Department of Political Science at Trinity College Dublin and at the Central European University (Budapest). He received his Ph.D. (1998) from Harvard University, Department of Government. His current research focuses on computational, quantitative methods for processing large amounts of textual data, mainly political texts and social media.

Stefan Müller is an Assistant Professor and Ad Astra Fellow in the School of Politics and International Relations at University College Dublin. Previously, he was a Senior Researcher at the University of Zurich. He received his Ph.D. (2019) from Trinity College Dublin, Department of Political Science. His current research focuses on political representation, party competition, political communication, public opinion, and quantitative text analysis. His work has been published in journals such as the *American Political Science Review*, *The Journal of Politics*, *Political Communication*, the *European Journal of Political Research*, and *Political Science Research and Methods*, among others.

Kohei Watanabe is xxx at Lazard Asset Management. Before returning to Tokyo, he was a research officer at the Department of Methodology and the Department of International Relations of the London School of Economics and Political Science. He is also a member of a project on popular mobilization in Russia at the LSE's International Relations Department and a co-author of the **quanteda** quantitative text analysis package for R. He holds a PhD in Social Research Methods from the LSE and an MA in Political Science (Central European University).

The *Quanteda Initiative* is a non-profit company we founded in 2018 in London, devoted to the promotion of open-source text analysis software. It supports active development of these tools, in addition to providing training materials, training events, and sponsoring workshops and conferences. Its main product is the open-source **quanteda** package for R, but the Quanteda Initiative also supports a family of independent but interrelated packages for providing additional functionality for natural language processing and document management.

Its core objectives, as stated in its charter, are to:

- Support open-source, text analysis software developed for research and scientific analysis. These efforts focus mainly on the open-source software library

quanteda, written for the R programming language, and its related family of extension packages.

- Promote interoperability between text analytic software libraries, including those written in other languages such as Python, Java, or C++.
- Provide ongoing user, technical, and development support for open-source text analysis software. Organize training and dissemination activities related to open-source text analysis software.

Preface

The uses and means for analysing text, especially using quantitative and computational approaches, have exploded in recent years across the fields of academic, industry, policy, and other forms of research and analysis. Text mining and text analysis have become the focus of a major methodological wave of innovation in the fields of political science in particular, but also extending to finance, media and communications, and sociology.

Text analysis is a broad label for a wide range of tools applied to *textual* data, a form of structured data that is created from typically unstructured raw data in the form of natural language documents. It includes descriptive analysis, keyword analysis, topic analysis, measurement and scaling, clustering, text and vocabulary comparisons, or sentiment analysis. It may include causal analysis or predictive modelling. In its most advanced current forms, it extends to the natural language generation models that power the newest generation of artificial intelligence systems.

This book provides a practical guide introducing the fundamentals of text analysis methods and how to implement them using the R programming language. It covers a wide range of topics to provide a useful resource for a range of students from complete beginners to experienced users of R wanting to learn more advanced techniques.

Why R (and another book on text mining in R)?

This book is intended for a specific audience: those wishing to apply the methods of quantitative analysis and natural language processing text using the R language. Excellent alternatives exist, especially those in Python. So why use R for such a task?

Text mining tools for R have existed for many years, led by the venerable **tm** package (Feinerer, Hornik, and Meyer 2008). In 2015, the first version of **quanteda** was published on CRAN. Since then, the package has undergone three major versions, with each release improving its consistency, power, and usability. Starting with version 2, **quanteda** split into a series of packages designed around its different functions (such as plotting, statistics, or machine learning), with **quanteda** retaining the core text processing functions. Other popular alternatives exist, such as **tidytext**, although as we explain in [further-tidy], **quanteda** works perfectly well with tidyverse and other tidytext-like approaches.

Why another book about text analysis using R?

Why use R for this? When perfectly adequate packages exist for Python?

(This might belong in the Introduction instead.)

And why **quanteda**?

What to Expect from This Book

This book is meant to be as a practical resource for those confronting the practical challenges of text analysis for the first time, focusing on how to do this in R. Our main focus is on the **quanteda** package (Benoit et al. 2018) and its extensions, although we also cover more general issues including a brief overview of the R functions required to get started quickly with practical text analysis We cover an introduction to the R language, for

> **Note**
>
> Many text analysis packages exist for the R programming language, but here our focus is on the **quanteda** package and its extensions. "quanteda" is a portmanteau name indicating the purpose of the package, which is the quantitative analysis of textual data.

Benoit (2020) provides a detailed overview of the analysis of textual data, and what distinguishes textual data from other forms of data. It also clearly articulates what is meant by treating text "as data" for analysis. This book and the approaches it presents are firmly geared toward this mindset. All of three of the authors are academics, although we also have experience working in industry or in applying text analysis for non-academic purposes. The typical reader may be a student of text analysis in the literal sense (of being an student) or in the general sense of someone studying techniques in order to improve their practical and conceptual knowledge.

Each chapter is structured so to provide a continuity across each topic. For each main subject explained in a chapter, we clearly explain the objective of the chapter, then describe the text analytic methods in an applied fashion so that readers are aware of the workings of the method. We then provide practical examples, with detailed working code in R as to how to implement the method. Next, we identify any special issues involved in correctly applying the method, including how to hand the more complicated situations that may arise in practice. Finally, we provide further reading for readers wishing to learn more, and exercises for those wishing for hands-on practice, or for assigning these when using them in teaching environment. Our goal is to make the book is suitable for self-learning or to form the basis for teaching and learning in a course on applied text analysis.

Who This Book Is For

Prerequisites. We don't assume any previous knowledge of text mining and very little knowledge of R, although it will help to start with some. Resources for complete beginners that we recommend include: …

Our orientation as social scientists, with a specialization in political text and political communications and media. But this book is for everyone: social scientists, computer scientists, scholars of digital humanities, researchers in marketing and management, and applied researchers working in policy, government, or business fields.

How to use the book depending on where the reader fits within the continuum of R experience and prior experience with text analysis.

How to Use This Book

Chapters Outline

Outline

The book is divided into seven sections. These group topics that we feel represent common stages of learning in text analysis, or similar groups of topics that different users will be drawn too. By grouping stages of learning, we make it possible also for intermediate or advanced users to jump to the section that interests them most, or to the sections where they feel they need additional learning.

Our sections are:

- **(Working in R):** This section is designed for beginners to learn quickly the R required for the techniques we cover in the book, and to guide them in learning a proper R workflow for text analysis.
- **Acquiring texts:** Often described (by us at least) as the hardest problem in text analysis, we cover how to source documents, including from Internet sources, and to import these into R as part of the quantitative text analysis pipeline.
- **Managing textual data using quanteda:** In this section, we introduce the **quanteda** package, and cover each stage of textual data processing, from creating structured corpora, to tokenisation, and building matrix representations of these tokens. WE also talk about how to build and manage structured lexical resources such as dictionaries and stop word lists.

- **Exploring and describing texts:** How to get overviews of texts using summary statistics, exploring texts using keywords-in-context, extracting target words, and identifying key words.
- **Statistics for comparing texts:** How to characterise documents in terms of their lexical diversity. readability, similarity, or distance.
- **Machine learning for texts:** How to apply scaling models, predictive models, and classification models to textual matrices.
- **Further methods for texts:** Advanced methods including the use of natural language models to annotate texts, extract entities, or use word embeddings; integrating **quanteda** with "tidy" data approaches; and how to apply text analysis to "hard" languages such as Chinese (hard because of the high dimensional character set and the lack of whitespace to delimit words).

Finally, in several **appendices**, we provide more detail about some tricky subjects, such as text encoding formats and working with regular expressions.

Chapter Structure

Our approach in each chapter is split into the following components, which we apply in every chapter:

- Objectives. We explain the purpose of each chapter and what we believe are the most important learning outcomes.
- Methods. We clearly explain the methodological elements of each chapter, through a combination of high-level explanations, formulas, and references.
- Examples. We use practical examples, with R code, demonstrating how to apply the methods to realise the objectives.
- Issues. We identify any special issues, potential problems, or additional approaches that a user might face when applying the methods to their text analysis problem.
- Further Reading. In part because our scholarly backgrounds compel us to do so, and in part because we know that many readers will want to read more about each method, each chapter contains its own set of references and further readings.
- Exercises. For those wishing additional practice or to use this text as a teaching resource (which we strongly encourage!), we provide exercises that can be assigned for each chapter.

Throughout the book, we will demonstrate with examples and build models using a selection of text data sets. A description of these data sets can be found in Appendix @ref(appendixdata).

We use three kinds of info boxes throughout the book to invite attention to notes and other ideas.

> **Note**
>
> The first type of boxes provides notes, information on best practices, and links to further information.

> **Warning**
>
> Some boxes call out warnings or possible problems to watch out for.

> **Important**
>
> Boxes marked with hexagons highlight information about specific R packages and how they are used. We use **bold** for the names of R packages.

Data used in examples

All examples and code are bundled as a companion R package to the book, available from our public GitHub repository.

We largely rely on data from three sources:

- the built-in-objects from the **quanteda** package, such as the US Presidential Inaugural speech corpus;
- additional **quanteda** corpora from the **quanteda.corpora** package; and
- added corpora from the book's companion package.

Although not commonly used, our scheme for naming data follows a very consistent scheme. The data objects being with `data`, have the object class as the second part of the name, such as `corpus`, and the third and final part of the data object name contains a description. The three elements are separated by the underscore (_) character. This means that any object is known by its name to be data, so that it shows up in the index of package objects (from the all-important help page, e.g. from `help(package = "quanteda")`) in one location, under "d". It also means that its object class is known from the name, without further inspection. So `data_dfm_lbgexample` is a dfm, while `data_corpus_inaugural` is clearly a corpus. We use this scheme and

others like with an almost religious fervour, because we think that learning the functionality of a programming framework for NLP and quantitative text analysis is complicated enough without having also to decipher or remember a mishmash of haphazard and inconsistently named functions and objects. The more you use our software packages and specifically **quanteda**, the more you will come to appreciate the attention we have paid to implementing a consistent naming scheme for objects, functions, and their arguments as well as to their consistent functionality.

How to Contact Us

Please address comments and questions concerning this book by filing an issue on our GitHub page, https://github.com/quanteda/Text-Analysis-Using-R/issues/.

At this repository, you will also find instructions for installing the companion R package, **TAURbook**.

For more information about the authors or the Quanteda Initiative, visit our website.

Acknowledgements

We are so thankful for everything. Especially the ERC. Gush, gush.

Colophon

This book was written in RStudio using **Quarto**. The website is hosted via GitHub Pages, and the complete source is available on GitHub.

This version of the book was built with R version 4.2.1 (2022-06-23) and the following packages:

Package	Version	Source
quanteda	3.2.1	CRAN (R 4.2.0)
quanteda.textmodels	0.9.4	CRAN (R 4.2.0)
quanteda.textplots	0.94.1	CRAN (R 4.2.0)
quanteda.textstats	0.95	CRAN (R 4.2.0)
readtext	0.81	CRAN (R 4.2.0)
stopwords	2.3	CRAN (R 4.2.0)
tibble	3.1.7	CRAN (R 4.2.0)
tidyverse	1.3.1	CRAN (R 4.2.0)

1 Introduction and Motivation

1.1 Objectives

To dive right in to text analysis using R, by means of an extended demonstration of how to acquire, process, quantify, and analyze a series of texts. The objective is to demonstrate the techniques covered in the book and the tools required to apply them.

This overview is intended to serve more as a demonstration of the kinds of natural language processing and text analysis that can be done powerfully and easily in R, rather than as primary instructional material. Starting in Chapter @ref(rbasics), you learn about the R fundamentals and work our way gradually up to basic and then more advanced text analysis. Until then, enjoy the demonstration, and don't be discouraged if it seems too advanced to follow at this stage. By the time you have finished this book, this sort of analysis will be very familiar.

1.2 Application: Analyzing Candidate Debates from the US Presidential Election Campaign of 2020

The methods demonstrated in this chapter include:

- acquiring texts directly from the world-wide web;
- creating a corpus from the texts, with associated document-level variables;
- segmenting the texts by sections found in each document;
- cleaning up the texts further;
- tokenizing the texts into words;
- summarizing the texts in terms of who spoke and how much;
- examining keywords-in-context from the texts, and identifying keywords using statistical association measures;
- transforming and selecting features using lower-casing, stemming, and removing punctuation and numbers;
- removing selected features in the form of "stop words";
- creating a document-feature matrix from the tokens;
- performing sentiment analysis on the texts;
- fitting topic models on the texts.

For demonstration, we will use the corpus of televised debate transcripts from the U.S. Presidential election campaign of 2020. Donald Trump and Joe Biden participated in two televized debates. The first debate took place in Cleveland, Ohio, on 29 September 2020. The two candidates met again in Nashville, Tennessee on 10 December.[1]

1.2.1 Acquiring text directly from the world wide web.

Full transcripts of both televized debates are available from The American Presidency Project. We start our demonstration by scraping the debates using the **rvest** package (Wickham 2021). You will learn more about scraping data from the internet in Chapter @ref(acquire-internet). We first need to install and load the packages required for this demonstration.

```
# install packages
install.packages("quanteda")
install.packages("quanteda.textstats")
install.packages("quanteda.textplots")
install.packages("rvest")
install.packages("stringr")
install.packages("devtools")
devtools::install_github("quanteda/quanteda.tidy")
```

```
# load packages
library("quanteda")
library("rvest")
library("stringr")
library("quanteda.textstats")
library("quanteda.textplots")
library("quanteda.tidy")
```

Next, we identify the presidential debates for the 2020 period, assign the URL of the debates to an object, and load the source page. We retrieve metadata (the location and dates of the debates), store the information as a data frame, and get the URL debates. Finally, we scrape the search results and save the texts as a character vector, with one element per debate.

[1]The transcripts are available at https://www.presidency.ucsb.edu/documents/presidential-debate-case-western-reserve-university-cleveland-ohio and https://www.presidency.ucsb.edu/documents/presidential-debate-belmont-university-nashville-tennessee-0.

```r
# search presidential debates for the 2020 period
# https://www.presidency.ucsb.edu/advanced-search?field-keywords=&field-keywords2=&field-keywords3

# assign URL of debates to an object called url_debates
url_debates <- "https://www.presidency.ucsb.edu/advanced-search?field-keywords=&field-keywords2=&f

source_page <- read_html(url_debates)

# get debate meta-data

nodes_pres <- ".views-field-title a"
text_pres <- ".views-field-field-docs-start-date-time-value.text-nowrap"

debates_meta <- data.frame(
    location = html_text(html_nodes(source_page, nodes_pres)),
    date =  html_text(html_nodes(source_page, text_pres)),
    stringsAsFactors = FALSE
)

# format the date
debates_meta$date <- as.Date(trimws(debates_meta$date),
                             format = "%b %d, %Y")

# get debate URLs
debates_links <- source_page |>
    html_nodes(".views-field-title a") |>
    html_attr(name = "href")

# add first part of URL to debate links
debates_links <- paste0("https://www.presidency.ucsb.edu", debates_links)

# scrape search results
debates_scraped <- lapply(debates_links, read_html)

# get character vector, one element per debate
debates_text <- sapply(debates_scraped, function(x) {
    html_nodes(x, "p") |>
        html_text() |>
        paste(collapse = "\n\n")
})
```

Having retrieved the text of the two debates, we clean up the character vector. More specifically, we clean up the details on the location using regular expressions and

the **stringr** package (see Chapter @ref(appendix-regex)).

```
debates_meta$location <- str_replace(debates_meta$location, "^.* in ", "")
```

1.2.2 Creating a text corpus

Now we have two objects. The character vector `debates_text` contains the text of both debates, and the data frame `debates_meta` stores the date and location of both debates. These objects allow us to create a **quanteda** corpus, with the metadata. We use the `corpus()` function, specify the location of the document-level variables, and assign the corpus to an object called `data_corpus_debates`.

```
data_corpus_debates <- corpus(debates_text,
                              docvars = debates_meta)

# check the number of documents included in the text corpus
ndoc(data_corpus_debates)
```

```
[1] 2
```

The object `data_corpus_debates` contains two documents, one for each debate. While this unit of analysis may be suitable for some analyses, we want to identify all utterances by the moderator and the two candidates. With the function `corpus_segment()`, we can segment the corpus into statements. The unit of analysis changes from a full debate to a statement during a debate. Inspecting the page for one of the debates[2] reveals that a new utterance starts with the speaker's name in ALL CAPS, followed by a colon. We use this consistent pattern for segmenting the corpus to the level of utterances. The regular expression `"\\s*[[:upper:]]+:\\s+"` identifies speaker names in ALL CAPS (`\\s*[[:upper:]]+`), followed by a colon `+:` and a white space `\\s+`.

```
data_corpus_debatesseg <- corpus_segment(data_corpus_debates,
                                         pattern =  "\\s*[[:upper:]]+:\\s+",
                                         valuetype = "regex",
                                         case_insensitive = FALSE)

summary(data_corpus_debatesseg, n = 4)
```

```
Corpus consisting of 1207 documents, showing 4 documents:
```

[2] https://www.presidency.ucsb.edu/documents/presidential-debate-case-western-reserve-university-cleveland-ohio

```
   Text Types Tokens Sentences        location       date       pattern
text1.1   139    251        13 Cleveland, Ohio 2020-09-29 \n\nWALLACE:
text1.2     6      6         1 Cleveland, Ohio 2020-09-29   \n\nBIDEN:
text1.3     5      5         1 Cleveland, Ohio 2020-09-29   \n\nTRUMP:
text1.4     3      3         1 Cleveland, Ohio 2020-09-29   \n\nBIDEN:
```

The segmentation results in a new corpus consisting of 1,207 utterances. The `summary()` function provides a useful overview of each utterance. The first text, for example, contains 251 tokens, 139 types (i.e., unique tokens) and 13 sentences.

Having segmented the corpus, we improve the document-level variables since such meta information on each document is crucial for subsequent analyses like subsetting the corpus or grouping the corpus to the level of speakers. We create a new document-level variable called "speaker" based on the pattern extracted above.

```
# str_trim() removes empty whitespaces,
# str_remove_all() removes the colons,
# and str_to_title() changes speaker names
# from UPPER CASE to Title Case

data_corpus_debatesseg <- data_corpus_debatesseg |>
    rename(speaker = pattern) |>
    mutate(speaker = str_trim(speaker),
           speaker = str_remove_all(speaker, ":"),
           speaker = str_to_title(speaker))

# cross-table of speaker statements by debate
table(data_corpus_debatesseg$speaker,
      data_corpus_debatesseg$location)
```

```
        Cleveland, Ohio Nashville, Tennessee
Biden               269                  84
Trump               340                 122
Wallace             246                   0
Welker                0                 146
```

The cross-table reports the number of statement by each speaker in each debate. The first debate in Cleveland seems to be longer: the number of Trump's and Biden's statements during the Cleveland debates are three times higher than those in Tennessee. The transcript reports 246 utterances by Chris Wallace during the first

and 146 by Kristen Welker during the second debate. We can inspect all cleaned document-level variables in this text corpus.

```
data_corpus_debatesseg |>
    docvars() |>
    glimpse()
```

```
Rows: 1,207
Columns: 3
$ location <chr> "Cleveland, Ohio", "Cleveland, Ohio", "Cleveland, Ohio", "Cle~
$ date     <date> 2020-09-29, 2020-09-29, 2020-09-29, 2020-09-29, 2020-09-29, ~
$ speaker  <chr> "Wallace", "Biden", "Trump", "Biden", "Wallace", "Trump", "Wa~
```

1.2.3 Tokenizing a corpus

Next, we tokenize our text corpus. Typically, tokenization involves separating texts by white spaces. We tokenize the text corpus without any pre-processing using `tokens()`.

```
toks_usdebates2020 <- tokens(data_corpus_debatesseg)

# let's inspect the first six tokens of the first four documents
print(toks_usdebates2020, max_ndoc = 4, max_ntoken = 6)
```

```
Tokens consisting of 1,207 documents and 3 docvars.
text1.1 :
[1] "Good"      "evening"   "from"      "the"       "Health"    "Education"
[ ... and 245 more ]

text1.2 :
[1] "How"   "you"   "doing" ","     "man"   "?"

text1.3 :
[1] "How"   "are"   "you"   "doing" "?"

text1.4 :
[1] "I'm"  "well" "."

[ reached max_ndoc ... 1,203 more documents ]
```

```
tokens(data_corpus_debatesseg)
```

```
Tokens consisting of 1,207 documents and 3 docvars.
text1.1 :
 [1] "Good"       "evening"    "from"       "the"        "Health"
 [6] "Education"  "Campus"     "of"         "Case"       "Western"
[11] "Reserve"    "University"
[ ... and 239 more ]

text1.2 :
[1] "How"   "you"   "doing" ","     "man"   "?"

text1.3 :
[1] "How"   "are"   "you"   "doing" "?"

text1.4 :
[1] "I'm"  "well" "."

text1.5 :
 [1] "Gentlemen" ","         "a"         "lot"       "of"        "people"
 [7] "been"      "waiting"   "for"       "this"      "night"     ","
[ ... and 137 more ]

text1.6 :
 [1] "Thank" "you"   "very"  "much"  ","     "Chris" "."     "I"     "will"
[10] "tell"  "you"   "very"
[ ... and 282 more ]

[ reached max_ndoc ... 1,201 more documents ]
```

```
# check number of tokens and types
toks_usdebates2020 |>
    ntoken() |>
    sum()
```

```
[1] 43742
```

```
toks_usdebates2020 |>
    ntype() |>
    sum()
```

```
[1] 28174
```

Without any pre-processing, the corpus consists of 43,742 tokens and 28,174 types. We can easily check how these numbers change when transforming all tokens to lowercase and removing punctuation characters.

```
toks_usdebates2020_reduced <- toks_usdebates2020 |>
    tokens(remove_punct = TRUE) |>
    tokens_tolower()

# check number of tokens and types
toks_usdebates2020_reduced |>
    ntoken() |>
    sum()
```

```
[1] 36740
```

```
toks_usdebates2020_reduced |>
    ntype() |>
    sum()
```

```
[1] 24965
```

The number of tokens and types decreases to 36,740 and 28,174, respectively, after removing punctuation and harmonizing all terms to lowercase.

1.2.4 Keywords-in-context

In contrast to a document-feature matrix (covered below), tokens objects still preserve the order of words. We can use tokens objects to identify the occurrence of keywords and their immediate context.

```
kw_america <- kwic(toks_usdebates2020,
                   pattern = c("america"),
                   window = 2)

# number of mentions
nrow(kw_america)
```

```
[1] 18
```

```
# print first 6 mentions of America and the context of ±2 words
head(kw_america, n = 6)
```

```
Keyword-in-context with 6 matches.
 [text1.284, 66]        towns in | America | , how
 [text1.290, 80]       states in | America | with a
 [text1.290, 96]       States of | America | , and
  [text1.335, 5] worst president | America | has ever
 [text1.489, 37]        whole of | America | . But
 [text1.500, 53]      applied in | America | ."
```

1.2.5 Text processing

The keywords-in-context analysis above reveals that all terms are still in upper case and that very frequent, uninformative words (so-called stopwords) and punctuation are still part of the text. In most applications, we remove very frequent features and transform all words to lowercase. The code below shows how to adjust the object accordingly.

```
toks_usdebates2020_processed <- data_corpus_debatesseg |>
    tokens(remove_punct = TRUE) |>
    tokens_remove(pattern = stopwords("en")) |>
    tokens_tolower()
```

Let's inspect if the changes have been implemented as we expect by calling `kwic()` on the new tokens object.

```
kw_america_processed <- kwic(toks_usdebates2020_processed,
                             pattern = c("america"),
                             window = 2)

# print first 6 mentions of America and the context of ±2 words
head(kw_america_processed, n = 6)
```

```
Keyword-in-context with 6 matches.
 [text1.284, 32]     class towns | america | well guy
 [text1.290, 29]     half states | america | significant increase
 [text1.290, 38]   united states | america | wants open
  [text1.335, 3] worst president | america | ever come
 [text1.489, 15]  equality whole | america | never accomplished
 [text1.500, 25] equally applied | america | believe separate
```

Pre	Keyword	Post	Pattern
class towns	america	well guy	america
half states	america	significant increase	america
united states	america	wants open	america
worst president	america	ever come	america
equality whole	america	never accomplished	america
equally applied	america	believe separate	america
defeat racism	america		america
increase homicides	america	summer particularly	america
less violence	america	today president	america
fired plant	america	one's going	america
fire plant	america	going move	america
united states	america	situation thousands	america
every company	america	blow away	america
united states	america		america
united states	america	anybody seeking	america
section race	america	want talk	america
institutional racism	america	always said	america
growing industry	america	electric excuse	america

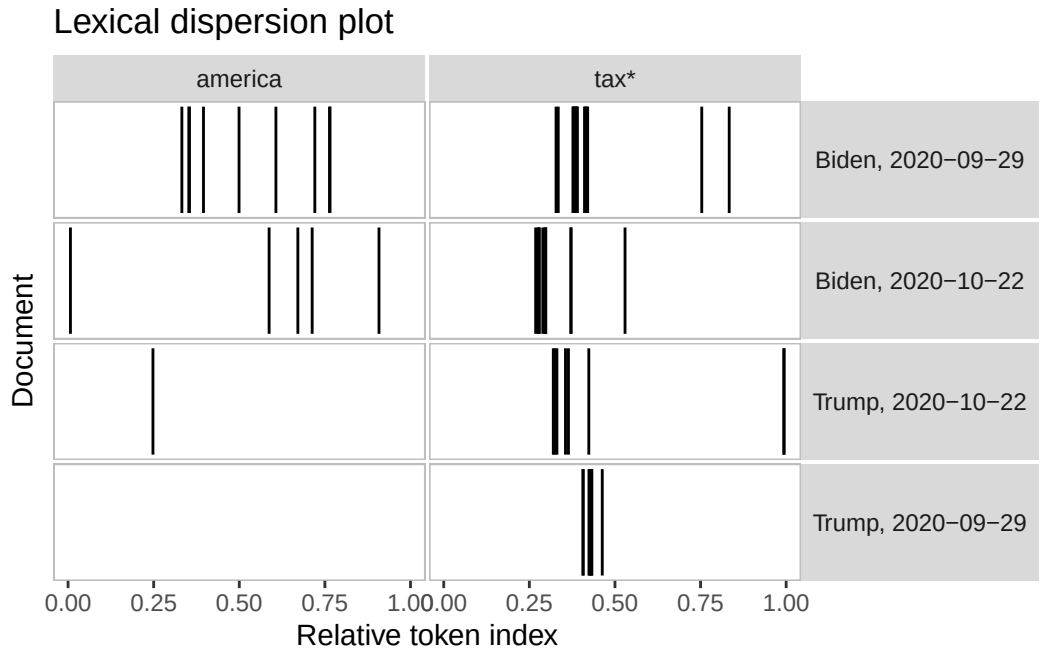
```
# test: print as table+
library(kableExtra)
kw_america_processed |> data.frame() |>
  dplyr::select(Pre = pre, Keyword = keyword, Post = post, Pattern = pattern) |>
  kbl(booktabs = T) %>%
  kable_styling(latex_options = c("striped", "scale_down"), html_font = "Source Sans Pro", full_wi
```

The processing of the tokens object worked as expected. Let's imagine we want to group the documents by debate and speaker, resulting in two documents for Trump, two for Biden, and one for each moderator. `tokens_group()` allows us to change the unit of analysis. After aggregating the documents, we can use the function `textplot_xray()` to observe the occurrences of specific keywords during the debates.

```
# new document-level variable with date and speaker
toks_usdebates2020$speaker_date <- paste(
    toks_usdebates2020$speaker,
    toks_usdebates2020$date,
    sep = ", ")

# reshape the tokens object to speaker-date level
# and keep only Trump and Biden
toks_usdebates2020_grouped <- toks_usdebates2020 |>
    tokens_subset(speaker %in% c("Trump", "Biden")) |>
    tokens_group(groups = speaker_date)

# check number of documents
ndoc(toks_usdebates2020_grouped)
```

```
[1] 4
```

```
# use absolute position of the token in the document
textplot_xray(
    kwic(toks_usdebates2020_grouped, pattern = "america"),
    kwic(toks_usdebates2020_grouped, pattern = "tax*")
)
```



The grouped document also allows us to check how often each candidate spoke during the debate.

```
ntoken(toks_usdebates2020_grouped)
```

```
Biden, 2020-09-29 Biden, 2020-10-22 Trump, 2020-09-29 Trump, 2020-10-22
             7996              8093              8973              9016
```

The number of tokens between Trump and Biden do not differ substantively in both debates. Trump's share of speech is only slightly higher than Biden's (7996 v 8093 tokens; 8973 v 9016 tokens).

1.2.6 Identifying multiword expressions

Many languages build on multiword expressions. For instance, "income" and "tax" as separate unigrams have a different meaning than the bigram "income tax". The package **quanteda.textstats** includes the function `textstat_collocation()` that automatically retrieves common multiword expressions.

```
tstat_coll <- data_corpus_debatesseg |>
    tokens(remove_punct = TRUE) |>
    tokens_remove(pattern = stopwords("en"), padding = TRUE) |>
    textstat_collocations(size = 2:3, min_count = 5)

# for illustration purposes select the first 20 collocations
head(tstat_coll, 20)
```

```
       collocation count count_nested length   lambda        z
1  president trump    71           44      2 6.928550 22.01175
2        make sure    30            4      2 7.439572 21.53052
3  president biden    52           52      2 6.450825 20.63073
4     mr president    34           21      2 5.389748 19.78036
5      health care    20           15      2 7.580147 19.32484
6       number one    18           16      2 5.595991 17.94433
7         right now    18           12      2 4.801444 16.82701
8       number two    15           11      2 5.472508 16.69323
9     half million    15           13      2 6.852610 16.55670
10          mr vice    15           15      2 5.323919 16.52522
11      four years    19           15      2 7.433288 16.49838
12 american people    22            4      2 4.969408 16.44720
13     two minutes    28           19      2 8.504176 16.09104
```

```
14       come back    12           7      2 5.537489 15.46566
15     three years    12           5      2 5.464293 15.20941
16      one number    12          12      2 5.122270 14.65987
17  climate change    11           8      2 8.963835 14.65614
18  million people    18          18      2 4.073645 14.42229
19  final question    14           7      2 7.008316 14.40121
20  vice president    99          81      2 9.070192 14.05817
```

We can use `tokens_compound()` to compound certain multiword expressions before creating a document-feature matrix which does not consider word order. For illustration purposes, we compound `climate change` ,`social securit*`, and `health insurance*`. By default, compounded tokens are concatenated by _.

```
toks_usdebates2020_comp <- toks_usdebates2020 |>
    tokens(remove_punct = TRUE) |>
    tokens_compound(pattern = phrase(c("climate change",
                                       "social securit*",
                                       "health insuranc*"))) |>
    tokens_remove(pattern = stopwords("en"))
```

1.2.7 Document-feature matrix

We have come a long way already. We downloaded debate transcripts, segmented the texts to utterances, added document-level variables, tokenized the corpus, inspected keywords, and compounded multiword expressions. Next, we transform our tokens object into a document-feature matrix (dfm). A dfm counts the occurrences of tokens in each document. We can create a document feature matrix, print the structure, and get the most frequent words.

```
dfmat_presdebates20 <- dfm(toks_usdebates2020_comp)

# most frequent features
topfeatures(dfmat_presdebates20, n = 10)
```

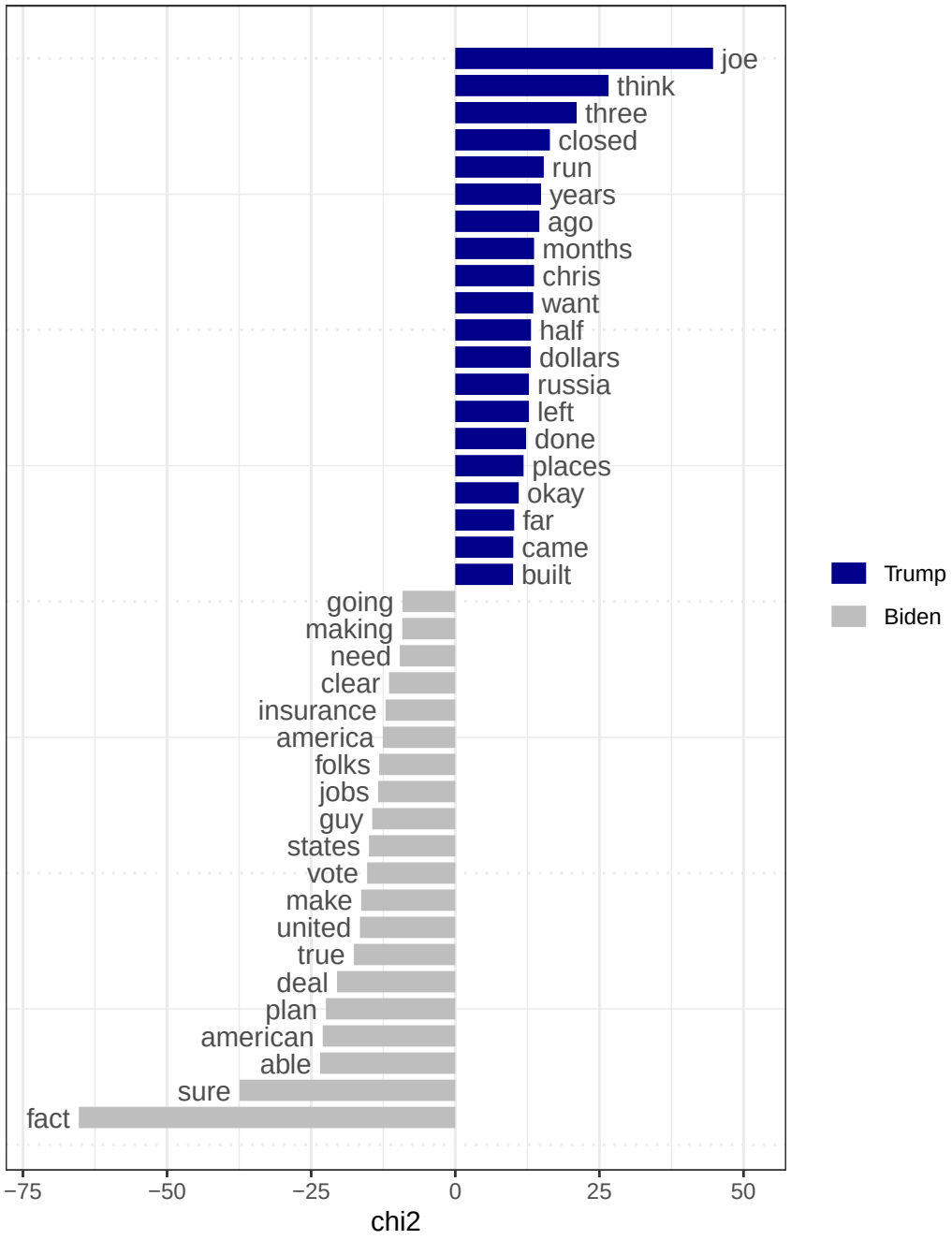
```
president     going    people      said      know      want       get       say
      288       278       259       164       139       129       119       114
     look      vice
      105       101
```

```
# most frequent features by speaker

topfeatures(dfmat_presdebates20, groups = speaker, n = 10)
```

```
$Biden
    going    people      fact      said       get      make      know president
      126       119        72        62        54        46        46        45
     sure       can
       44        43

$Trump
people  going   know   said   look   want    joe   done    say  think
   111     92     85     77     65     62     55     55     54     52

$Wallace
president       sir     going  question        mr      vice     trump        go
      110        56        44        39        39        36        33        30
      two     right
       28        25

$Welker
president      vice     trump     biden  question     right       let      move
      100        52        39        35        31        28        28        24
     talk      want
       21        18
```

Many methods build on document-feature matrices and the "bag-of-words" approach. In this section, we introduce `textstat_keyness()`, which identifies features that occur differentially across different categories – in our case, Trump's and Biden's utterances. The function `textplot_keyness()` provides a straightforward way of visualize the results of the keyness analysis (Figure **??**(fig:keynesspres).)

```
tstat_key <- dfmat_presdebates20 |>
    dfm_subset(speaker %in% c("Trump", "Biden")) |>
    dfm_group(groups = speaker) |>
    textstat_keyness(target = "Trump")

textplot_keyness(tstat_key)
```



1.2.8 Bringing it all together: Sentiment analysis

Before introducing the R Basics in the next chapter, we show how to conduct a dictionary-based sentiment analysis using the Lexicoder Sentiment Dictionary (Young and Soroka 2012). The dictionary, included in **quanteda** as

`data_dictionary_LSD2015` contains 1709 positive and 2858 negative terms (as well as their negations). We discuss advanced sentiment analyses with measures of uncertainty in Chapter ??.

```
toks_sent <- toks_usdebates2020 |>
    tokens_group(groups = speaker) |>
    tokens_lookup(dictionary = data_dictionary_LSD2015,
                  nested_scope = "dictionary")

# create a dfm with the count of matches,
# transform object into a data frame,
# and add document-level variables
dat_sent <- toks_sent |>
    dfm() |>
    convert(to = "data.frame") |>
    cbind(docvars(toks_sent))

# select Trump and Biden and aggregate sentiment
dat_sent$sentiment = with(
    dat_sent,
    log((positive + neg_negative + 0.5) /
            (negative + neg_positive + 0.5)))

dat_sent
```

```
   doc_id negative positive neg_positive neg_negative speaker   sentiment
1   Biden      357      465           33            8   Biden  0.19272394
2   Trump      430      509            7            2   Trump  0.15627088
3 Wallace      136      185            0            4 Wallace  0.32806441
4  Welker      106      102            0            0  Welker -0.03828219
```

1.3 Issues

Some issues raised from the example, where we might have done things differently or added additional analysis.

1.4 Further Reading

Some studies of debate transcripts? Or issues involved in analyzing interactive or "dialogical" documents.

1.5 Exercises

Add some here.

Part I

R

2 R Basics

2.1 Objectives

To provide a targeted introduction to R, for those needing an introduction or a review.

We will cover:

- core object types (atomic, vectors, lists, complex)
- detailed survey of characters and "strings"
- detailed coverage of list types
- detailed coverage of matrix types
- introduction to the data.frame
- output data in the form of a data.frame
- output data in the form of a plot (ggplot2 object)

The objective is to introduce how these core object types behave, and use them to store basic quantities required in text analysis.

2.2 Methods

The methods are primarily about how to use R, including:

We will build up the basic objects needed for understand the core structures needed in R to hold texts (character), meta-data (data.frame), tokens (lists), dictionaries (lists), and document-feature matrices (matrices).

2.3 Examples

Examples working through the construction of each text object container using the types above.

2.4 Issues

Could have used “tidy” approaches.

Sparsity.

Indexing.

Wouldn’t it be nice if we could nest some objects inside others?

2.5 Further Reading

Some resources on R.

2.6 Exercises

Add some here.

3 R Workflow

3.1 Objectives

How R works: - passing by value, compared to other languages - using “pipes” - object orientation: object classes and methods (functions)

Workflow issues: - clear code - reproducible workflow - R markdown and R notebooks

Using packages

3.2 Methods

Applicable methods for the objectives listed above.

3.3 Examples

Objects that work on character and lists.

Things that work for data.frames. Non-standard evaluation for named columns in a data.frame.

ggplot2

3.4 Issues

Additional issues.

3.5 Further Reading

Advanced reading on R. Packages.

3.6 Exercises

Add some here.

4 Working with Text in R

4.1 Objectives

A deeper dive into how text is represented in R. - character vectors vesus "strings" - factors - **base** package string handling - the **stringi** and **stringr** packages - regular expressions - encoding, briefly.

4.2 Methods

Applicable methods for the objectives listed above.

4.3 Examples

Lots of the above.

4.4 Issues

Additional issues.

4.5 Further Reading

String handling in R. More regular expressions. More on encoding.

4.6 Exercises

Add some here.

Part II

Acquiring Texts

5 Working with Files

5.1 Objectives

Really basic issues for working with files, such as: - Where files typically live on your computer; - Paths, the "current working directory" and how to change it; - Input formats and methods to convert them; - saving and naming files; - text editors; - cleaning texts; - detecting and converting text file encodings.

5.2 Methods

Applicable methods for the objectives listed above.

5.3 Examples

Examples of each of the above methods.

5.4 Issues

Additional issues.

5.5 Further Reading

Additional resources from libraries or the web.

5.6 Exercises

Add some here.

6 Using Text Import Tools

6.1 Objectives

- The **readtext** package
- Using API gateway packages: **twitteR** and other social media packages
- Dealing with JSON formats
- Newspaper formats, LexisNexis, etc.
- OCR tools

6.2 Methods

Applicable methods for the objectives listed above.

6.3 Examples

Examples of each of the above methods.

6.4 Issues

Additional issues.

6.5 Further Reading

Additional resources from libraries or the web.

6.6 Exercises

Add some here.

7 Obtaining Texts from the Internet

7.1 Objectives

- Web scraping
- Markup formats
- Challenges of removing tags
- APIs, and JSON

7.2 Methods

Applicable methods for the objectives listed above.

7.3 Examples

Examples of each of the above methods.

7.4 Issues

Additional issues.

7.5 Further Reading

Additional resources from libraries or the web.

7.6 Exercises

Add some here.

Part III

Managing Textual Data Using quanteda

8 Introducing the quanteda Package

8.1 Objectives

- Purpose
- Basic object types in **quanteda**
- Inter-related classes
- Workflow
- Key differences between **quanteda** and other packages
- Working with other packages

8.2 Methods

Applicable methods for the objectives listed above.

8.3 Examples

Examples of each of the above methods.

8.4 Issues

Additional issues.

8.5 Further Reading

Additional resources from libraries or the web.

8.6 Exercises

Add some here.

9 Creating and Managing Corpora

9.1 Objectives

Everything you always wanted to know about corpus creation and manipulation, but were afraid to ask. - corpus creation - reshaping corpora - segmenting corpora - subsetting corpora - extracting texts (using `texts()`) - managing metadata.

9.2 Methods

Applicable methods for the objectives listed above.

9.3 Examples

Examples of each of the above methods.

9.4 Issues

Additional issues.

9.5 Further Reading

Additional resources from libraries or the web.

9.6 Exercises

Add some here.

10 Creating and Managing tokens

10.1 Objectives

Creating and basic selection and manipulation of tokens.

- tokenizing and tokenization options
- selecting tokens
- removing tokens
- "stop words"
- stemming and lemmatization
- managing metadata in tokens objects

10.2 Methods

Applicable methods for the objectives listed above.

10.3 Examples

Examples of each of the above methods.

10.4 Issues

Additional issues.

10.5 Further Reading

Additional resources from libraries or the web.

10.6 Exercises

Add some here.

11 Advanced Token Manipulation

11.1 Objectives

Advanced operations with tokens.

- selecting within windows
- replacement
- splitting
- chunking
- compounding
- the `phrase()` function
- ngrams
- skipgrams
- brief introduction to lookup

11.2 Methods

Applicable methods for the objectives listed above.

11.3 Examples

Examples of each of the above methods.

11.4 Issues

Additional issues.

11.5 Further Reading

Additional resources from libraries or the web.

11.6 Exercises

Add some here.

12 Creating and Managing Dictionaries

12.1 Objectives

Creating and revising dictionaries. - key-value lists - more details on pattern types - the `phrase()` argument - importing foreign dictionaries - editing dictionaries - using dictionaries as `pattern` inputs to other functions - exporting dictionaries - using dictionaries: `tokens_lookup()`

12.2 Methods

Applicable methods for the objectives listed above.

12.3 Examples

Examples of each of the above methods.

12.4 Issues

Copyright.

Which format?

Weighted dictionaries.

12.5 Further Reading

On-line sources of dictionaries.

QI's **quanteda.dictionaries** package.

12.6 Exercises

Add some here.

13 Building Document-Feature Matrices

13.1 Objectives

Creating and revising dfm objects.

- the nature of a dfm object, and the importance of sparsity
- dfm creation - document selection (subsetting)
- document grouping (and via docvars)
- feature selection and transformation - weighting - trimming - smoothing - sorting - matching dfms for machine learning - fcm creation and manipulation - converting dfm objects for further use

13.2 Methods

Applicable methods for the objectives listed above.

13.3 Examples

Examples of each of the above methods.

13.4 Issues

Additional issues.

13.5 Further Reading

Additional resources from libraries or the web.

13.6 Exercises

Add some here.

Part IV

Exploring and Describing Texts

14 Describing Texts

14.1 Objectives

Describing the data in texts. - `n` functions - `summary()` - `docnames()` - `docvars()` - `meta()` - `featnames()` - `head()`, `tail()`, `print()`

14.2 Methods

Applicable methods for the objectives listed above.

14.3 Examples

Examples of each of the above methods.

14.4 Issues

Additional issues.

14.5 Further Reading

Additional resources from libraries or the web.

14.6 Exercises

Add some here.

15 Keywords-in-Context

15.1 Objectives

- What are “keywords”.

- The notion of a “concordance”
- Using patterns and different input types
- Multi-word patterns
- Using kwic objects for subsequent input - to `corpus()`, or to `textplot_xray()`
- Preview: Statistical detection of key words

15.2 Methods

Applicable methods for the objectives listed above.

15.3 Examples

Using `kwic()` to verify a dictionary usage.

15.4 Issues

Additional issues.

15.5 Further Reading

Additional resources from libraries or the web.

15.6 Exercises

Add some here.

16 Applying Dictionaries

16.1 Objectives

- The basic idea of counting token and feature matches as equivalence classes
- Matching phrases
- Weighting dictionary applications
- Multi-lingual applications
- Scaling dictionary results
- Confidence intervals and bootstrapping
- Dictionary validation using `kwic()`
- Dictionary construction using `textstat_keyness()`

16.2 Methods

Applicable methods for the objectives listed above.

16.3 Examples

Sentiment analysis.

16.4 Issues

Additional issues.

16.5 Further Reading

Additional resources from libraries or the web.

16.6 Exercises

Add some here.

17 Most Frequent Words

17.1 Objectives

- Summarizing features via a `dfm()`
- Most frequent features
- `dfm_group()` revisited
- Using dictionaries to count keys
- `topfeatures()`
- `textplot_wordcloud()`
- `textstat_frequency()`
- `textstat_keyness()`, `textplot_keyness()`

17.2 Methods

Applicable methods for the objectives listed above.

17.3 Examples

Using `kwic()` to verify a dictionary usage.

17.4 Issues

Additional issues.

17.5 Further Reading

Additional resources from libraries or the web.

17.6 Exercises

Add some here.

Part V

Statistics for Comparing Texts

18 Profiling Lexical Patterns and Usage

18.1 Objectives

- Sentence length
- Syllables
- Types versus tokens
- Usage in word lists
- Lexical diversity
- Readability
- Bootstrapping methods and uncertainty accounting

18.2 Methods

Applicable methods for the objectives listed above.

18.3 Examples

Sentiment analysis.

18.4 Issues

Non-English languages.

Sampling.

Text length and its effect on diversity. Zipf's law and Heap's law.

18.5 Further Reading

Additional resources from libraries or the web.

18.6 Exercises

Add some here.

19 Document Similarity and Distance

19.1 Objectives

- Distance and similarity measures
- Measuring similarity
- Measuring distance
- Clustering
- Multi-dimensional scaling
- Network analysis of document connections

19.2 Methods

Applicable methods for the objectives listed above.

19.3 Examples

Examples here.

19.4 Issues

Weighting and feature selection and its effects on similarity and distance.

Computational issues.

19.5 Further Reading

Additional resources from libraries or the web.

19.6 Exercises

Add some here.

20 Feature Similarity and Distance

20.1 Objectives

- Distance and similarity measures revisited
- Clustering
- Network analysis of feature connections
- Improving feature comparisons through detection and pre-processing of multi-word expressions

20.2 Methods

Applicable methods for the objectives listed above.

20.3 Examples

Examples here.

20.4 Issues

Computational issues.

20.5 Further Reading

Additional resources from libraries or the web.

20.6 Exercises

Add some here.

Part VI

Machine Learning for Texts

21 Supervised Document Scaling

21.1 Objectives

- Introduction to the notion of document scaling. Difference from classification, which we cover in Chapter @ref(ml-classifiers).

- Supervised versus unsupervised methods
- Wordscores
- Class affinity model

21.2 Methods

Applicable methods for the objectives listed above.

21.3 Examples

Examples here.

21.4 Issues

Training documents and how to select them.

Uncertainty accounting.

21.5 Further Reading

Additional resources from libraries or the web.

21.6 Exercises

Add some here.

22 Unsupervised Document Scaling

22.1 Objectives

- Notion of unsupervised scaling, based on distance
- LSA
- Correspondence analysis
- The "wordfish" model

22.2 Methods

Applicable methods for the objectives listed above.

22.3 Examples

Examples here.

22.4 Issues

The hazards of *ex post* interpretation of scaling.

Dimensionality.

22.5 Further Reading

Additional resources from libraries or the web.

22.6 Exercises

Add some here.

23 Methods for Text Classification

23.1 Objectives

- Class prediction versus scaling, and the notion of predicting classes
- Naive Bayes
- SVMs
- More advanced methods
- Feature selection for improving prediction
- Assessing performance

23.2 Methods

Applicable methods for the objectives listed above.

23.3 Examples

Examples here.

23.4 Issues

Why kNN is poor choice.

Which classifier?

ngrams or unigrams?

23.5 Further Reading

Additional resources from libraries or the web.

23.6 Exercises

Add some here.

24 Word embedding models

24.1 Objectives

- The `word2vec` method and its descendants
- Constructing a feature-cooccurrence matrix
- Fitting a word embedding model
- Using pre-trained embedding scores with documents
- Prediction based on embeddings
- Further methods

24.2 Methods

Applicable methods for the objectives listed above.

24.3 Examples

Examples here.

24.4 Issues

Issues here.

24.5 Further Reading

Additional resources from libraries or the web.

24.6 Exercises

Add some here.

25 Topic modelling

25.1 Objectives

- Latent Dirichlet Allocation
- Older methods (e.g. LSA)
- Advanced methods
- Using the **stm** package
- Pre-processing steps and their importance, including compounding MWEs

25.2 Methods

Applicable methods for the objectives listed above.

25.3 Examples

Examples here.

25.4 Issues

Using other topic model packages

25.5 Further Reading

Additional resources from libraries or the web.

25.6 Exercises

Add some here.

Part VII

Further Methods for Texts

26 Advanced NLP using R

26.1 Objectives

- What is "NLP" and how does it differ from what we have covered so far?
- Part of speech tagging
- Different schemes for part of speech tagging
- Differentiating homographs based on POS
- POS tagging in other languages
- Named Entity Recognition
- Noun phrase extraction
- Dependency parsing to extract syntactic relations

26.2 Methods

Applicable methods for the objectives listed above.

26.3 Examples

Examples of each of the above methods.

26.4 Issues

Alternative NLP tools. (Here we focus on **spacyr**.)

Other languages.

Training new models.

26.5 Further Reading

Additional resources from libraries or the web.

26.6 Exercises

Add some here.

27 Integrating "tidy" approaches

27.1 Objectives

- What is the "tidy" approach and how does it apply to text?
- The **tidytext** package and its advantages
- Switching from **quanteda** to **tidytext** (and back)
- Advantages of non-tidy text objects

27.2 Methods

Applicable methods for the objectives listed above.

27.3 Examples

Examples of each of the above methods.

27.4 Issues

Efficiency. Complexity.

27.5 Further Reading

Tidy resources. Tidytext book. Data Science Using R (Wickham and Grolemund).

27.6 Exercises

Add some here.

28 Text analysis in “Hard” Languages

28.1 Objectives

- Non-English characters, and encoding issues
- Segmentation in languages that do not use whitespace delimiters between words
- Right-to-left languages
- Emoji

28.2 Methods

Applicable methods for the objectives listed above.

28.3 Examples

Chinese, Japanese, Korean.

Hindi, Georgian.

Arabic and other RTL languages.

28.4 Issues

Stemming, syllables, character counts may be off.

Font issues.

28.5 Further Reading

Further reading here.

28.6 Exercises

Add some here.

References

Benoit, Kenneth. 2020. "Text as Data: An Overview." In *Handbook of Research Methods in Political Science and International Relations*, edited by Luigi Curini and Robert Franzese, 461–97. Thousand Oaks: Sage.

Benoit, Kenneth, Kohei Watanabe, Haiyan Wang, Paul Nulty, Adam Obeng, Stefan Müller, and Akitaka Matsuo. 2018. "Quanteda: An R Package for the Quantitative Analysis of Textual Data." *Journal of Open Source Software* 3 (30): 774. https://doi.org/10.21105/joss.00774.

Feinerer, Ingo, Kurt Hornik, and David Meyer. 2008. "Text Mining Infrastructure in R." *Journal of Statistical Software* 25 (5): 1–54. https://www.jstatsoft.org/v25/i05/.

Wickham, Hadley. 2021. *Rvest: Easily Harvest (Scrape) Web Pages*. https://CRAN.R-project.org/package=rvest.

Young, Lori, and Stuart N. Soroka. 2012. "Affective News: The Automated Coding of Sentiment in Political Texts." *Political Communication* 29 (2): 205–31.

A Installing the Required Tools

A.1 Objectives

- Installing R
- Installing RStudio
- Installing **quanteda**
- Installing **spacy**
- Installing companion package(s)
- Keeping up to date
- Troubleshooting problems

A.2 Installing R

R is a free software environment for statistical computing that runs on numerous platforms, including Windows, macOS, Linux, and Solaris. You can find details at https://www.r-project.org/, and link there to a set of mirror websites for downloading the latest version.

For this book, we have used R 3.5.3, which we recommend you use also. There are seldom reasons to use older versions of R, and the R Core Team and the maintainers of the largest repository of R packages, CRAN (for Comprehensive R Archive Network) put an enormous amount of attention and energy into assuring that extension packages work with stably and with one another.

You verify which version of R you are using by either viewing the messages on startup, e.g.

```
R version 3.5.3 (2019-03-11) -- "Great Truth"
Copyright (C) 2019 The R Foundation for Statistical Computing
Platform: x86_64-apple-darwin15.6.0 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.
```

```
  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.
>
```

or by calling

```
R.Version()$version.string
```

```
[1] "R version 4.2.1 (2022-06-23)"
```

A.3 Installing recommended tools and packages

A.3.1 RStudio

There are numerous ways of running R, including from the "command line" ("Terminal" on Mac or Linux, or "Command Prompt" on Windows) or from the R Gui console that comes with most installations of R. But our very strong recommendation is to us the outstanding RStudio Dekstop. "IDE" stands for *integrated development environment*, and provides a combination of file manager, package manager, help viewer, graphics viewer, environment browser, editor, and much more. Before this can be used, however, you must have installed R.

RStudio can be installed from https://www.rstudio.com/products/rstudio/download/.

A.3.2 Additional packages

The main package you will need for the examples in this book is **quanteda**, which provides a framework for the quantitative analysis of textual data. When you install this package, by default it will also install any required packages that **quanteda** depends on to function (such as **stringi** that it uses for many essential string handling operations). Because "dependencies" are installed into your local library, these additional packages are also available for you to use independently. You only need to install them once (although they might need updating, see below).

We also suggest you install:

- **readtext** - for reading in documents that contain text, and converting them automatically to plain text;
- **spacyr** - an R wrapper to the Python package spaCy for natural language processing, including part-of-speech tagging and entity extraction. While we have tried hard to make this automatic, installing and configuring **spacyr** actually involves some major work under the hood, such as installing a version of Python in a self-contained virtual environment and then installing spaCy and one of its language models.

A.3.3 Keeping packages up-to-date

R has an easy method for ensuring you have the latest versions of installed packages:

```
update.packages()
```

A.4 Additional Issues

A.4.1 Installing development versions of packages

The R packages that you can install using the methods described above are the pre-compiled binary versions that are distributed on CRAN. (Linux installations are the exception, as these are always compiled upon installation.) Sometimes, package developers will publish "development" versions of their packages that have yet to published on CRAN, for instance on the popular GitHub platform hosting the world's largest collection of open-source software.

The **quanteda** package, for instance, is hosted on GitHub at https://github.com/quanteda/quanteda, where its development version tends to be slightly ahead of the CRAN version. If you are feeling adventurous, or need a new version in which a specific issue or bug has been fixed, you can install it from the GitHub source using:

```
devtools::install_github("quanteda/quanteda")
```

Because installing from GitHub is the same as installing from the source code, this also involves compiling the C++ and Fortran source code that makes parts of **quanteda** so fast. For this source installation to work, you will need to have installed the appropriate compilers.

If you are using a Windows platform, this means you will need also to install the Rtools software available from CRAN.

If you are using macOS, you should install the macOS tools, namely the Clang 6.x compiler and the GNU Fortran compiler (as **quanteda** requires gfortran to build).[1] Linux always compiles packages containing C++ code upon installation, so if you are using that platform then you are unlikely to need to install any additional components in order to install a development version.

A.4.2 Troubleshooting

Most problems come from not having the latest versions of packages installed, so make sure you have updated them using the instructions above.

Other problems include: - Lack of permissions to install packages. This might affect Windows users of work laptops, whose workplace prevents user modification of any software. - Lack of internet access, or access being restricted by a firewall or proxy server.

A.5 Further Reading

Hadley Wickham's excellent book R Packages is well worth reading.

- Wickham, Hadley. (2015). *R packages: organize, test, document, and share your code.* O'Reilly Media, Inc..

[1] If you are still getting errors related to gfortran, follow the fixes here.

B Everything You Never Wanted to Know about Encoding

(and were afraid to ask)

B.1 Objectives

- The concept of encoding
- Classic 8-bit encodings
- How to detect encodings
- Converting files
- Converting text once loaded
- Encoding "bit" versus actual text encoding
- When you are likely to encounter problems (with what sorts of files)
- Unicode encodings (UTF-8, UTF-16)
- How to avoid encoding headaches forever

B.2 Methods

Applicable methods for the objectives listed above.

B.3 Examples

Examples here.

B.4 Issues

UTF-16 in some operating systems (Windows).

Editors that are "encoding smart".

B.5 Further Reading

Further reading here.

B.6 Exercises

Add some here.

C A Survival Guide to Regular Expressions

C.1 Objectives

- Basic of regular expressions
- Fixed matching and "globs" and relationship to regexes
- Base regular expressions versus **stringi**
- character classes
- Unicode character categories
- Lookaheads

C.2 Methods

Applicable methods for the objectives listed above.

C.3 Examples

Examples here.

C.4 Issues

Older variants: PCRE, versus GNU, versus **stringi** implementations.

Shortcut names versus Unicode categories.

C.5 Further Reading

Further reading here.

C.6 Exercises

Add some here.