Session 2:

Processing text & text as data

Agenda

1. Introduction to automated text analysis
2. Getting Data into R/Python
3. Cleaning/Normalizing Text
4. Choosing a Text-as-Data Representation
5. Outro

How can I access materials?

- Materials include
  - Reading list
  - Slides
  - R/Python Code (via Google Colab Notebook)

- Access via:
  - Summer school website
  - GitHub

Preparing the hands-on part in R

https://github.com/valeriehase/Salamanca-CSS-SummerSchool

Folder: "Processing text and text as data"

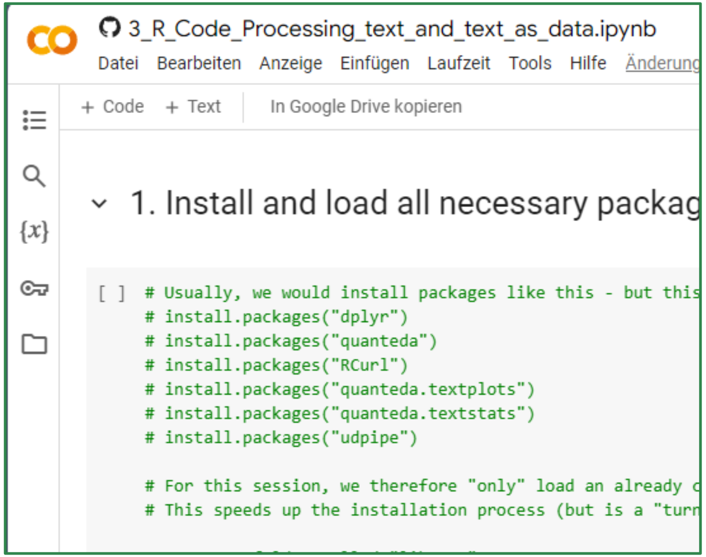
1_Slides_Processing text and text as data: Contains slides for respective session

2_Reading list_Processing text and text as data: Contains reading list for respective session. Please make sure to read the required text before the respective session.

3_R Code_Processing text and text as data: Contains R code for respective session via Colab Notebook. Open in Colab

4_Python Code_Processing text and text as data: Contains Python code for respective session via Colab Notebook. Open in Colab

data_tvseries : Contains CSV-dataset on best rated TV series. Provided under MIT license via Kaggle.



Preparing the hands-on part in R

R

```r
# Usually, we would install packages like this - but this takes forever on Colab notebooks (at least 15 min.)
# install.packages("dplyr")
# install.packages("quanteda")
# install.packages("RCurl")
# install.packages("quanteda.textplots")
# install.packages("quanteda.textstats")
# install.packages("udpipe")

# For this session, we therefore "only" load an already compiled, zipped file with all R packages
# This speeds up the installation process (but is a "turnaround")

# create a folder called "library"
system("mkdir library")

# download the R environment file containing complied packages
R_environment_file <- "https://drive.usercontent.google.com/download?id=1vmeZC68FTNNyanEl3c6DRWOvjumE1RMu&expo
download.file(R_environment_file, destfile="./library.tar.gz")

# unzip the compressed R library file: 'library.tar.gz' into the R library folder
untar("library.tar.gz", "library")

# change the R library directory into './library'
.libPaths("library")
```

How we would usually install packages

Turnaround for today

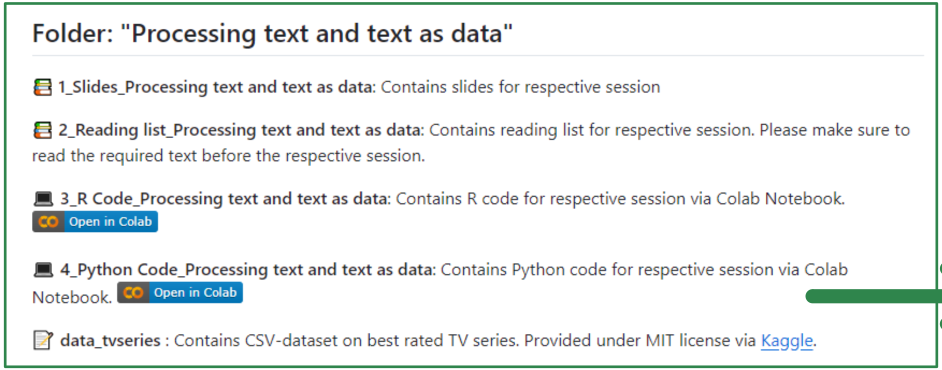
Preparing the hands-on part in R

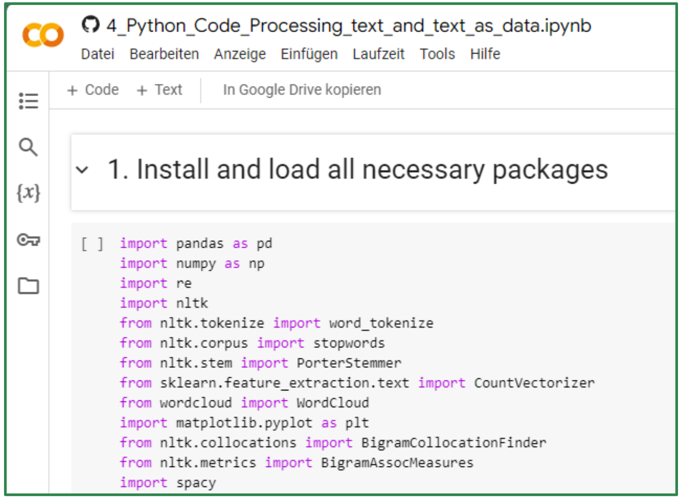
R

```r
# We activate relevant packages
library("dplyr")
library("quanteda")
library("RCurl")
library("quanteda.textplots")
library("quanteda.textstats")
library("udpipe")
```

Preparing the hands-on part in Python

https://github.com/valeriehase/Salamanca-CSS-SummerSchool





Preparing the hands-on in Python

Python

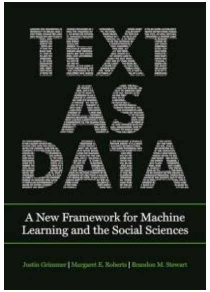
```python
import pandas as pd
import numpy as np
import re
import nltk
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from sklearn.feature_extraction.text import CountVectorizer
from wordcloud import WordCloud
import matplotlib.pyplot as plt
from nltk.collocations import BigramCollocationFinder
from nltk.metrics import BigramAssocMeasures
import spacy

# Ensure you have the necessary NLTK resources
nltk.download('punkt')
nltk.download('stopwords')
nlp = spacy.load("en_core_web_sm")
```

1. Introduction to Automated Text Analysis

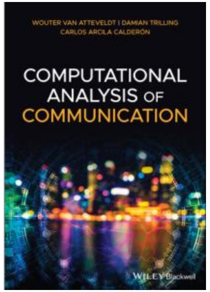
Automated text analysis: Definition

Automated text analysis describe the **automated (e.g., via programming scrips) analysis** of content (text, images). However, humans often act in a **supervisory way** through manual preparation, inspection, or validation of data. (Hase, 2023)

Benoit, 2020



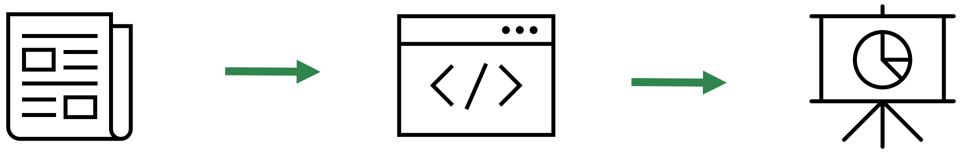
Grimmer et al., 2022



van Atteveldt et al., 2022

Automated text analysis: Definition

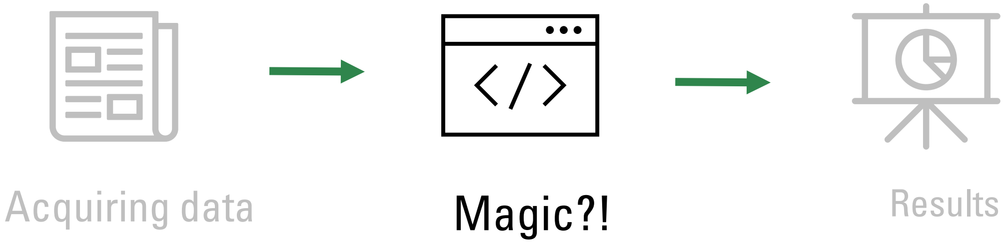
Automated text analysis describe the **automated (e.g., via programming scrips) analysis** of content (text, images). However, humans often act in a **supervisory way** through manual preparation, inspection, or validation of data. (Hase, 2023)



Acquiring data | Magic?! | Results

Automated text analysis: Definition

Automated text analysis describe the **automated (e.g., via programming scrips) analysis** of content (text, images). However, humans often act in a **supervisory way** through manual preparation, inspection, or validation of data. (Hase, 2023)



Unboxing „magic“: Typical steps

1. Preprocessing

«In most cities in Spain, the weather is sunny today»

Read in, clean, normalize & choose representation

«in»	(2x)
«most»	(1x)
«cities»	(1x)
…	…

- We read data (text, images) into R/Python.
- We clean/normalize text.
- We choose an appropriate text-as-data representation.

Unboxing „magic“: Typical steps

1. Preprocessing

«In most cities in Spain, the weather is sunny today»

Read in, clean, normalize & choose representation

«in»	(2x)
«most»	(1x)
«cities»	(1x)
…	…

2. Analysis

Question: How does the article describe the weather in Spain?

Answer: The text contains more features associated with positive sentiment («sunny»).

We choose a method to analyze *manifest* features (e.g., count of features associated with positive sentiment) to infer *latent constructs* (such as: “opinion of weather in Spain”).

Unboxing „magic“: Typical steps

1. Preprocessing

«In most cities in Spain, the weather is sunny today»

Read in, clean, normalize & choose representation

«in»	(2x)
«most»	(1x)
«cities»	(1x)
…	…

2. Analysis

Question: How does the article describe the weather in Spain?

Answer: The text contains more features associated with positive sentiment («sunny»).

3. Test against Quality Criteria

We evaluate the automated analysis in light of quality criteria (e.g., reproducibility, replicability, validity).

Unboxing „magic“: Typical steps

1. Preprocessing

2. Analysis

3. Test against Quality Criteria

Focus of this session

Focus of remaining sessions

Extremely important – but only touched upon in remaining sessions

Key take-aways

- **Definition automated content analysis:** automated analysis of content (text, images) where humans are still involved, e.g., for training/supervision/validation.
- **Typical steps**:
    - Preprocessing
    - Analysis
    - Test against quality criteria

2. Getting Data into R/Python

How can I access large-scale data?

News media content

- via Application Programming Interfaces (APIs)
- via scraping
- via news media databases (e.g., Nexis, Factiva, Media Cloud)

Social media content

- via Application Programming Interfaces (APIs)
- via tracking
- via data donations

For more sources, see: Puschmann (2022)

What do I need to consider?

- Legal contexts (e.g., „*Am I allowed to scrape data sources?*“)
- Ethical aspects (e.g., „*Should I scrape data sources?*“)
- Methodological aspects (e.g., „*What bias may this data bring about?*“)

Time for R/Python!

Getting text into R/Python

R

```r
# We load data (a csv-file with ratings and content of TV series) from the Github repository
url = getURL("https://raw.githubusercontent.com/valeriehase/Salamanca-CSS-SummerSchool/
main/Processing%20text%20and%20text%20as%20data/data_tvseries.csv")
data = read.csv2(text = url)
```

Python

```python
# We load data (a csv-file with ratings and content of TV series) from the Github repository
url = "https://raw.githubusercontent.com/valeriehase/Salamanca-CSS-SummerSchool/
main/Processing%20text%20and%20text%20as%20data/data_tvseries.csv"
data = pd.read_csv(url, sep = ";")
```

Inspecting results

R

```
#Check data by inspecting first rows via head()
head(data)

# Inspect weird data in variable "Year" for first observation
data %>%
  select(Year) %>%
  slice(1)
```

A data.frame: 1 × 1

Year
<chr>
2011â€"2019

?

Python

```
#Check data by inspecting first rows via head()
data.head()

# Inspect data in variable "Year" for first observation - any issues?
data.iloc[0, 1]
```

Encoding issues

- Computers store characters (e.g., letters “w”, “o”, “r”, and “d”) using numeric codes (bytes) assigned to each character.
- For example, “word” stored as “01110111 01101111 01110010 01100100”.
- Encodings are the key for translating between numeric codes and characters.

Encoding issues

- The problem: several encodings co-exist (language-specific characters such as „ü“, „ñ“, „ğ“, or „β“; emojis, etc.)

<u>Solution:</u>

1. Preferably read in texts with the correct encoding.
2. Otherwise: cleaning/normalizing text

Encoding issues

- The problem: several encodings co-exist (language-specific characters such as „ü“, „ñ“, „ğ“, or „β“; emojis, etc.)

Solution:

1. Preferably read in texts with the correct encoding.
2. Otherwise: **cleaning/normalizing text**

Key take-aways

- **Getting data into R/Python:** many different ways
  - external .csv/.txt/.pdf file
  - API
  - scraping
  - ….
- Consider **legal/ethical/methodological** aspects of data acquisition
- Be aware of **encoding issues** (i.e., „wrong“ translation between numeric codes and characters, for example due to computer settings)

3. Preprocessing Text

Basic preprocessing steps (for an overview, see Chai, 2023)

Preprocessing: **Reducing complexity** of textual data while preserving its **substantial meaning**

Goal:

- reducing (systematic) errors (*cleaning*, often via regular expressions)
- making text comparable across different documents (*normalizing*)

Problem:

How do we reduce complexity without loosing too much meaningful information?

Basic preprocessing steps (for an overview, see Chai, 2023)

- Clean (e.g., removing formatting errors via regular expressions)
- Tokenize
- Transform to lower case
- Remove «special» characters
- Remove stopwords
- Lemmatize/stem

Basic preprocessing steps

Headline: On the state of the Germany economy: Will we have another financial crisis in Germany in 2023?

Basic preprocessing steps

Headline: On the state of the Germany economy:
Will we have another financial crisis in Germany in 2023?

Cleaning (e.g., removing formatting)

Cleaning text via regular expressions

- ***String patterns***: sequences of characters (for instance, letter)

  Example: ”`Headline`”: string pattern for the word “Headline”.

- ***Regular expressions***: string patterns with *non-literal meaning*

  Example: „`[H|h]eadline`“: regular expression to detect „Headline“ OR „headline“

→ For an extended tutorial (in R) see [here](), for R/Python see [here]().

Cleaning text via regular expressions

Regular expressions allow us to detect (and also clean) text via …

- **Logical operators** (e.g., `“&”` indicates logical condition “and”)

  Example: `“Headline & headline”` matches: “Headline” and “headline”

Cleaning text via regular expressions

Regular expressions allow us to detect (and also clean) text via …

- **Character classes** (e.g., "`[a-z]`" indicates "any lowercase letter")

  Example: "`[a-z]eadline`" matches: "headline" and "deadline"

character.classes	meaning
[a-z]	finds any letter (lowercase)
[A-Z]	finds any letter (uppercase)
[[:alpha:]]	finds any letter (lowercase and uppercase)
[0-9]	finds any number

Cleaning text via regular expressions

Regular expressions allow us to detect (and also clean) text via …

- **Quantifiers** (e.g., `"*"` indicates the preceding expression may or may not occur)

  <u>Example</u>: `"[Head]*line"` matches: "Headline" and "line"

quantifier	meaning
?	The preceding expression occurs at most once
+	The preceding expression occurs at least once
*	The preceding expression may or may not occur
{n}	The preceding expression occurs exactly n times
{n,}	The preceding expression occurs at least n times
{n,m}	The preceding expression occurs at least n times and at most m times

Cleaning text via regular expressions

With regular expressions we can….

- **Identify texts containing specific strings**

  Example: Which text contains the word “headline”?

- **Remove/replace specific strings**

  Example: Can I remove the word “headline” from my text?

- **Count how often specific stringy occur**

  Example: Can I count how often the word “headline” occurs in my text?

Time for R/Python!

Cleaning text via regular expressions

R

```r
#Let's remove the number, point and blank space before the TV series in our
#variable "Title" using gsub()
data = data %>%
  mutate(Title = gsub("^[0-9]+[[:punct:]] ", "", Title))
```

Python

```python
# Let's remove the number, point and blank space before the TV series in our
# variable "Title" using replace()
data["Title"] = data["Title"].replace("^[0-9]+\.", "", regex = True)
```

Cleaning text via regular expressions

R `"^[0-9]+[[:punct:]] "`

`^` = Check for match
at the beginning of the string

Python `"^[0-9]+\. "`

Cleaning text via regular expressions

R `"^[0-9]+[[:punct:]] "`

`[0-9]+` = Check for any number between 0-9 that occurs at least once

Python `"^[0-9]+\. "`

Cleaning text via regular expressions

R `"^[0-9]+[[:punct:]] "`

`[[.punct:]]` in R, `\.` in Python = remove the point*

Python `"^[0-9]+\. "`

*Background: re Python module does not support POSIX

Cleaning text via regular expressions

R `"^[0-9]+[[:punct:]] "`

= remove blank space

Python `"^[0-9]+\. "`

Identifying texts via regular expressions

R

```r
# Ok, let's have some fun with this.
# Using the grepl() function, we find all TV series
# that contain the word "drama" in the variable "Description".
# We use filter() to identify these observations.
data %>%

  #filter all observations containing the word "drama"
  filter(grepl("[D|d]rama", Description)) %>%

  # see first 5 rows of data set
  head(5)
```

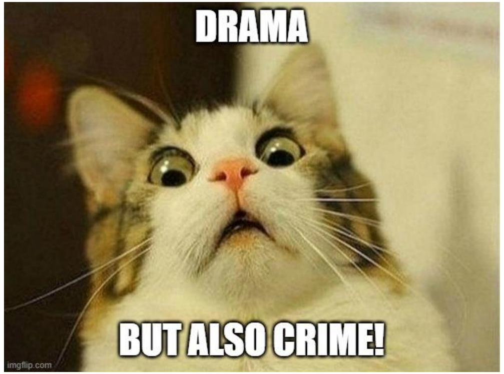
Python

```python
# Ok, let's have some fun with this.
# Using the str.contains() function, we identify all TV series
# that contain the word "drama" in the variable "Description".
data[data["Description"].str.contains("[D|d]rama")].head()
```

Identifying texts via regular expressions

R

```r
#Let's get all observations that contain the word
# "drama" or the word "crime" in the variable "Description"
data %>%

  #filter all observations containing the word "drama"
  filter(grepl("[D|d]rama|[C|c]rime", Description)) %>%

  # see first rows of data set
  head(5)
```



Python

```python
#Let's get all observations that contain the word
# "drama" or the word "crime" in the variable "Description"
data[data["Description"].str.contains("[D|d]rama|[C|c]rime")].head()
```

Your turn!

Can you...

- **Identify** all series that play in Spain?
- **Identify** all series that deal with superheroes & **replace** the term "superhero/superheroes” in the variable with "Description" with "fancy R/Python programmers"?

Identify all series that play in Spain

R

```r
# Your turn!
# Can you identify all series that play in Spain?
data %>%
  filter(grepl("in Spain", Description)) %>%
  head(5)
```

Python

```python
# Your turn!
# Can you identify all series that play in Spain?
data[data["Description"].str.contains("in Spain")]
```

Identify & replace superhero-related terms

R

```
# Your turn!
# Can you identify all series that deal with superheroes
# and replace the term "superhero/superheroes in the variable "Description"
# with "fancy R programmers"?
data %>%
    filter(grepl("[S|s]uperhero[es]*", Description)) %>%
      mutate(Description = gsub("[S|s]uperhero[es]* ", "fancy R programmers ", Description)) %>%
      select(Description) %>%
      head()
```

Python

```
# Your turn!
# Can you identify all series that deal with superheroes
# and replace the term "superhero/superheroes in the variable "Description"
# with "fancy Python programmers"?
data["Description"].str.replace("[S|s]uperhero[es]* ", "fancy Python programmers", regex = True).head()
```

Your turn!

A data.frame: 3 × 1

Description
<chr>
A group of vigilantes set out to take down corrupt fancy R programmers who abuse their superpowers.
An adult animated series based on the Skybound/Image comic about a teenager whose father is the most powerful fancy R programmers on the planet.
The adventures of Superman's cousin and her own fancy R programmers career.

Short break

Basic preprocessing steps (for an overview, see Chai, 2023)

- ~~Clean (e.g., removing formatting errors via regular expressions)~~
- Tokenize (break text down to single string patterns: features)
- Transform to lower case
- Remove «special» characters (e.g., numbers, punctuation)
- Remove stopwords
- Lemmatize/stem

Basic preprocessing steps

On the state of the German economy:

Will we have another financial crisis in Germany in 2023?

Tokenize

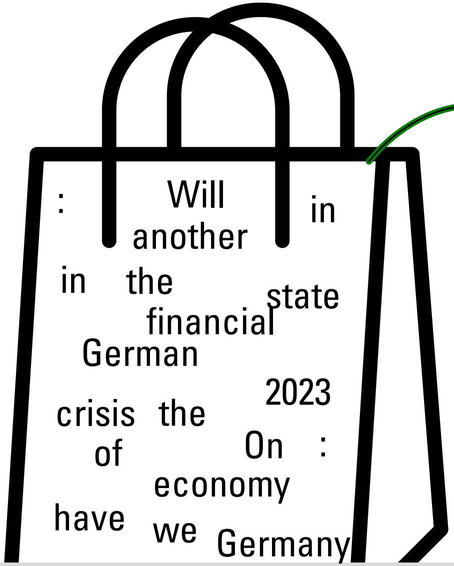
On: 1
the: 2
state: 1
of: 1
German: 1
economy: 1
:: 1

Will: 1
we: 1
have: 1
another: 1
financial: 1
crisis: 1
in: 2
Germany: 1
2023: 1
?: 1

Basic preprocessing steps

On the state of the German economy:

Will we have another financial crisis in Germany in 2023?



Bag-of-words assumption

On: 1

the: 2

state: 1

of: 1

German: 1

economy: 1

……..

Basic preprocessing steps

On the state of the German economy:

Will we have another financial crisis in Germany in 2023?

Bag-of-words assumption

Through tokenization (i.e. reducing text to unique features), we assume that order & context of features have no influence on interpretation.

Basic preprocessing steps

On the state of the German economy:

Will we have another financial crisis in Germany in 2023?

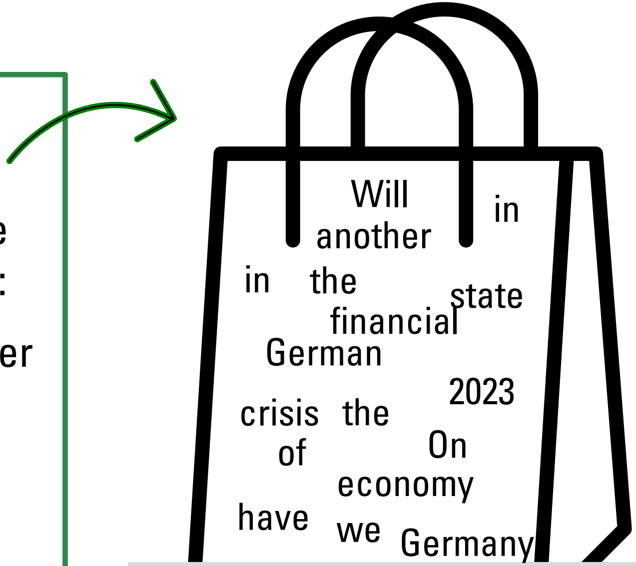
Bag-of-words assumption

Can you think of any examples where this assumption may be violated?

Basic preprocessing steps

On the state of the German economy:

Will we have another financial crisis in Germany in 2023?



Bag-of-words assumption

The order/context of features matters!

This is why we move to other text-as-data representations that relax this assumption (later).

Basic preprocessing steps

On the state of the German economy:

Will we have another financial crisis in Germany in 2023?

Transform to lower case

On: 1
the: 2
state: 1
of: 1
German: 1
economy: 1
:: 1

Will: 1
we: 1
have: 1
another: 1
financial: 1
crisis: 1
in: 2
Germany: 1
2023: 1
?: 1

Basic preprocessing steps

On the state of the German economy:

Will we have another financial crisis in Germany in 2023?

Transform to lower case

on: 1
the: 2
state: 1
of: 1
german: 1
economy: 1
:: 1

will: 1
we: 1
have: 1
another: 1
financial: 1
crisis: 1
in: 2
germany: 1
2023: 1
?: 1

Is lowercasing always helpful?

- Oftentimes, lowercasing does not alter the meaning of features:
  - „In this seminar, you will spend some hours on **learning** R.“
  - „**Learning** R will be something you will spend some hours on in this seminar.“

- <u>However, exceptions:</u>

Bild vs. BILD

Basic preprocessing steps

On the state of the German economy:

Will we have another financial crisis in Germany in 2023?

Remove „special“ characters (punctuation, numbers, etc.)

on: 1
the: 2
state: 1
of: 1
german: 1
economy: 1
:: 1

will: 1
we: 1
have: 1
another: 1
financial: 1
crisis: 1
in: 2
germany: 1
2023: 1
?: 1

Basic preprocessing steps

On the state of the German economy:

Will we have another financial crisis in Germany in 2023?

Remove „special“ characters (punctuation, numbers, etc.)

on: 1
the: 2
state: 1
of: 1
german: 1
economy: 1

will: 1
we: 1
have: 1
another: 1
financial: 1
crisis: 1
in: 2
germany: 1

Is removing special characters always helpful?

● Oftentimes, signs, numbers, punctuations may not add meaning.

<u>However, **a lot of** exceptions</u>

- #metoo
- Is this true ???!!!!!!!
- 👍🤬🥳🙈
- www.nzz.ch
- G7

Basic preprocessing steps

On the state of the German economy:

Will we have another financial crisis in Germany in 2023?

Remove stopwords

on: 1
the: 2
state: 1
of: 1
german: 1
economy: 1

will: 1
we: 1
have: 1
another: 1
financial: 1
crisis: 1
in: 2
germany: 1

Is removing stopwords always helpful?

- There is no consensus definition of „stop words“ – heavily depends on context!
- Some packages provide off-the-shelf lists of stopwords

Example – List of stopwords provided by R package quanteda

```
> stopwords("english")
  [1] "i"          "me"         "my"         "myself"     "we"         "our"
 [10] "your"       "yours"      "yourself"   "yourselves" "he"         "him"
 [19] "her"        "hers"       "herself"    "it"         "its"        "itself"
 [28] "theirs"     "themselves" "what"       "which"      "who"        "whom"
 [37] "those"      "am"         "is"         "are"        "was"        "were"
 [46] "have"       "has"        "had"        "having"     "do"         "does"
 [55] "should"     "could"      "ought"      "i'm"        "you're"     "he's"
 [64] "they're"    "i've"       "you've"     "we've"      "they've"    "i'd"
 [73] "we'd"       "they'd"     "i'll"       "you'll"     "he'll"      "she'll"
 [82] "aren't"     "wasn't"     "weren't"    "hasn't"     "haven't"    "hadn't"
 [91] "won't"      "wouldn't"   "shan't"     "shouldn't"  "can't"      "cannot"
[100] "that's"     "who's"      "what's"     "here's"     "there's"    "when's"
```

Is removing stopwords always helpful?

- There is no consensus definition of „stop words" – heavily depends on context!
- Some packages provide off-the-shelf lists of stopwords
- However, use with great care – often better to keep stopwords and/or define organic list

? Can you think of any example where words like „we", „our" etc. could be very meaningful?

Basic preprocessing steps

On the state of the German economy:

Will we have another financial crisis in Germany in 2023?

Remove stopwords

on: 1
the: 2
state: 1
of: 1
german: 1
economy: 1

will: 1
we: 1
have: 1
another: 1
financial: 1
crisis: 1
in: 2
germany: 1

Basic preprocessing steps

On the state of the German economy:

Will we have another financial crisis in Germany in 2023?

Remove stopwords

state: 1

another: 1

financial: 1

german: 1

crisis: 1

economy: 1

germany: 1

Basic preprocessing steps

On the state of the German economy:

Will we have another financial crisis in Germany in 2023?

Lemmatizing/stemming

state: 1

another: 1

financial: 1

german: 1

crisis: 1

economy: 1

germany: 1

Is lemmatizing/stemming always helpful?

Stemming: Reduces words to their base form/roots by removing the suffix:

- „running“ „runs“ „run“ → „run“

<u>However, does not always work/help</u>

„run“ „ran“ → „run“ „ran“

Lemmatizing: Reduced words to their lemma (dictionary form)

„running“ „ran“ → „run“

Basic preprocessing steps

On the state of the German economy:

Will we have another financial crisis in Germany in 2023?

Lemmatizing/stemming

state: 1

german: 1

economy: 1

another: 1

financial: 1

crisis: 1

germany: 1

Basic preprocessing steps

On the state of the German economy:

Will we have another financial crisis in Germany in 2023?

Lemmatizing/stemming

state: 1

another: 1

financial: 1

german: 1

crisis: 1

economy: 1

germani: 1

The order of preprocessing steps

On the state of the German economy:

Will we have another financial crisis in Germany in 2023?

?

Can you think of any examples where the order of preprocessing steps would matter?

The order of preprocessing steps

On the state of the German economy:

Will we have another financial crisis in Germany in 2023?

?

Yes – for example, if you **first** stem features («yourselves» → «yourselv»), **then** try to remove stopwords (feature included as stopword «yourselves», not «yourselv»), stopword will not be recognized!

Text before and after „basic“ preprocessing

On the state of the German economy:

Will we have another financial crisis in Germany in 2023?

Processing

state: 1

another: 1

financial: 1

german: 1

crisis: 1

economy: 1

germani: 1

Time for R/Python!

Normalizing text

R

```r
#Run preprocessing steps using tokens() and subfunctions
tokens <- tokens(data$Description,
                 what = "word",
                 remove_punct = TRUE,
                 remove_numbers = TRUE) %>%
          tokens_tolower() %>%
          tokens_remove(stopwords("english")) %>%
          tokens_wordstem()
```

Python

```python
# Initialize the stop words and stemmer
stop_words = set(stopwords.words("english"))
stemmer = PorterStemmer()

#Write a function that contains all necessary preprocessing steps
def clean_description(description):
    # Tokenize the description
    words = word_tokenize(description)
    # Remove special signs and convert to lower case
    words = [word.lower() for word in words if word.isalpha()]
    # Remove stopwords
    words = [word for word in words if word not in stop_words]
    # Apply stemming
    words = [stemmer.stem(word) for word in words]
    return words

tokens = [clean_description(description) for description in data["Description"]]
```

Normalizing text

R

```r
#Look at original first text
data$Description[1]
#Look at preprocessed first text
tokens[1]
```

Python

```python
#Look at original first text
data["Description"].iloc[0]
#Look at preprocessed first text
tokens[0]
```

Text before and after „basic“ preprocessing

Nine noble families fight for control over the lands of Westeros, while an ancient enemy returns after being dormant for millennia.

Processing

nine
nobl
famili
fight
control
land

westero
ancient
enemi
return
dormant
millenia

Your turn!

Can you...

- **Create** a list of 3-5 stop-words you think are unique to this corpus and **remove these** as part of the existing preprocessing pipeline?
- Scroll through the descriptions – would there be any content-related reason **not to use lowercasing**?

Identify & remove unique stopwords

R

```r
unique_stopwords = c("one", "two", "three", "four", "five")

tokens <- tokens(data$Description,
                 what = "word",
                 remove_punct = TRUE,
                 remove_numbers = TRUE) %>%
        tokens_tolower() %>%
        tokens_remove(stopwords("english")) %>%
        tokens_remove(unique_stopwords) %>% #simply add here
        tokens_wordstem()
```

Python

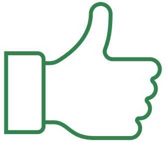
```python
unique_stopwords = ["one", "two", "three", "four", "five"]

#Write a function that contains all necessary preprocessing steps
def clean_description(description):
    # Tokenize the description
    words = word_tokenize(description)
    # Remove special signs and convert to lower case
    words = [word.lower() for word in words if word.isalpha()]
    # Remove stopwords
    words = [word for word in words if word not in stop_words]
    # Remove unique list of stopwords
    words = [word for word in words if word not in unique_stopwords]
    # Apply stemming
    words = [stemmer.stem(word) for word in words]
    return words

tokens = [clean_description(description) for description in data["Description"]]
```

Reflect on lowercasing

- "Follows the personal and professional lives of six twenty to thirty year-old friends living in the **Manhattan** borough of **New York City**." (Text 4)
- "**Sheriff Deputy Rick Grimes** wakes up from a coma to learn the world is in ruins and must lead a group of survivors to stay alive." (Text 5)

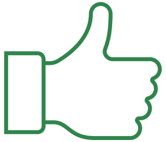
→ Like other processing steps, lowercasing can "throw away" important information, for example which features identify locations or actors.

Promises & challenges of preprocessing

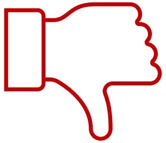


- Speeding up the analysis by focusing on meaningful features (and removing irrelevant ones)
- Normalizing text (e.g., across data sources) to increase comparability

Promises & challenges of preprocessing



- Speeding up the analysis by focusing on meaningful features (and removing irrelevant ones)
- Normalizing text (e.g., across data sources) to increase comparability



- Degrees of freedom in how to conduct preprocessing (meaningful vs. irrelevant features, order of preprocessing steps)
- Preprocessing can have downstream effects on analysis (Denny & Spirling, 2018; Maier et al., 2020), including introducing systematic measurement error

Promises & challenges of preprocessing

- Carefully consider **which** preprocessing steps you apply – depends on the type of data & what information you consider “meaningful” for your analysis.
- Carefully consider **in which order** you apply preprocessing steps (negative downstream effects possible!).

Key Take-Aways

- **Preprocessing**: includes cleaning & normalization to ensure that features (e.g., words) are comparable (e.g., to recognize similar/the same words) and relevant.
- **Features**: Unit of analysis to which texts are broken down (often: unique words)
- **Tokenization**: Process of breaking down texts to features
- **Bag of words assumption**: Assumption that the order and context of features have no influence on their interpretation.

4. Choosing a Text-as-Data Representation

Bag-of-words representation

On the state of the German economy:

Will we have another financial crisis in Germany in 2023?

Processing

state: 1

another: 1

financial: 1

german: 1

crisis: 1

economy: 1

germani: 1

Document-feature-matrix

Doc 1
On the state of the German economy: Will we have another financial crisis in Germany in 2023?

Doc 2
The BIP is going down, unemployment is up: Crisis ahead in Germany?

Doc 3
German economy tumbles into crisis: Goverment under pressure

Preprocessing

	economi		german	down
Doc 1	?		1	0
Doc 2	?		0	1
Doc 3	?		1	0

Which values should be in these cells?

?

Document-feature-matrix

Doc 1
On the state of the German economy: Will we have another financial crisis in Germany in 2023?

Doc 2
The BIP is going down, unemployment is up: Crisis ahead in Germany?

Doc 3
German economy tumbles into crisis: Goverment under pressure

Preprocessing

	economi	crisi	germani	german	down
Doc 1	**1**	1	1	1	0
Doc 2	**0**	1	1	0	1
Doc 3	**1**	1	0	1	0

Document-feature-matrix

- A matrix where rows identify texts, columns its features, and cells feature frequencies
- Text-as-data: turns text into numeric data through bag-of-words assumption
- Can be used as “input” for different methods, e.g., dictionaries or machine learning
- Consider: with little preprocessing, this matrix will consider many zeroes (“sparsity”), which can make computing inefficient

Time for R/Python!

Creating document-feature-matrizes (dfms)

R

```r
dfm = tokens %>%
  dfm()

#check result
dfm
```

Python

```python
#Write a new dfm function that contains all necessary preprocessing steps
def clean_description_dfm(description):
    # Tokenize the description
    words = word_tokenize(description)
    # Remove special signs and convert to lower case
    words = [word.lower() for word in words if word.isalpha()]
    # Remove stopwords
    words = [word for word in words if word not in stop_words]
    # Apply stemming
    words = [stemmer.stem(word) for word in words]
    #Additionally re-join as string
    return ' '.join(words)  # Join the tokens back into a single string

tokens_dfm = [clean_description_dfm(description) for description in data["Description"]]

#Create a document-feature matrix
vectorizer = CountVectorizer()
dfm = vectorizer.fit_transform(tokens_dfm)

#print the result in dense format
pd.DataFrame(dfm.todense(), columns = vectorizer.get_feature_names_out()).head()
```

Example dfm (in R)

R

```
Document-feature matrix of: 900 documents, 4,243 features (99.66% sparse) and 0 docvars.
       features
docs     nine nobl famili fight control land westero ancient enemi return
  text1     1    1      1     1       1    1       1       1     1      1
  text2     0    0      1     0       0    0       0       0     0      0
  text3     0    0      0     0       0    0       0       0     0      0
  text4     0    0      0     0       0    0       0       0     0      0
  text5     0    0      0     0       0    0       0       0     0      0
  text6     0    0      0     0       0    0       0       0     0      0
[ reached max_ndoc ... 894 more documents, reached max_nfeat ... 4,233 more features ]
```

Checking top features

R

```r
#Check most frequent features
topfeatures = topfeatures(dfm, 10) %>%
  as.data.frame() %>%
  rename("count" = '.')

topfeatures
```

Python

```python
# Convert dfm to a dense format for calculation
dfm_dense = dfm.toarray()

# Get feature names
feature_names = vectorizer.get_feature_names_out()

#Check most frequent features
def top_features(matrix, feature_names, top_n):
    # Sum the occurrences of each feature
    feature_sums = np.sum(matrix, axis = 0)
    # Create a data frame to hold feature names and their corresponding sums
    feature_sums_df = pd.DataFrame({'feature': feature_names, 'count': feature_sums})
    # Sort the data frame by count in descending order and get the top N features
    top_features_df = feature_sums_df.sort_values(by = "count", ascending = False).head(top_n)
    return top_features_df

topfeatures = top_features(dfm_dense, feature_names, 10)

topfeatures
```

Example top features (in Python)

Python

	feature	count
1993	life	108
2013	live	108
1229	famili	107
2362	new	103
3871	world	75
1308	follow	74
3896	young	74
1358	friend	70
1278	find	69
3082	seri	65

Creating a word cloud

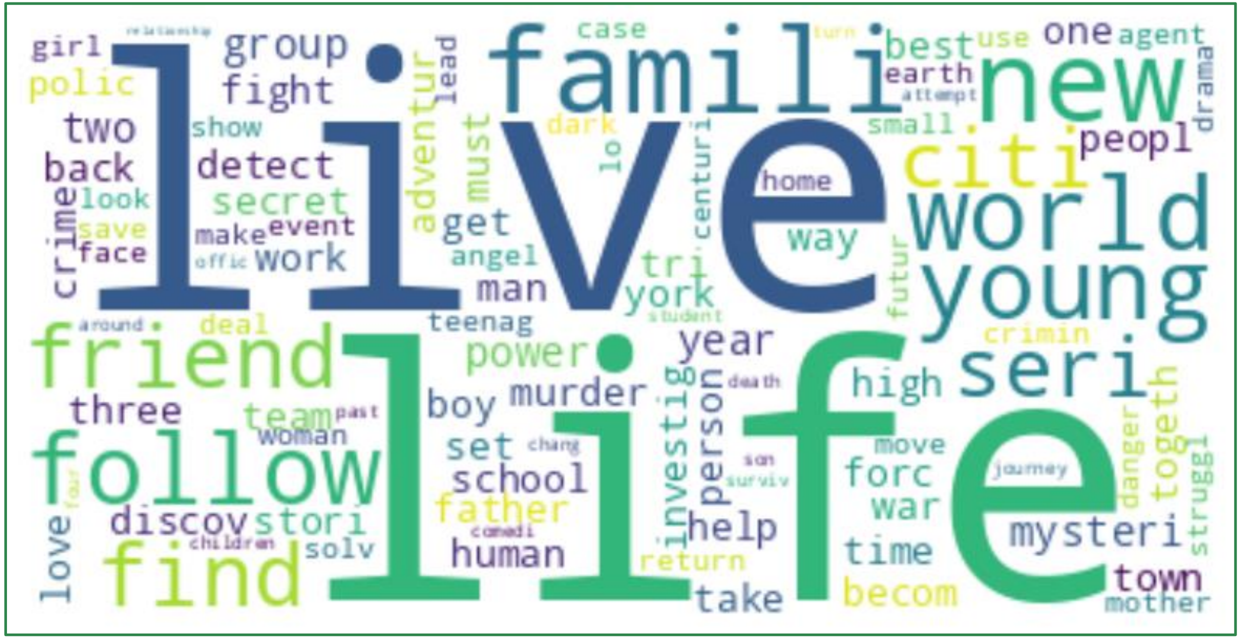
R

```r
#Visualize results with a word cloud
textplot_wordcloud(dfm, max_words = 100)
```

Python

```python
#get feature sums
feature_sums = np.sum(dfm_dense, axis=0)

# Create a dictionary of features and their corresponding sums
feature_counts = dict(zip(feature_names, feature_sums))

# Generate a word cloud
wordcloud = WordCloud(max_words = 100, background_color = "white").generate_from_frequencies(feature_counts)

# Display the word cloud using matplotlib
plt.figure(figsize = (10, 5))
plt.imshow(wordcloud, interpolation = "bilinear")
plt.axis("off")
plt.show()
```

A cautionary note on wordclouds



Short break

Going beyond „bag-of-words"

Likely ❌ wrong assumption that we can..

- "treat every word as having a distinct, unique meaning" (Grimmer et al., 2022, p. 79)
- represent text "as if it were a bag of words, [...] an unordered set of words with their position ignored, keeping only their frequency in the document."(Jurafsky & Martin, 2023, p. 60)

Going beyond „bag-of-words“

Likely ❌ violated / not helpful when dealing with…

- **Polysemy**: “I love this sound.” vs. “Sound solution!”
- **Negation**: “Not bad!”
- **Named Entities**: “United States”, “Olaf Scholz”
- **Features with similar meanings**: “I like greens.” vs. “I like vegetables.”

Going beyond „bag-of-words“

Solutions to relax this assumption include…

- Relying on **ngrams** (e.g., collocations)
- Relying on **syntax** (e.g., part-of-speech tagging)
- Relying on **semantic spaces** (e.g., word embeddings)

Going beyond „bag-of-words“

Solutions to relax this assumption include…

- Relying on **ngrams** (e.g., collocations)
- Relying on **syntax** (e.g., part-of-speech tagging)
- Relying on **semantic spaces** (e.g., word embeddings)

Ngrams

- Instead of using unigrams (i.e., single words) as features, we can use bigrams, trigrams, etc. as features
  - Unigram: “that”
  - Bigram: “that is”
  - Trigram: “that is great”

- This includes collocations as sequences of features which symbolize shared semantic meaning and often co-occur, e.g., “United States”

Time for R/Python!

Identifying collocations

R

```r
# Get most frequent collocations
tokens %>%
  textstat_collocations(min_count = 10) %>%
  arrange(-lambda) %>%
  head(10)
```

Python

```python
# Flatten the list of lists into a single list of tokens
all_tokens = [token for sublist in tokens for token in sublist]

# Find bigram collocations
finder = BigramCollocationFinder.from_words(all_tokens)

# Filter out bigrams that occur less than 10 times
finder.apply_freq_filter(10)

# Score the bigrams using the likelihood ratio
scored = finder.score_ngrams(BigramAssocMeasures.likelihood_ratio)

# Convert to a DataFrame for easier manipulation
scored_df = pd.DataFrame(scored, columns = ["bigram", "likelihood_ratio"])

# Sort by the likelihood ratio in descending order and take the top 10
top_10_collocations = scored_df.sort_values(by = "likelihood_ratio", ascending=False).head(10)

# Print the top 10 collocations
top_10_collocations
```

Identifying collocations (in R)

A collocations: 9 × 6

collocation	count	count_nested	length	lambda	z
<chr>	<int>	<int>	<dbl>	<dbl>	<dbl>
los angel	22	0	2	11.981406	7.848877
new york	39	0	2	9.623992	6.736654
serial killer	10	0	2	8.654799	11.833936
person profession	13	0	2	7.806213	12.174181
antholog seri	10	0	2	7.621656	8.599479
best friend	25	0	2	7.041008	15.126324
high school	22	0	2	7.030228	16.464804
york citi	19	0	2	5.799736	16.040393
seri follow	10	0	2	4.312310	11.333288

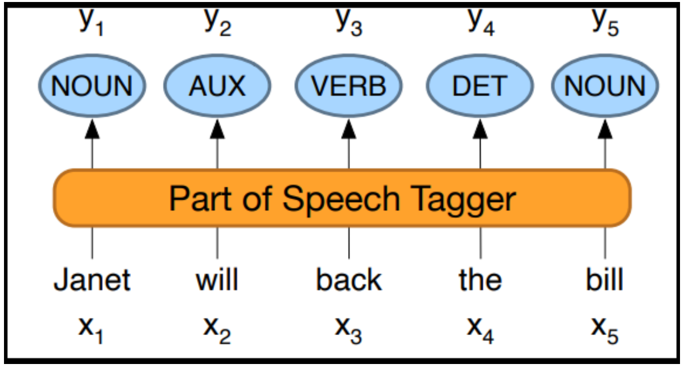
Going beyond „bag-of-words“

Solutions to relax this assumption include…

- **Relying on ngrams (e.g., collocations)**
- Relying on **syntax** (e.g., part-of-speech tagging)
- Relying on **semantic spaces** (e.g., word embeddings)

Relying on syntax: Part-of-speech tagging

- Part-of-speech tagging as the "process of assigning a part-of-speech to each word in a text" (Jurafsky & Martin, 2023, p. 163)
- Tags based on feature & context, can for example be used to identify named entities



Note. Figure from Jurafsky & Martin (2023, p. 164).
For explanation of tags, see de Marneffe et al. (2021).

Time for R/Python!

Part-of-Speech Tagging

R

```r
data$Description %>%

  #change format for udpipe package
  as_tibble() %>%
  mutate(doc_id = paste0("text", 1:n())) %>%
  rename(text = value) %>%

  #for simplicity, run for fewer documents
  slice(1) %>%

  #part-of-speech tagging, include only related variables
  udpipe("english") %>%
  select(doc_id, sentence_id, token_id, token, upos) %>%
  head(10)
```

Python

```python
# For simplicity, run for fewer documents
sample = data.head(1)

# Part-of-speech tagging, include only related variables
pos_tags = []
for idx, row in sample.iterrows():
    doc = nlp(row["Description"])
    for sent in doc.sents:
        for token in sent:
            pos_tags.append({
                'sentence_id': sent.start,
                'token_id': token.i,
                'token': token.text,
                'upos': token.pos_
            })

# Convert the list of dictionaries to a DataFrame
pos_df = pd.DataFrame(pos_tags)

# Display the first 10 rows
pos_df.head(10)
```

Part-of-Speech Tagging

R

A data.frame: 10 × 5

	doc_id	sentence_id	token_id	token	upos
	<chr>	<int>	<chr>	<chr>	<chr>
1	text1	1	1	Nine	PROPN
2	text1	1	2	noble	ADJ
3	text1	1	3	families	NOUN
4	text1	1	4	fight	VERB
5	text1	1	5	for	ADP

Python

	sentence_id	token_id	token	upos
0	0	0	Nine	NUM
1	0	1	noble	ADJ
2	0	2	families	NOUN
3	0	3	fight	VERB
4	0	4	for	ADP

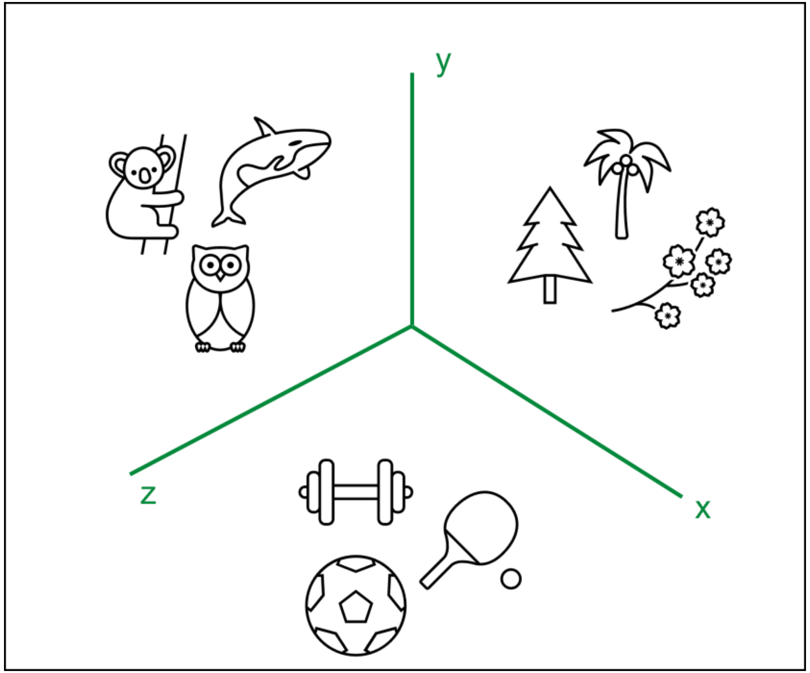
Going beyond „bag-of-words“

Solutions to relax this assumption include…

- Relying on ngrams (e.g., collocations)
- Relying on **syntax** (e.g., named entity recognition, dependency parsing)
- Relying on **semantic spaces** (e.g., word embeddings)

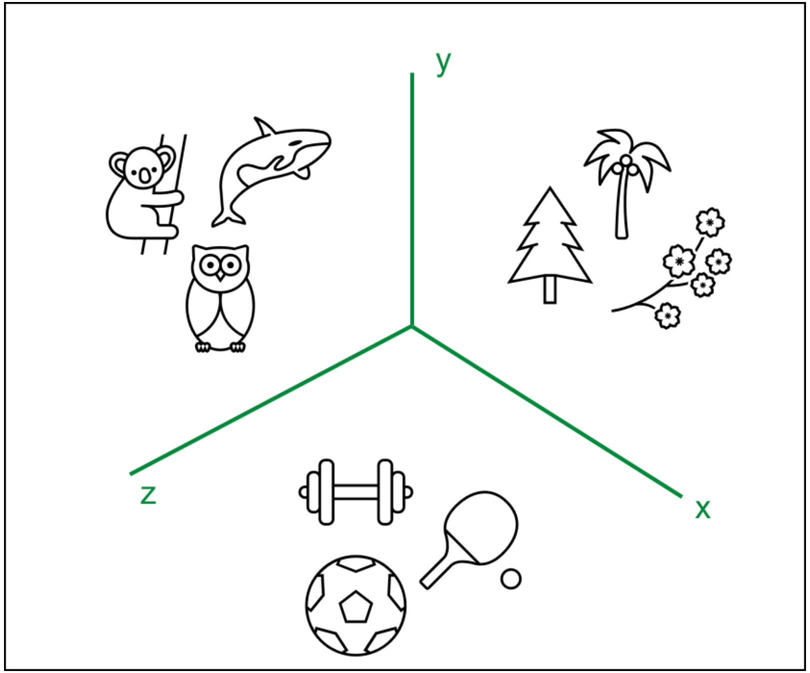
Semantic spaces: word embeddings

Embeddings as **dense vectors** for representing words in a **N-dimensional space**, with these dimensions encoding meaning



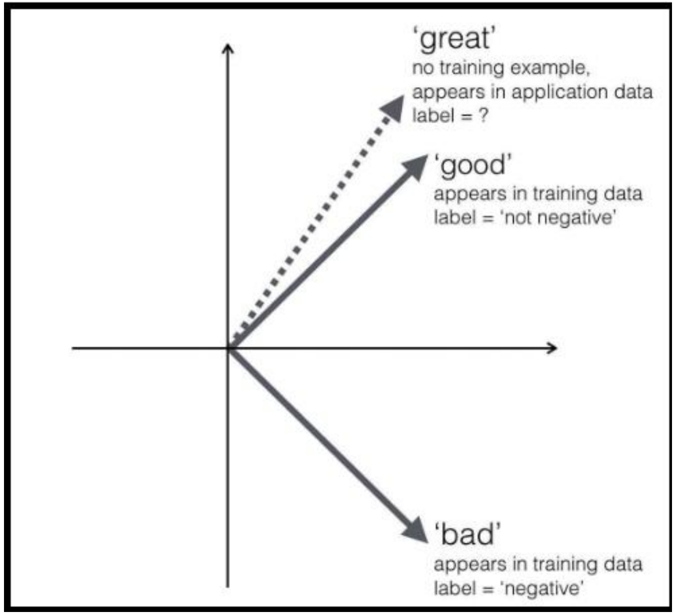
Semantic spaces: word embeddings

Word embeddings “predict a focal word as a function of the other words that appear within a small window of that focal word” (Rodriguez et al., 2023, p. 3)



Semantic spaces: word embeddings

Word embeddings "predict a focal word as a function of the other words that appear within a small window of that focal word" (Rodriguez et al., 2023, p. 3)



Note. Figure from Rudkowsky et al. (2018, p. 144).

Key Take-Aways

- **Document-feature-matrix**: matrix where rows identify texts, columns its features, and cells feature frequencies
- **Collocations**: sequences of features which symbolize shared semantic meaning and often co-occur, e.g. “United States”
- **Part-of-speech tagging**: assigning a part-of-speech (e.g., “verb”, “noun”) to features
- **Word embeddings**: dense vectors for representing words in a N-dimensional space

5. Take Away & Outlooks

Unboxing „Magic“: Typical Steps

1. Preprocessing

2. Analysis

3. Test against Quality Criteria

Focus of this session

Focus of remaining sessions

Extremely important – but only touched upon in remaining sessions

Overview of methods

(1) Classifying content in pre-defined categories:

Rule-based approaches, dictionaries → Johannes Gruber

Supervised Machine Learning → Damian Trilling & Johannes Gruber

(2) Exploring content without pre-defined categories:

Unsupervised Machine Learning → me again

deductive

inductive

Typical (wrong) assumptions

We can let programming scripts do all the work, without human supervision or intervention.

Automated "methods augment humans, not replace them" (Grimmer & Stewart, 2013, p. 270). Automated methods can even **increase** the workload due to additional human supervision or intervention.

Typical (wrong) assumptions

Automated methods democratize research: Everyone can learn and apply these methods!

Yes, but with limitations such as „English before everything“ (Baden et al., 2022, p. 9) – unfortunately, a lot of methods were developed for specific types of data (Hase et al., 2023) or languages (Baden et al., 2022).

Typical (wrong) assumptions

Using manifest indicators measured and modelled via automated methods, we can correctly represent latent theoretical constructs.

The issue of (systematic) error: “All quantitative models of language are wrong—but some are useful” (Grimmer & Stewart, 2013, p. 269) or the issue of „technology before validation“ (Baden et al., 2022, p. 3).

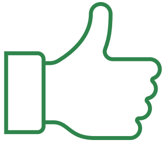
Typical (wrong) assumptions

We have to choose a single correct method for measuring latent theoretical concepts of interest.

Optimistic view: “There is no globally best method” (Grimmer & Stewart, 2013, p. 270) – what is deemed „correct“ varies across epistemologies or data.

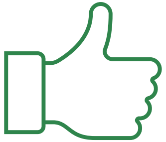
Less optimistic view: „Specialization before integration“ (Baden et al., 2022, p. 6)

Promises

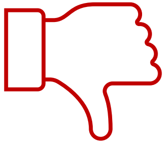


- Identifying theoretically important differences based on large-scale analyses (e.g., across countries, time)
- Exploring new types of data & variables
- Interdisciplinary perspectives on theories & measurements

Promises & Challenges



- Identifying theoretically important differences based on large-scale analyses (e.g., across countries, time)
- Exploring new types of data & variables
- Interdisciplinary perspectives on theories & measurements



- More of the same: text-focused, Western bias
- More of the same, but worse: agreement on quality criteria?

Thanks!
Any Questions?

Dr. Valerie Hase
IfKW, LMU Munich

orcid.org/0000-0001-6656-4894

valeriehase

@valeriehase.bsky.social

www.valerie-hase.com

References

- Baden, C., Pipal, C., Schoonvelde, M., & Van Der Velden, M. A. C. G. (2022). Three Gaps in Computational Text Analysis Methods for Social Sciences: A Research Agenda. *Communication Methods and Measures*, *16*(1), 1–18. https://doi.org/10.1080/19312458.2021.2015574
- Benoit, K. (2020). Text as Data: An Overview. L. Cuirini & R. Franzese (eds.), *Handbook of Research Methods in Political Science and International Relations* (pp. 461–497). Thousand Oaks: Sage.
- Chai, C. P. (2023). Comparison of Text Preprocessing Methods. *Natural Language Engineering*, *29*(3), 509–553. https://doi.org/10.1017/S1351324922000213
- De Marneffe, M.-C., Manning, C. D., Nivre, J., & Zeman, D. (2021). Universal Dependencies. *Computational Linguistics*, 1–54. https://doi.org/10.1162/coli_a_00402
- Denny, M. J., & Spirling, A. (2018). Text Preprocessing For Unsupervised Learning: Why It Matters, When It Misleads, and What to Do about It. *Political Analysis*, *26*(2), 168–189. https://doi.org/10.1017/pan.2017.44
- Grimmer, J., Roberts, M. E., & Stewart, B. M. (2022). *Text as Data: A New Framework for Machine Learning and the Social Sciences*. Chapter 6 on "Bag of Words". Princeton University Press.
- Grimmer, J., & Stewart, B. M. (2013). Text as Data: The Promise and Pitfalls of Automatic Content Analysis Methods for Political Texts. *Political Analysis*, *21*(3), 267–297. https://doi.org/10.1093/pan/mps028
- Hase, V. (2023). Automated Content Analysis. In F. Oehmer-Pedrazzi, S. H. Kessler, E. Humprecht, K. Sommer, & L. Castro (Eds.), *Standardisierte Inhaltsanalyse in der Kommunikationswissenschaft – Standardized Content Analysis in Communication Research* (pp. 23–36). Springer Fachmedien Wiesbaden. https://doi.org/10.1007/978-3-658-36179-2_3
- Hase, V., Mahl, D., & Schäfer, M. S. (2023). The "computational turn": An "interdisciplinary turn"? A systematic review of text as data approaches in journalism studies. *Online Media and Global Communication*, *2*(1), 122–143. https://doi.org/10.1515/omgc-2023-0003

References

- Jurafsky, D., & Martin, J. H. (2023). *Speech and Language Processing. An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. https://web.stanford.edu/~jurafsky/slp3/ed3book_jan72023.pdf
- Maier, D., Niekler, A., Wiedemann, G., & Stoltenberg, D. (2020). How Document Sampling and Vocabulary Pruning Affect the Results of Topic Models. *Computational Communication Research*, *2*(2), 139–152. https://doi.org/10.5117/CCR2020.2.001.MAIE
- Puschmann (2022). Ultimate 🚀 Data Sources for [Textual] Content Analysis in Communication and Media Research (UDS4CA). Retrieved via https://docs.google.com/document/d/1pfEDiIU6iDbrbMSnfkTgDsZBAY5h02zmVYGPZIB25gE/edit#heading=h.c9dklerp24c4
- Rodriguez, P. L., Spirling, A., & Stewart, B. M. (2023). Embedding Regression: Models for Context-Specific Description and Inference. *American Political Science Review*, 1–20. https://doi.org/10.1017/S0003055422001228
- Rudkowsky, E., Haselmayer, M., Wastian, M., Jenny, M., Emrich, Š., & Sedlmair, M. (2018). More than Bags of Words: Sentiment Analysis with Word Embeddings. *Communication Methods and Measures*, *12*(2–3), 140–157. https://doi.org/10.1080/19312458.2018.1455817
- Van Atteveldt, W., Trilling, D., & Calderón, C. A. (2022). *Computational Analysis of Communication*. Chapter 9 on “Processing Text” and Chapter 10 on “Text as Data”. Wiley Blackwell. Link to online chapter 9 and Link to online chapter 10.
- Welbers, K., van Atteveldt, W., & Benoit, K. (2017). Text Analysis in R. *Communication Methods and Measures*, *11*(4), 245–265. https://doi.org/10.1080/19312458.2017.1387238