Text as Data: Foundations & Preprocessing
Multimodal Computational Methods in Political Science

Tamara Grechanaya [1]

[1]LMU Munich — Computational Social Science MA Program

April, 2026

What is this course about?

We learn how to turn **political communication in unstructured form**—text, images, video, and audio—into **data we can analyze computationally** in order to study and explain political phenomena.

Data sources

Social media posts
Party manifestos
Parliamentary speeches
Press releases
Images, video, audio

What we do

Preprocess raw data
Turn documents and media into features
Compare texts, actors, and campaigns
Detect patterns at scale

What we can learn

Ideology and framing
Polarization and persuasion
Agenda-setting and conflict
Protest, strategy, and public opinion

From raw communication to measurable political evidence

How can computational methods reveal politics?

Computational methods turn **words, images, and sound** into evidence about **power, strategy, persuasion, and protest**. Some examples of what this looks like in practice:

The 450M-post distraction

China's government fabricates ~450M social media posts/year — but they don't argue with critics. They change the subject during sensitive moments.

King, Pan & Roberts (2017), APSR

Leaked posts + text classification

Polarization you can hear

In the 1870s, one sentence of Congressional speech told you nothing about party. Today it predicts party with ~75% accuracy — polarization is audible.

Gentzkow, Shapiro & Taddy (2019), Econometrica

High-dimensional text scaling

Contact backfires

Paying Republicans to follow liberal Twitter accounts made them more conservative, not less. Cross-partisan exposure can deepen divides.

Bail et al. (2018), PNAS

Field experiment with bots

Anger pays — but not for women

Analyzing facial expressions in televised debates: when male candidates show anger, voters reward them. When women do the same, voters punish them. Transcripts alone miss this entirely.

Boussalis, Coan, Holman & Müller (2021), APSR

Computer vision + audio + text

Seeing protest, explainably

A vision model analyzing 141,000 protest photos that tells you not just *"this is a protest,"* but which visual elements — flags, banners, candles, police — drove the decision. Interpretable AI for politics.

Scholz et al. (2025), Political Analysis

Two-level image classification

Why is this useful beyond academia?

Every industry now drowns in **unstructured text, images, and audio** — customer reviews, news coverage, social media, call transcripts, internal documents. The methods you learn here are exactly what's needed to turn that mess into evidence.

Consulting & policy

Measure how a reform is discussed in the media. Track public concerns across thousands of survey open-ends. Map stakeholder positions from reports.

McKinsey, BCG, think tanks, ministries, GIZ

Journalism & NGOs

Analyze leaked document dumps. Detect coordinated disinformation. Quantify bias in news coverage. Scrape and classify parliamentary records.

Reuters, SZ, Correctiv, Amnesty, Bellingcat

Industry & tech

Mine customer reviews for product signals. Monitor brand reputation at scale. Build classifiers for content moderation. Prototype with LLMs.

Market research, UX, trust & safety, data science roles

The bottleneck is rarely data — it's people who can turn messy communication into evidence.

Course structure

How the semester is organized The course is split into two blocks: **Lectures 1–7** on computational text analysis, followed by **Lectures 8–14** on computational visual, video, and audio analysis. Materials stored at: tamaragrechanaya.com

Block I: Text analysis
Lectures: 1–7
Instructor: Tamara Grechanaya
Email: Tamara.Grechanaya@lmu.de

Block II: Visual / video / audio
Lectures: 8–14
Instructor: Clara Fochler
Email: clara.fochler@gsi.lmu.de

	L1	L2	L3	L4	L5	L6	L7
Text	14 Apr	21 Apr	28 Apr	5 May	12 May	19 May	26 May

	L8	L9	L10	L11	L12	L13	L14
Multi	2 Jun	9 Jun	16 Jun	23 Jun	30 Jun	7 Jul	14 Jul

First half: text analysis — Second half: multimodal analysis

Course Roadmap

1. **Text as Data: Foundations & Preprocessing** ← *Today*
2. Classical Text Classification (Logistic Regression, Naive Bayes)
3. Word Embeddings & Vector Spaces
4. Document Representations & Topic Models
5. Neural Networks & Sequence Models
6. Attention & the Transformer Architecture
7. Transfer Learning & Fine-Tuning BERT

Arc: from counting words $\longrightarrow$ understanding language. Each lecture builds on the previous one. By Lecture 7, you will fine-tune a state-of-the-art language model.

Prerequisites & practical setup

You do **not** need to be an expert, but you should have a **basic working familiarity** with statistics and coding.

Statistics

Basic statistical knowledge is expected: probabilities, regression, and core concepts.

We will inevitably use statistics, but the focus is on understanding the main ideas rather than formal derivations.

Programming

Basic knowledge of **R** and **Python** is expected.

You do not need to be a strong programmer, but you should be comfortable reading and modifying simple code.

Tools we will use

R (Lectures 1–4), **Python** (Lectures 5–7)

Most practical work can be done in **Google Colab**; the free version should be sufficient.

Goal: conceptual understanding + basic hands-on implementation

Course materials

Core materials

1. Main course book: *Text as Data: A New Framework for Machine Learning and the Social Sciences.* J. Grimmer, M.E. Roberts, and B.M. Stewart.

2. Practical reference: *Speech and Language Processing* (3rd ed. draft). D. Jurafsky and J.H. Martin. Open access

3. Academic articles: selected readings for each lecture; either open access or accessible with LMU credentials

Code

R and Python code examples will be supplied by me and made available on the course website.

Data

Datasets used in practical sessions will be open source.

Structure of each session

Each class combines **conceptual explanation** with a short **hands-on practical component**, followed by an optional exercise for further practice.

1. Theory 60 min

Main concepts, methods, and real research examples from political science.

Understand why the method exists and when to use it.

2. Practice 30 min

Guided walk-through in **R** or **Python** on a real political dataset.

See the method run end-to-end, line by line.

3. At home (optional)

A short home assignment extending the in-class example to a new question.

Consolidate by doing it yourself.

Learn the idea → see it in code → try it yourself

Final evaluation

You can choose between the following:

Option A: Term paper

A research paper applying methods from the course to an empirical question.

Submission: Paper + code

Option B: In-class presentation

A presentation of your own research project using methods from the course.

Submission: Presentation + code

Important. In both options, the **use of course methods** is compulsory, and the **code must be submitted** together with your final work. Communicate your choice by April 28th (3rd lecture).

Same standard, different format: written paper or oral presentation

Today's Outline

Part I: Motivation

- Why analyze text computationally?
- What political texts exist?
- Foundational principles (Grimmer & Stewart)

Part II: Preprocessing

- Tokenization
- Normalization & lowercasing
- Stop word removal
- Stemming vs. lemmatization

Part III: Numerical Representation

- Bag of words
- Document-term matrix (DTM)
- TF-IDF weighting
- Sparsity and its implications

Part IV: Practical Session

- Preprocessing U.K. Commons speeches in R
- Building a DTM with `quanteda`
- Which words distinguish parties?

Table of Contents

1. Part I: Why Text Analysis?
2. Part II: The Preprocessing Pipeline
3. Part III: Numerical Representation
4. Part IV: Tools & Practical Session
5. Wrap-up

Why Analyze Political Text Computationally?

Politics is conducted through language: speeches, legislation, manifestos, media coverage, and public discourse.

The problem with manual coding:

- The Manifesto Project: trained coders spent **decades** hand-coding party platforms
- Inter-coder reliability is often $\kappa < 0.7$
- Cannot scale to millions of social media posts, news articles, or parliamentary speeches
- Expensive: researcher time is the bottleneck

The promise of computational methods:

- **Scale**: process millions of documents
- **Consistency**: same algorithm, same result every time
- **Discovery**: find patterns invisible to close reading
- **Replicability**: fully documented, reproducible pipeline
- **Speed**: analyze a legislative period in minutes

What Political Texts Do We Analyze?

Parliamentary Speeches

Bundestag, EU Parliament, Congress

Party Manifestos

Electoral programs, policy platforms

News Media

Articles, editorials, headlines, TV

Social Media

Twitter/X, Facebook, Reddit, forums

Key Distinction

Each text type has different **structure** (formal vs. informal), **length** (tweets vs. manifestos), **language** (legal vs. colloquial), and **metadata** (speaker, date, party). Your preprocessing must adapt accordingly.

Key Research Applications

Measurement & Scaling

- Estimating party positions on policy dimensions (Laver, Benoit & Garry 2003)
- Measuring legislative polarization over time
- Scaling legislators on ideological dimensions from speeches

Classification

- Sentiment analysis of political communication
- Detecting hate speech or misinformation
- Coding policy topics in manifesto sentences

Discovery & Exploration

- Topic models: what issues dominate the agenda?
- Detecting emerging political frames
- Tracing how political language changes over time

Comparison

- Cross-country comparisons of parliamentary discourse
- Media framing across outlets
- Government vs. opposition language

Four Principles of Quantitative Text Analysis

Grimmer & Stewart (2013), "Text as Data: The Promise and Pitfalls of Automatic Content Analysis Methods for Political Texts," *Political Analysis* 21(3).

1. **All quantitative models of language are wrong — but some are useful.**
   No model captures all nuance of human language. The goal is *useful approximation*, not perfect representation. A bag-of-words model "knows" nothing about grammar — yet it can distinguish party platforms remarkably well.

2. **Quantitative methods for text augment humans, not replace them.**
   Automated tools help you read more, not stop reading. You still need substantive knowledge to design the analysis and interpret results.

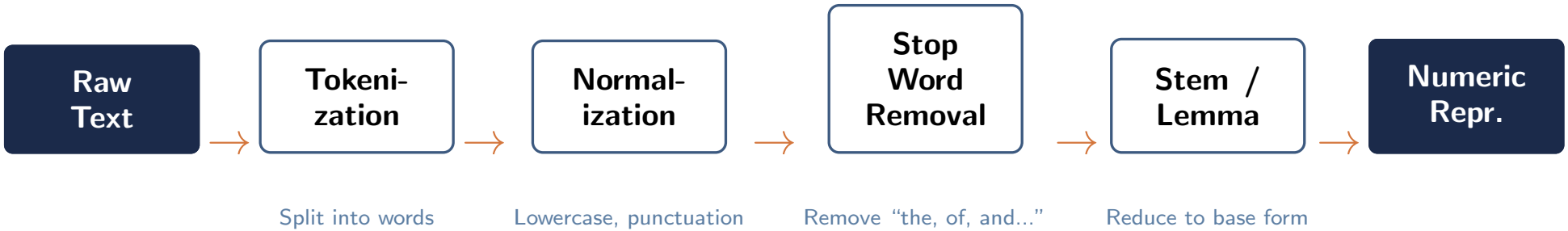
3. **There is no universally best method for text analysis.**
   The right method depends on your research question, your corpus, and your theory. This course gives you a toolkit; you choose the right tool.

4. **Validate, validate, validate.**
   Always check that computational outputs match human judgment. Compare against hand-coded samples. Report validation metrics.

Table of Contents

1. Part I: Why Text Analysis?
2. Part II: The Preprocessing Pipeline
3. Part III: Numerical Representation
4. Part IV: Tools & Practical Session
5. Wrap-up

The Big Picture: From Raw Text to Numbers



Caution

Every preprocessing step is a **modeling choice** that affects your results. There is no "neutral" default pipeline. You must **document** and **justify** every decision — and ideally test whether your results are robust to alternative choices (**sensitivity analysis**).

Step 1: Tokenization

Definition: Splitting a text string into individual units (*tokens*) — usually words.

Illustrative political statement

Input:

`‘‘The Prime Minister doesn’t think Article 50 should be reopened by the European Union in 2024.’’`

Output (word tokens):

`[‘‘The’’, ‘‘Prime’’, ‘‘Minister’’, ‘‘does’’, ‘‘n’t’’, ‘‘think’’, ‘‘Article’’, ‘‘50’’, ‘‘should’’, ‘‘be’’, ‘‘reopened’’, ‘‘by’’, ‘‘the’’, ‘‘European’’, ‘‘Union’’, ‘‘in’’, ‘‘2024’’, ‘‘.’’]`

Design decisions you must make:

- **Punctuation**: Keep the final `.` or drop it? For bag-of-words it is often noise, but for sentence splitting it matters.
- **Contractions**: `doesn’t` → split into `does` + `n’t` (spaCy default), keep it whole, or expand it to `does not`? **Warning**: if you split and then remove stop words, you may accidentally lose the negation.
- **Multi-word expressions**: Should `Prime Minister` and `European Union` be one token or two? We will return to this when we discuss *n-grams*.
- **Numbers & references**: Is `Article 50` just a number plus a word, or a meaningful political reference that should be preserved?

Step 2: Lowercasing & Normalization

Goal: Reduce superficial variation so that equivalent forms map to the same token.

Lowercasing

- "Brexit" → "brexit", "Parliament" → "parliament"
- Merges capitalized sentence-initial words with their lowercase counterparts
- But: destroys distinctions for acronyms (NHS, EU, UN, MP, GOP) and collapses proper nouns ("Bush" the president vs. "bush" the plant)

Removing punctuation

- Commas, periods, quotation marks stripped
- But: loses sentence boundaries

Removing numbers

- "2016", "Section 230", "$500" removed
- But: years, legal references, and monetary amounts can be substantively meaningful (e.g., 1973, 2008, 2016)

Removing URLs & handles

- Essential for social media data
- "@10DowningStreet", "https://..." → removed
- For parliamentary text: rarely needed

Unicode normalization

- Handles accents, smart quotes, special characters
- Important for multilingual corpora

Step 3: Stop Word Removal

Stop words: High-frequency, low-information words (articles, prepositions, auxiliaries).

Common English stop words:

`the, of, and, a, to, in, is, it, that, for, as, was, on, are, with, be, at, by, not, this, have, from, or, one, had, but, what, some, we, can, out, other, were, all ...`

Standard lists: `stopwords("en")` in quanteda contains ∼175 words. The Snowball list, NLTK list, and spaCy list differ.

Before and After

Before:
The Prime Minister doesn't think Article 50 should be reopened by the European Union in 2024

After:
Prime Minister think Article 50 reopened European Union 2024

Caution

"not" is on most English stop word lists. Removing it flips sentiment: "not acceptable" → "acceptable". Same for "no", "nor", "against". Consider a **custom stop word list** for your task. In sentiment analysis or stance detection, keep negation words!

Step 4: Stemming vs. Lemmatization

Goal: Map inflected word forms to a common base, reducing vocabulary size.

Stemming (rule-based)

Chop off suffixes using heuristic rules (e.g., Porter / Snowball Stemmer).

govern	→	govern
governs	→	govern
governed	→	govern
governing	→	govern
government	→	govern
political	→	polit
universal	→	univers
university	→	univers (!)

✓ Fast, no dictionary needed
✗ Produces non-words ("polit")
✗ **Over-stems**: unrelated words merged ("universal" / "university")
✗ **Under-stems**: related words split ("ran" / "run")

Lemmatization (dictionary-based)

Look up the word in a morphological dictionary to find its canonical form (lemma).

govern	→	govern
governs	→	govern
governed	→	govern
governing	→	govern
government	→	government
political	→	political
universal	→	universal
university	→	university ✓

✓ Produces real words
✓ Linguistically correct
✓ Preserves meaningful distinctions
✗ Slower, needs a language model

Recommendation for English

For serious research, lemmatization is generally preferred. Use `spacyr` (R wrapper for spaCy) or `udpipe` — both have excellent pre-trained English models. For quick exploratory work, the Porter stemmer via `tokens_wordstem()` in `quanteda` is acceptable. For topic models, stemming often works fine.

N-grams: Capturing Multi-Word Expressions

Problem: Bag-of-words treats each word independently, losing meaningful phrases.

Definitions:

- **Unigram**: single word
  "climate", "change"
- **Bigram**: two consecutive words
  "climate change", "social justice"
- **Trigram**: three words
  "European Central Bank"
- **N-gram**: n consecutive words

Politically important bigrams:

- "climate change" / "climate emergency"
- "minimum wage"
- "Prime Minister"
- "European Union"
- "working class"
- "border control"

Unigrams vs. Bigrams

Sentence: "The European Central Bank raised interest rates"

Unigrams:
`[european, central, bank, raised, interest, rates]`

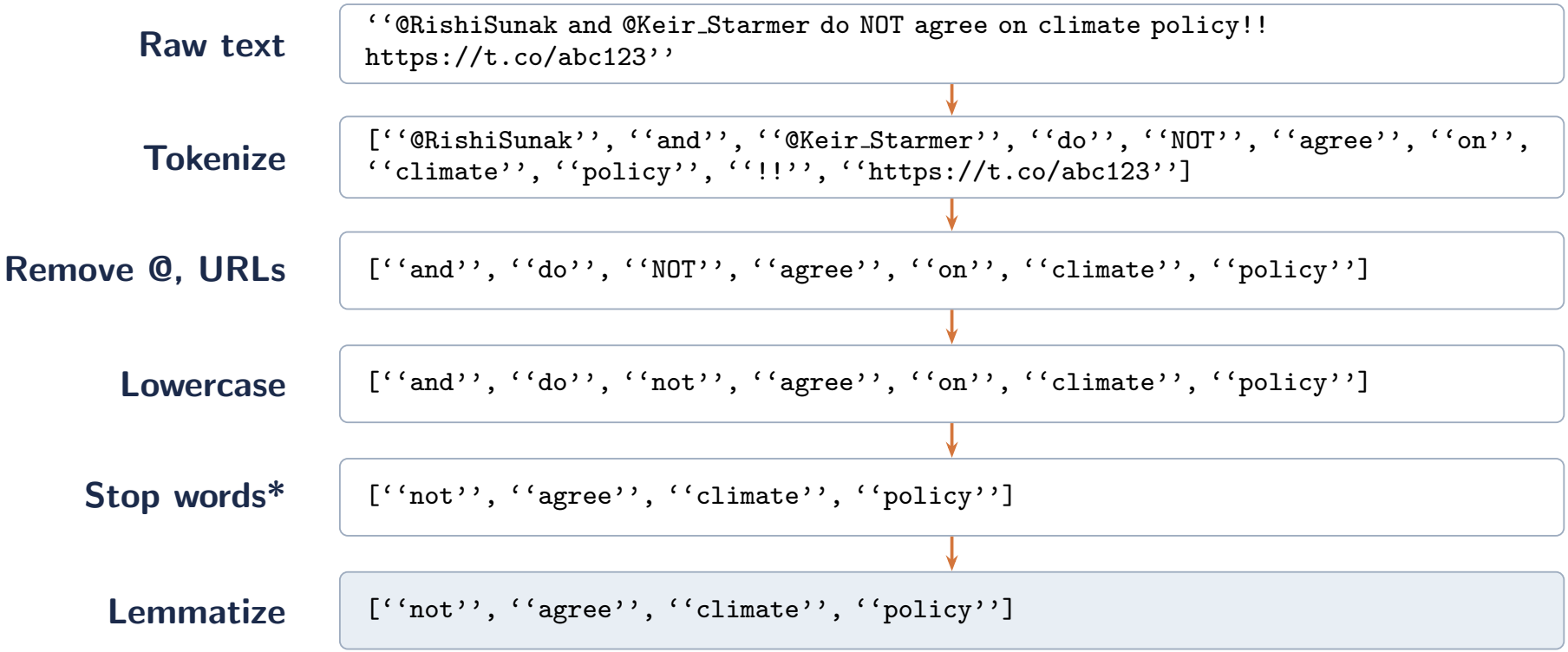
Bigrams:
`[european_central, central_bank, bank_raised, raised_interest, interest_rates]`

$\rightarrow$ "interest rates" and "central bank" are captured as meaningful units.

Caution

Adding bigrams **dramatically increases vocabulary size**. If you have $|V| = 50{,}000$ unigrams, you could have up to $|V|^2 = 2.5$ billion bigrams. In practice, keep only frequent or theoretically motivated n-grams.

The Complete Preprocessing Example



*Using a custom stop word list that preserves "not". With the default `stopwords("en")`, "not" would also be removed — silently flipping the sentiment.

Table of Contents

1. Part I: Why Text Analysis?
2. Part II: The Preprocessing Pipeline
3. Part III: Numerical Representation
4. Part IV: Tools & Practical Session
5. Wrap-up

From Words to Numbers: The Core Challenge

The Fundamental Problem

Machine learning algorithms require **numerical input**. We need a systematic way to convert a collection of texts into a **matrix of numbers** while preserving the information relevant to our research question.

Three representations we'll cover today (simplest to more sophisticated):

1. **Bag of Words (BoW)**: Count how often each word appears in each document.
2. **Document-Term Matrix (DTM)**: Stack all BoW vectors into a matrix.
3. **TF-IDF**: Weight the DTM to emphasize distinctive words.

What we'll cover in later lectures (preview):

- *Lecture 3*: Dense word embeddings (Word2Vec, GloVe) — 300-dimensional real-valued vectors instead of sparse counts
- *Lecture 7*: Contextual embeddings (BERT) — where the same word gets a different vector depending on its context

Bag of Words (BoW)

Idea: Represent each document as the **count of each word**, ignoring word order entirely.

Example

Document (after preprocessing):
"climate action government tackle climate emergency"

Bag-of-Words representation:

	climate	action	government	tackle	emergency
Count	2	1	1	1	1

Strengths:

- Simple, fast, interpretable
- Works surprisingly well for classification and scaling
- Foundation for more complex methods

Key limitation:

- "The dog bit the man" = "The man bit the dog"
- All word order and syntax is lost
- Partially addressed by n-grams

The Document-Term Matrix (DTM)

Stack all documents' BoW vectors into a single matrix: rows = documents, columns = vocabulary.

	climate	government	economy	nhs	brexit	⋯	*row sum*
Speech 1 (Green)	**5**	2	1	0	0	⋯	47
Speech 2 (Cons.)	0	1	2	1	**7**	⋯	52
Speech 3 (Labour)	2	3	2	**4**	1	⋯	61
⋮						⋱	
Speech m	1	0	3	2	0	⋯	38

Formally: Given m documents and a vocabulary V of size $|V|$, the DTM is:

$$\mathbf{X} \in \mathbb{N}^{m \times |V|}, \qquad X_{ij} = \text{count of word } j \text{ in document } i$$

The Sparsity Problem

A typical political corpus might have $|V| = 50{,}000+$ unique words. A single speech uses perhaps 200 distinct words. That means $> 99.5\%$ of cells in the DTM are zero. This **sparse, high-dimensional** matrix creates computational and statistical challenges.

Dealing with Sparsity: Practical Strategies

Several techniques to reduce the size of the DTM:

1. **Remove very rare words** (e.g., appearing in < 5 documents)
   Rationale: words that appear only once or twice are often typos, proper nouns, or too rare to generalize from. In `quanteda`: `dfm_trim(min_docfreq = 5)`
2. **Remove very frequent words** (e.g., appearing in $> 95\%$ of documents)
   Rationale: words that appear in almost every document cannot distinguish between documents. Stop word removal already handles many of these.
3. **Use TF-IDF weighting** (next slide)
   Automatically down-weights words that appear everywhere. Doesn't reduce matrix size, but makes the values more informative.
4. **Feature selection**: Keep only the top k most informative features
   E.g., select 5,000 words with highest variance, or highest χ^2 association with the outcome variable.

Rule of thumb: For most political science tasks, trimming to ~5,000–15,000 features works well. Always check what you're removing!

TF-IDF: Weighting Words by Distinctiveness

Intuition: Not all words are equally informative. Words that are frequent in a document *but rare across the corpus* are the most distinctive.

Term Frequency (TF)

How often does word t appear in document d?

$$\text{TF}(t,d) = \frac{f_{t,d}}{|d|}$$

where $f_{t,d}$ is the raw count and $|d|$ is the document length.
Also common: raw count, or $\log(1+f_{t,d})$

Inverse Document Frequency (IDF)

How rare is word t across all N documents?

$$\text{IDF}(t) = \log \frac{N}{\text{df}(t)}$$

where $\text{df}(t)$ = number of documents containing t.
Rare word $\Rightarrow$ high IDF.
Common word $\Rightarrow$ low IDF.

$$\text{TF-IDF}(t,d) = \text{TF}(t,d) \times \text{IDF}(t)$$

Result: words frequent in a document but rare overall get the **highest weight**. Words like "die" and "und" that appear everywhere get pushed toward zero — even if you didn't remove them as stop words.

TF-IDF: A Worked Example

Corpus: 1,000 U.K. House of Commons speeches. Consider the word "*Brexit*" in a speech by a Conservative MP.

Step 1: Term Frequency

Speech has 500 words. "Brexit" appears 8 times.

$$\text{TF} = \frac{8}{500} = 0.016$$

Step 2: Inverse Document Frequency

"Brexit" appears in 50 of 1,000 speeches.

$$\text{IDF} = \log \frac{1000}{50} = \log(20) = 1.301$$

Step 3: TF-IDF

$$\text{TF-IDF} = 0.016 \times 1.301 = 0.0208$$

Compare with "the":

"the" appears 25 times in this speech (TF = 0.05), but in *all* 1,000 speeches:

$$\text{IDF}_{\text{the}} = \log \frac{1000}{1000} = 0$$

$$\text{TF-IDF}_{\text{the}} = 0.05 \times 0 = 0$$

$\rightarrow$ "the" gets **zero weight**, even though it's the most frequent word!

Key Concept

TF-IDF is a **heuristic**, not a probabilistic model. It has no parameters to tune and requires no training. It simply re-weights the raw DTM. Many downstream methods (classification, clustering, similarity) benefit from TF-IDF preprocessing.

What TF-IDF Reveals: Party-Distinctive Vocabulary

Illustrative example: Highest TF-IDF terms by party in U.K. House of Commons speeches.

Conservative

- free enterprise
- hardworking families
- border control
- lower taxes
- law and order
- national interest
- traditional values
- British values

Labour

- working people
- NHS
- public services
- inequality
- austerity
- fair wages
- social justice
- public ownership

Green Party

- climate emergency
- renewable energy
- net zero
- biodiversity
- fossil fuels
- sustainable
- carbon
- pollution

Takeaway: Even this simple method — counting words and weighting by distinctiveness — reveals the core substantive priorities of each party. No reading required.
But: *you need substantive knowledge to interpret what you see.* (Principle #2!)

Preprocessing Choices: There Is No Default Pipeline

Decision	Argument for	Argument against
Remove stop words?	Reduces noise; smaller matrices; faster computation	"not" carries meaning; negation lost; "against" and "oppose" matter
Stem / lemmatize?	Groups word variants together	Overstemming collapses unrelated words ("universal"/"university")
Lowercase?	Merges "Parliament" with "parliament" at sentence start	Destroys acronyms (NHS, EU, UN, MP), proper nouns
Remove rare words?	Massive dimensionality reduction; removes noise	Rare words ("furlough" in 2020) may be the most informative
Use n-grams?	Captures "climate change" as a unit	Vocabulary explosion ($\|V\|^2$ potential bigrams)
Min. document freq.?	Removes hapax legomena	If only one MP says it, that may be the finding

Best Practice

Run your analysis with **multiple preprocessing configurations** and report whether the substantive conclusions change. If they do, that's informative. If they don't, your findings are robust.

Table of Contents

1. Part I: Why Text Analysis?
2. Part II: The Preprocessing Pipeline
3. Part III: Numerical Representation
4. Part IV: Tools & Practical Session
5. Wrap-up

R Toolkit for Text Analysis

Core packages (Lectures 1–4):

`quanteda` — the workhorse
Creates corpora, tokens, DFMs. TF-IDF, n-grams, grouping by metadata. Think of it as the `tidyverse` of text analysis.

`quanteda.textstats`
Frequency tables, keyness, readability, lexical diversity.

`quanteda.textplots`
Word clouds, comparison plots, network plots.

`tidytext`
Tidy approach to text mining. Integrates with `dplyr` and `ggplot2`. Good for exploration and visualization.

Supporting packages:

`readtext`
Import from PDF, DOCX, CSV, JSON, URLs.

`spacyr`
R interface to spaCy. Lemmatization, NER, POS-tagging. Excellent pre-trained English models.

`udpipe`
Alternative to spaCy. Pure R, no Python dependency. Good English models; supports 60+ languages.

Later (Lectures 5–7):

We will switch to **Python** with `PyTorch` and `Hugging Face Transformers` for neural models and BERT fine-tuning. Python is the standard for deep learning in NLP.

Practical Session: Overview

Exploring U.K. House of Commons Speeches in R with `quanteda`

What we'll do in the next 30 minutes:

1. **Load** a corpus of U.K. parliamentary speeches (`readtext` + `quanteda`)
2. **Preprocess**: tokenize, remove stop words, lowercase
3. **Build** a document-feature matrix (DFM)
4. **Compute TF-IDF** and explore party-distinctive words
5. **Visualize** top features per party with `ggplot2`

Data: A pre-prepared sample from **ParlSpeech V2** (Rauh & Schwalbach, Harvard Dataverse) — U.K. House of Commons speeches with columns: `text`, `speaker`, `party`, `date`, `agenda`.

Setup — install if needed:

```
install.packages(c("quanteda", "quanteda.textstats",
  "quanteda.textplots", "readtext", "tidyverse"))
```

Practical: Loading & Creating a Corpus

```r
library(quanteda)
library(quanteda.textstats)
library(quanteda.textplots)
library(tidyverse)

# Load the data
speeches <- read.csv("uk_commons.csv", stringsAsFactors = FALSE)

# Quick inspection
glimpse(speeches)
table(speeches$party)

# Create a quanteda corpus
corp <- corpus(speeches, text_field = "text")

# Inspect: what does a corpus look like?
summary(corp, n = 5)
cat(as.character(corp[1]))  # print the first document
```

A corpus in quanteda stores texts alongside metadata ("docvars"). The metadata travels with the text through every step.

Practical: Tokenization & Preprocessing

```r
# Tokenize: split into words
toks <- tokens(corp,
               remove_punct = TRUE,     # remove punctuation
               remove_numbers = TRUE,   # remove numbers
               remove_url = TRUE)       # remove URLs

# Remove English stop words
toks <- tokens_remove(toks, stopwords("en"))

# Lowercase
toks <- tokens_tolower(toks)

# Optional: add bigrams
toks_bi <- tokens_ngrams(toks, n = 2)

# Inspect a processed document
head(toks[[1]], 30)
```

Caution

What did we lose by removing stop words? What if "not" appeared in an important context? What did lowercasing do to acronyms like NHS or EU? These are not hypothetical problems — they're real decisions researchers face.

Practical: Building & Exploring the DFM

```r
# Build the Document-Feature Matrix
dfm_speeches <- dfm(toks)

# Basic stats
cat("Documents:", ndoc(dfm_speeches), "\n")
cat("Features:", nfeat(dfm_speeches), "\n")
cat("Sparsity:", round(sparsity(dfm_speeches) * 100, 1), "%\n")

# Top 20 most frequent words
topfeatures(dfm_speeches, 20)

# Trim rare words (appear in < 10 documents)
dfm_trimmed <- dfm_trim(dfm_speeches, min_docfreq = 10)
cat("Features after trimming:", nfeat(dfm_trimmed), "\n")

# Group by party and look at top words
dfm_party <- dfm_group(dfm_speeches, groups = party)
topfeatures(dfm_party["Conservative", ], 10)
topfeatures(dfm_party["Labour", ], 10)
topfeatures(dfm_party["Green", ], 10)
```

Practical: TF-IDF & Party Comparison

```r
# Compute TF-IDF
dfm_tfidf <- dfm_tfidf(dfm_speeches)

# Which words are most distinctive for each party?
# Use "keyness" -- chi-squared test vs. rest of corpus
dfm_lab <- dfm_subset(dfm_speeches, party == "Labour")
dfm_other <- dfm_subset(dfm_speeches, party != "Labour")

# textstat_keyness: statistical test of over/under-use
keyness_lab <- textstat_keyness(
  rbind(dfm_group(dfm_lab), dfm_group(dfm_other)),
  target = 1L
)

# Visualize
textplot_keyness(keyness_lab, n = 15,
  color = c("#E4003B", "gray60")) +
  ggtitle("Words distinctive of Labour vs. all other parties")
```

Discussion point: Keyness uses a χ^2 test. High positive = over-represented in target group. High negative = under-represented. Do the results make substantive sense?

Practical: Visualization

```
# Word frequency comparison across parties
freq_by_party <- textstat_frequency(dfm_speeches,
                                    n = 10,
                                    groups = party)

ggplot(freq_by_party, aes(x = reorder_within(feature, frequency, group),
                          y = frequency, fill = group)) +
  geom_col(show.legend = FALSE) +
  coord_flip() +
  facet_wrap(~group, scales = "free_y") +
  scale_x_reordered() +
  scale_fill_manual(values = c(
    "Conservative" = "#0087DC",
    "Labour"       = "#E4003B",
    "LibDem"       = "#FAA61A",
    "SNP"          = "#FDF38E",
    "Green"        = "#6AB023",
    "Reform"       = "#12B6CF")) +
  labs(x = NULL, y = "Frequency",
       title = "Top 10 Words by Party (UK House of Commons)")
```

We can see party priorities directly from word counts. If simple word counting reveals this much, imagine what deeper methods can do.

Table of Contents

1. Part I: Why Text Analysis?
2. Part II: The Preprocessing Pipeline
3. Part III: Numerical Representation
4. Part IV: Tools & Practical Session
5. Wrap-up

Key Takeaways

1. **Text is the most abundant source of political data** — and it requires systematic transformation before quantitative analysis.
2. **Preprocessing is not neutral.** Every step (stop words, stemming, lowercasing) encodes assumptions about what information matters. Document and justify your choices.
3. **The Document-Term Matrix** is the bridge from text to quantitative analysis. It's sparse, high-dimensional, and the foundation for most methods we'll cover.
4. **TF-IDF** improves on raw counts by automatically weighting words by how distinctive they are. Common words get down-weighted; rare, document-specific words get up-weighted.
5. **Validation matters.** Always check your preprocessing by inspecting what you kept and what you removed. Always verify that computational outputs match your substantive understanding.

Home Assignment

Task: Apply the preprocessing pipeline to a political text corpus of your choice.

Instructions:

1. Choose a corpus: party manifestos (Manifesto Project), EU Parliament speeches, newspaper editorials, or your own collection.
2. Preprocess with explicit justification for each choice: tokenization, stop words (which list?), stemming/lemmatization, n-grams, minimum frequency thresholds.
3. Build a DFM. Report: number of documents, vocabulary size, sparsity.
4. Compute TF-IDF. Identify the 10 most distinctive words per group (party, country, time period — whatever grouping is relevant).
5. **Sensitivity check**: re-run with one different preprocessing choice (e.g., with vs. without stop words, or stem vs. lemmatize). Do the top-10 distinctive words change?
6. Write a **1-page interpretation** of what the vocabulary differences reveal substantively.

Submit: R script + 1-page write-up. Due before Lecture 2.

Readings

Required:

- 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.

Recommended:

- Benoit, K. (2020). "Text as Data: An Overview." In *The SAGE Handbook of Research Methods in Political Science and International Relations*.
- Welbers, K., Van Atteveldt, W., & Benoit, K. (2017). "Text Analysis in R." *Communication Methods and Measures*, 11(4), 245–265.
- `quanteda` tutorials: `https://tutorials.quanteda.io`

Next week: Classical Text Classification

Logistic Regression & Naive Bayes for political text