university of groningen

Quantitative Text Analysis – Essex Summer School

Principles of QTA. Cleaning data. Preprocessing data

dr. Martijn Schoonvelde

University of Groningen

Today's class

- Principles of automated text analysis (Grimmer, Roberts, and Stewart, 2022)
- Clean text in R: string operations and regular expressions (using the **stringr** library)
- Preprocessing data: going from text to numbers (using the **quanteda** library)

Developing a corpus

Principles of selection (Grimmer, Roberts and Stewart, 2022)

1. The usefulness of a corpus depends on the question the researcher wants to answer and the population they want to study
   - The preponderance of textual data doesn't mean all text is useful. E.g., *X* data may be less useful for studying public opinion.
2. There is no values-free construction of a corpus. Selecting which documents to include has ethical ramifications
   - Just because it is found data (Salganik, 2018) doesn't mean you can just run with it

Developing a Corpus: Be Aware of Bias

When building a text corpus, be mindful of four common types of bias (Grimmer, Roberts & Stewart, 2022):

1. Resource Bias
   - Texts tend to overrepresent groups with more resources to create, record, and preserve documents.
   - Example: Government agencies vs. grassroots organizations.
2. Incentive Bias
   - People are strategic about what they post or preserve.
   - Example: Skydive photos on Instagram, but not laundry day; performative "hot takes" on *X*.
3. Medium Bias
   - The platform or format shapes what people say and how.
   - Example: More civil political discussions on Twitter after the 280-character limit was introduced (*Jaidka et al., 2019*); see also algorithmic drift (*Salganik, 2018*).
4. Retrieval Bias
   - Your search and sampling methods may skew the corpus.
   - Example: Searching for articles about the economy using only the keyword "economy" may miss relevant content.

Pre-existing Text Corpora

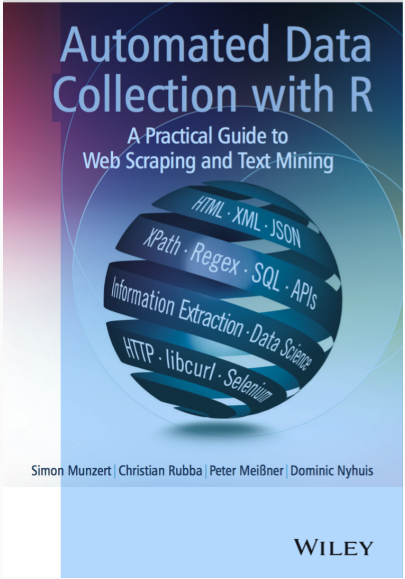
Before building your own corpus, consider using high-quality existing datasets:

- **News and media:**
  - Nexis Uni, Factiva – rich collections of newspaper and media content.
- **Political speeches:**
  - EUSpeech (Schumacher et al.), Parlspeech (Rauh et al.), ParlEE (Silvester et al.)
  - UN General Debate Corpus (Jankin et al.)
  - DICEU – Decision-making in the EU (Wratil & Hobolt)
- **Institutional documents:**
  - ECB Speeches: `https://centralbanktalk.eu/data-speech`
  - Party Manifestos (Manifesto Project): `https://manifesto-project.wzb.eu/`
- **Replication data:**
  - Many published papers share corpora in their replication folders (e.g., Harvard Dataverse, OSF).
- **Meta-repository (dataset of datasets)**:
  - `https://github.com/erikgahner/PolData`

Getting Text Data from the Web

- **APIs (Application Programming Interfaces):**
  - APIs let you access structured data from websites and online databases.
  - R packages: `WikidataR`, `openai`, `censusr`, and more.
- **Web scraping:**
  - If no API is available, you can extract content directly from websites.
  - Use the `rvest` package to parse HTML content.
- **Common file formats:**
  - API and scraped data often come in **JSON**, **HTML**, or **XML** formats.
  - You'll likely need to clean and reformat the data – this is called **data wrangling**.
  - Helpful packages: `rjson`, `jsonlite`, `jsontools`.

Getting data from the web



Representations of text

There are many different ways to represent text: **bag of words, word embeddings, sentence embeddings, document embeddings, dependency trees**, to name just a few

- These representations vary in their complexity and in the information they contain.

There is no one right way to represent text for all research questions (Grimmer, Roberts, Stewart, 2022) – it really depends on the question

- What is the quantity of interest that you are trying to measure? How will it manifest itself in the text?

Principles of Automated Text Analysis (Grimmer, Roberts & Stewart 2022)

These principles represents an agnostic approach to text analysis (as opposed to a structural approach)

1. Social science theories and domain knowledge are essential for research design
   - E.g., when examining parliamentary speech, we need to know how a bill becomes a law in a country.
2. QTA does not replace humans – it augments them
   - For example, QTA may help us categorise our documents or find key documents
3. We have to separate the discovery phase from the testing phase
4. The best method depends on the task
5. Validations are essential and depend on the theory and the task

Preparing texts for analysis

Once you've selected your texts, they need to be prepared for analysis. This preparation typically involves three key steps:

1. **Import the texts:** Load your documents into R.
2. **Clean the texts:** Remove unwanted content such as metadata, HTML tags, punctuation, and numbers. This helps focus the analysis on meaningful words.
3. **Preprocess the texts:** Standardize the text by applying steps such as lowercasing, stemming or lemmatization, and removing stopwords. The goal is to simplify the text while preserving meaning.

Importing text into R

- **Common file formats:**
  - `.txt`, `.csv`, `.tsv`: Use base R functions like `read.csv()` or `read.table()`, or Tidyverse alternatives like `read_csv()` and `read_tsv()`.
  - `.json`, `.xml`, `.html`, `.pdf`: Use specialized tools like the `readtext` package to handle more complex formats.
- **Choosing the right package:**
  - R offers many ways to do the same task – there?s no single "correct" method.
  - Check package documentation, browse examples on Stack Overflow, or Google "[format] R import" for up-to-date advice.

Importing Text into R

Challenges with character encoding:

- When importing text – especially in non-English languages – you may encounter issues with character encoding.
- Text is stored using encoding schemes like UTF-8, UTF-16, or ASCII. If R guesses the wrong encoding, you may see strange symbols or question marks.
- There is no one-size-fits-all solution; the correct encoding often depends on the original source and format.

Example: Specify encoding manually

```
text <- readLines("data/french_speech.txt", encoding = "UTF-8")
```

This tells R to read the file assuming it uses UTF-8 encoding (common for web text and modern documents).

More on encoding: `http://kunststube.net/encoding/`

Text snippet

```
<p>Ladies and gentlemen,</p><p>It is an honour to be here today to introduce the
theme of 'recession and recovery'. If you will permit,   I would like to suggest that
this afternoon we focus more on recovery than on recession. I think we know enough
about the recession side   of the story.</p><p>It started with the fall of Lehman
Brothers on 15 September 2008.. I happened to be here, at the Blouin Creative
Leadership Summit, only ten days later. Everyone was talking about the collapse of
Lehman. They were shocked and alarmed.   But even then we could hardly imagine that
its impact would be so dramatic, so historic.</p><p>As we now know, this event
triggered a global financial and economic crisis. Governments were forced to give
cash injections running into billions to prevent an economic and financial meltdown.
When credit dried up and demand fell, businesses struggled  to keep their heads above
water, and many went under. Ordinary people's jobs, homes and pensions were at
risk.</p><p>
```

Cleaning Text with String Operations

- You can clean and manipulate text using **base R** functions like `gsub()`, `tolower()`, and `nchar()` – no extra packages needed.
- For more readable and powerful string handling, use packages like:
  - **stringr** (part of the Tidyverse): Consistent syntax and naming.
  - **stringi**: Fast and supports complex operations.

- **Regular Expressions (regex)** allow you to match patterns in text:
  - Powerful for removing or extracting specific content (e.g., email addresses, punctuation, tags).
  - Often requires trial and error, especially for complex patterns.
  - Helpful guide: **stringr** cheat sheet – `https://github.com/rstudio/cheatsheets/blob/master/strings.pdf`

The `stringr` package

Key features of `stringr`:

- All functions begin with `str_` for consistency.
- The first argument is always a character vector.
- Most functions support **regular expressions** for flexible pattern matching.

Examples:

```
corpus <- c("a", "ab", "abba", "bbbb")

str_length(corpus)          # Number of characters
# [1] 1 2 4 4

str_count(corpus, "a")      # Count of "a" in each string
# [1] 1 1 2 0
```

stringr example

```
corpus <- c(" This Is text_one <font size = '6'> ",
            "And here is teXt Number 2!?",
            "And text %%$ number 3")

#remove html tag
> corpus <- str_remove(corpus, "<.*>")
> print(corpus)
[1] " This Is text_one  "
[2]"And here is teXt Number 2!?"
[3] "And text %%$ number 3"
```

stringr example

Transform to lower case

```
corpus <- str_to_lower(corpus)
print(corpus)
```

```
[1] " this is text_one  "
[2] "and here is text number 2!?"
[3] "and text %%$ number 3"
```

stringr example

Remove everything but letters or numbers

```
corpus <- str_replace_all(corpus, "[^[:alnum:]]", " ")
print(corpus)
```

```
[1] " this is text one  "
[2] "and here is text number 2    "
[3] "and text number 3  "
```

Source: RStudio Stringr cheat sheet

stringr example

Remove multiple white spaces, as well as surrounding white spaces

```
corpus <- str_squish(corpus)
print(corpus)
```

```
[1] "this is text one"
[2] "and here is text number 2"
[3] "and text number 3"
```

Biden inaugural address

"So now, on this hallowed ground where just days ago violence sought to shake this Capitol's very foundation, we come together as one nation, under God, indivisible, to carry out the peaceful transfer of power as we have for more than two centuries." (Biden, 2021)

Bag of Words Representation

- The **Bag of Words (BoW)** model is a simple and widely used method for representing text data numerically.
  - Also known as a Document-Term Matrix (DTM) or Document-Feature Matrix (DFM).
- **Structure:**
  - Each row = one document.
  - Each column = one word (or feature).
  - Each cell = how often that word appears in the document (frequency).
- **Common variations:**
  - **N-grams:** Include word sequences like bigrams (2-grams) or trigrams (3-grams) instead of just single words.
  - **Weighting:** Use TF-IDF to reduce the influence of common but uninformative words.
  - **Binary matrix:** Just track presence or absence of words (1 or 0) instead of frequencies.

Biden inaugural address

"So now, on this hallowed ground where just days ago violence sought to shake this Capitol's very foundation, we come together as one nation, under God, indivisible, to carry out the peaceful transfer of power as we have for more than two centuries." (Biden, 2021)

	features									
docs	so	now	,	on	this	hallowed	ground	where	just	days
2021-Biden.12	1	1	5	1	2	1	1	1	1	1

Implications of the Bag of Words Model

- **Advantages:**
  - Easy to implement and interpret.
  - Captures useful word frequency patterns for many tasks (e.g., topic modeling, supervised classification).
- **Limitations:**
  - **Ignores word order:** The phrase "not good" is treated the same as the separate words $\{$*not*, *good*$\}$, which may distort meaning.
  - **Ambiguity:** Can't distinguish between meanings of the same word (e.g., *bank* as a financial institution vs. riverbank).
  - **No semantic context:** Words are treated independently, unlike modern models like word embeddings or transformer-based approaches (e.g., BERT).
  - These issues can be partially addressed using **n-grams** or phrase detection.

Bag of Words Model Pipeline

1. **Choose the unit of analysis**
   - Document, sentence, paragraph, etc.
2. **Reduce complexity:**
   - Convert to lowercase
   - Remove punctuation and numbers
   - Remove stopwords (e.g., "the", "and")
   - Lemmatize or stem words
     - Treat similar forms (e.g., *run*, *running*) as the same token type
   - Filter by word frequency (e.g., remove very rare or very common terms)
3. **Tokenize**
   - Break text into individual words or n-grams
4. **Build the Document-Feature Matrix (DFM)**
   - Rows = documents, columns = features (tokens), cells = frequencies

Tokenization

Tokenization is the process of dividing text into individual units – usually words or phrases.

- Each unit is called a token.
- Tokens that are the same (e.g., repeated words) belong to the same type.
- The full set of unique types in a corpus is called the vocabulary.

Example:

```
Text: "Data is data."

Tokens:      ["Data", "is", "data"]
Types:       {"Data", "is", "data"}  (3 tokens, 2 types if case-insensitive)
Vocabulary:  {"data", "is"}          (after lowercasing)
```

Tokenization

```
biden_sentence <- corpus(biden_sentence)
biden_tokens <- tokens(biden_sentence)
as.character(biden_tokens)
```

```
 [1] "So"  "now"  ","   "on"   "this" "hallowed"  "ground"
 [8] "where"  "just"  "days"  "ago"  "violence" "sought"  "to"
[15] "shake"  "this" "Capitol's"  "very"  "foundation"  "," "we"
[22] "come"  "together" "as" "one"  "nation"  ","  "under"
[29] "God"  ","  "indivisible" ","  "to" "carry" "out"
[36] "the"  "peaceful"  "transfer"   "of" "power"  "as"  "we"
[43] "have"  "for"  "more" "than" "two"  "centuries"   "."
```

NB everything that is not a white space is tokenized

Tokenization

The simplest way to tokenize is to divide into unigrams, but sometimes we lose important information (e.g., European Union, United States, International Monetary Fund). This is where multiword expressions come in.

```
> biden_tokens <- tokens_compound(biden_tokens, phrase(c("one nation", "under
God")))
> as.character(biden_tokens)

 [1] "So"  "now"  ","  "on"  "this"  "hallowed" "ground"  "where"  "just" "days"
[11] "ago" "violence" "sought"  "to"  "shake"  "this"  "Capitol's" "very"
[19] "foundation"  ","  "we" "come" "together" "as"  "one_nation"  ","
[27] "under_God"  "," "indivisible" "," "to" "carry" "out" "the" "peaceful"
[36] "transfer" "of"  "power"  "as" "we"   "have" "for"  "more"  "than"  "two"
[46] "centuries"   "."
```

Remove punctuation

```
> biden_tokens <- tokens(biden_sentence,
+                        remove_punct = TRUE)
> as.character(biden_tokens)

 [1] "So"  "now"  "on"  "this"  "hallowed" "ground"  "where"
 [8] "just"  "days"  "ago"   "violence"  "sought"  "to" "shake"
[15] "this"  "Capitol's"  "very"  "foundation"  "we" "come"  "together"
[22] "as"  "one"  "nation" "under"  "God"  "indivisible" "to"
[29] "carry"  "out"  "the"  "peaceful"  "transfer" "of"  "power"
[36] "as"  "we"  "have"  "for"  "more" "than" "two"
[43] "centuries"
```

NB everything that is not a white space is tokenized

Remove stopwords

```
> biden_tokens <- tokens_select(biden_tokens,
+                               pattern = stopwords("en"),
+                               selection = "remove")
> as.character(biden_tokens)

 [1] "now"  "hallowed"  "ground"  "just"  "days"  "ago"  "violence"
 [8] "sought"  "shake"  "Capitol's"  "foundation"  "come"  "together" "one"
[15] "nation" "God"  "indivisible" "carry"  "peaceful"  "transfer"  "power"
[22] "two"  "centuries"
```

Stemming

- Stemming: algorithmic conversion of inflected forms of words into their root forms
- Fast but not perfect:
  - Unrelated words may be grouped together; related words may not be grouped together
  - Stems may not be words themselves – problematic if further analysis is based on dictionaries

```
> biden_tokens_stemmed <- tokens_wordstem(biden_tokens,
language = quanteda_options("language_stemmer"))
> as.character(biden_tokens_stemmed)

 [1] "now"  "hallow"  "ground"  "just"  "day" "ago" "violenc" "sought" "shake"
[10] "Capitol" "foundat" "come"  "togeth" "one" "nation" "God" "indivis" "carri"
[19] "peac"  "transfer" "power"  "two"  "centuri"
```

Lemmatization

- Use of a dictionary to replace words with their root form
- Results are always words; neither does it group together unrelated words, nor does it miss to group together related words

```
words <- c("hallowed", "violence", "foundation")
lemma <- rep("XX", length(words))
biden_tokens_lemmatized <- tokens_replace(biden_tokens, words,
                                          lemma,
                                          valuetype = "fixed")
```

```
 [1] "now"   "XX"    "ground"  "just"    "days"   "ago"   "XX"
 [8] "sought"  "shake"  "Capitol's" "XX"   "come" "together" "one"
[15] "nation" "God"  "indivisible" "carry" "peaceful" "transfer"  "power"
[22] "two" "centuries"
```

Preprocessing data in languages other than English

Global linguistic diversity is enormous...

- Some 6,000–7,000 languages are spoken globally today
- Among other dimensions they vary in their script (e.g, Latin, Cyrillic, and Greek), their morphology (the system through which words get formed) and their syntax (how words are brought together to make sentence)

...but diversity in computational tools and methods much less so

- Keep in mind that preprocessing steps are language-specific!

dfm in quanteda

When we are happy with our features, we can create the document feature matrix using the dfm function

```
> dfm(biden_tokens)
Document-feature matrix of: 1 document, 23 features (0.00% sparse) and 4 docvars.
               features
docs             now hallowed ground just days ago violence sought shake capitol’s
  2021-Biden.12    1        1      1    1    1   1        1      1     1
   1
```

dfm in quanteda

```
> dfmat_inaugural <- data_corpus_inaugural %>%
+     tokens(remove_punct = TRUE) %>%
+     tokens_remove(stopwords("en")) %>%
+     dfm()
dfmat_inagural[,1:5]
Document-feature matrix of: 59 documents, 5 features (68.47% sparse) and 4 docvars
                 features
docs              fellow-citizens senate house representatives among
  1789-Washington               1      1     2               2     1
  1793-Washington               0      0     0               0     0
  1797-Adams                    3      1     0               2     4
  1801-Jefferson                2      0     0               0     1
  1805-Jefferson                0      0     0               0     7
  1809-Madison                  1      0     0               0     0
[ reached max_ndoc ... 53 more documents ]
```

Filtering and Weighting

"the greatest signal from words are those in the goldilocks region – neither too rare nor too frequent" (Grimmer, Roberts and Stewart, p. 75)

- **Feature Selection:** Not all terms contribute equally to an analysis. It's often useful to exclude:
  - Very common words that appear in many documents.
  - Very rare words that appear in few documents.

- **Document Frequency Filtering:** Apply thresholds to filter terms based on how frequently they appear across documents, either as a count or proportion.
- **Weighting Methods:** Assign weights to terms to enhance relevance, using techniques like tf-idf (term frequency-inverse document frequency).

Understanding TF-IDF Weighting

- **Term Frequency (TF):** Measures how frequently a term occurs in a document. The more often a term appears in a document, the higher its importance within that document.
- **Inverse Document Frequency (IDF):** Measures the importance of the term across a set of documents. The more documents the term appears in, the less unique it is, thus lowering its IDF score.
- **TF-IDF:** Combines TF and IDF, resulting in a weight that increases with the number of times a term appears in a document but decreases with the frequency of the term across documents.
- **Intuition:** TF-IDF penalizes common terms while giving higher weight to terms that are unique to a particular document. This helps in distinguishing documents and understanding their specific context.