Quantitative Text Analysis and Natural Language Processing using Python

Day 2

Joshua Cova and Luuk Schmitz

2026-01-23

Overview

Part 1: The building blocks of NLP

Part 2: Classification metrics

Coding Exercise 1: Building a DTM

Part 3: Frequency analysis in practice

Part 4: The limits of counting

Coding Exercise 2: Document similarity

Part 5: Setting the scene for word embeddings & LLMs

Part 1: The building blocks of NLP: From text to matrix

The fundamental challenge

We want to use statistical techniques on text to measure the frequency and valence of concepts.

But text is:

- Variable length
- Unstructured
- Symbolic (words, not numbers)

We need a **representation** that converts text to numbers.

Converting text into units

```
Raw text
    → Preprocessing (tokenization, lemmatization, stopwords)
        → Representation (DTM, TF-IDF)
            → Analysis (frequency, similarity, classification)
```

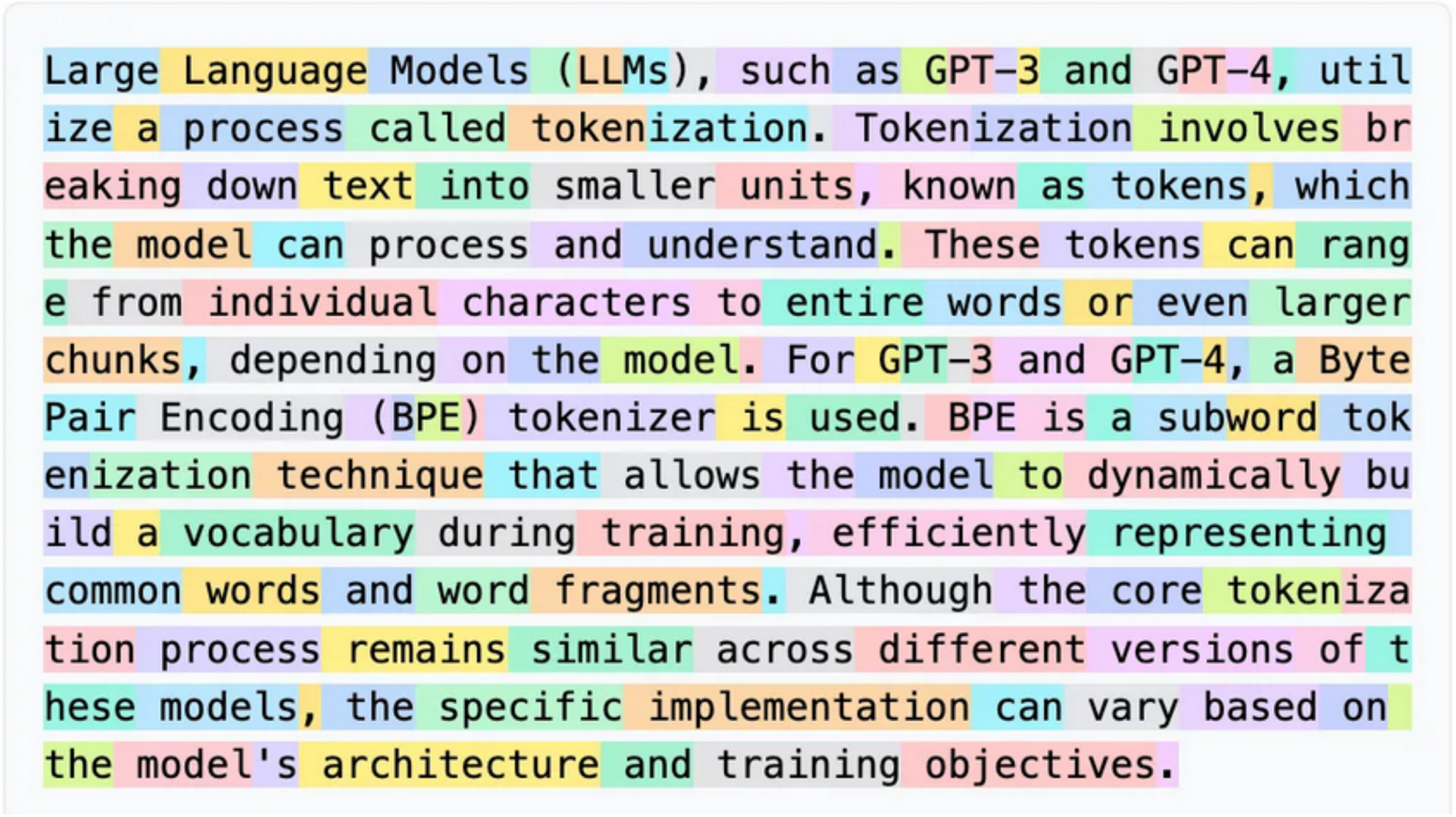
Computers work by processing texts into numbers.

The first step therefore has to be to convert raw text into a format that computers can work with

$\rightarrow$ **Tokenization**

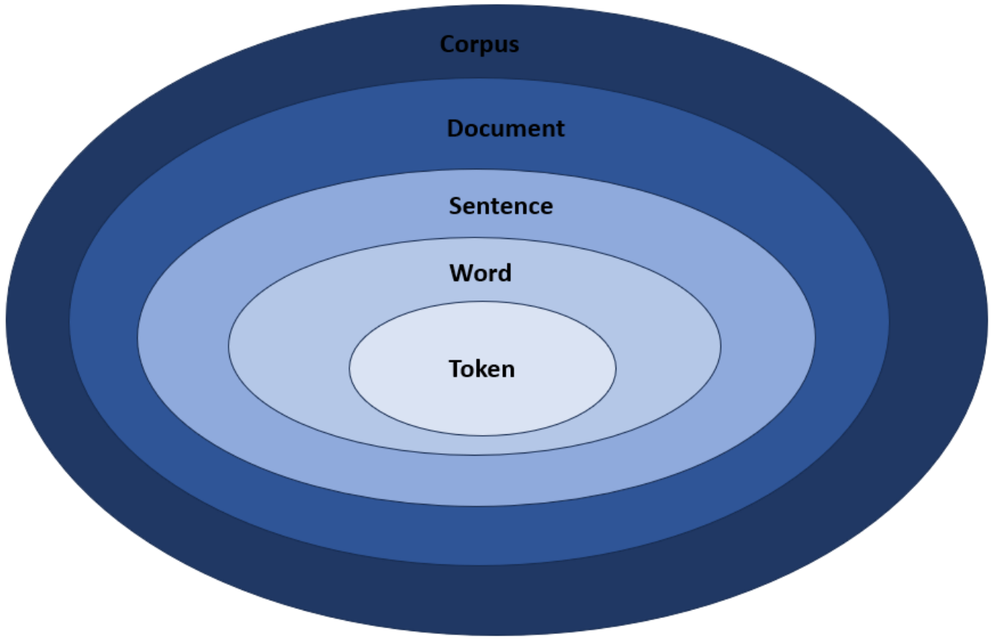
Structured representation of texts, break the text up into smaller units that a processor can compute

Tokenization example



Tokenization example. Source: Medium

Different types of units



Corpus: All electoral manifestos in Germany in the 2025 federal election

Document: The electoral manifesto of the SPD in the 2025 German federal election

Sentence: “Der Mangel an Fachkräften darf nicht zur Achillesferse unseres Bildungssystems werden.”

Word: “Mangel”

Token: “m”; “angel”

Pre-processing

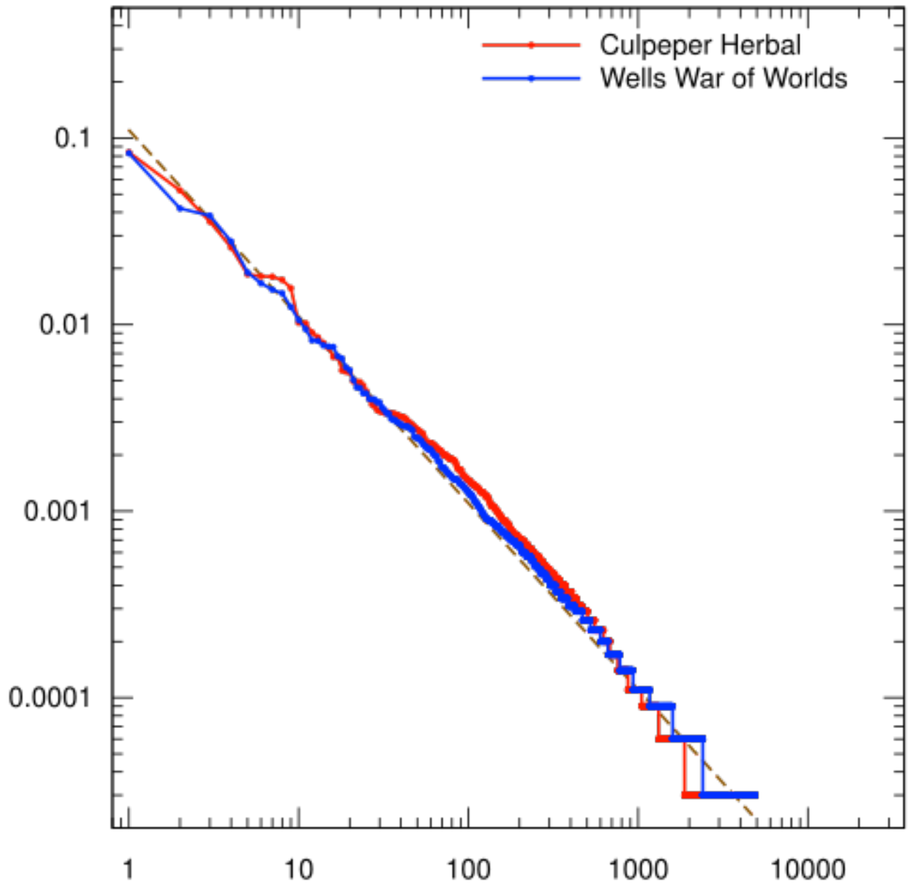
Common steps:

- Lowercasing (“Democracy” vs. “democracy”)
- Removing punctuation (“election!” vs. “election”)
- Removing numbers
- Removing non-words (signs, html tags)

Goal:

- Reduce noise and keep meaningful variation
- **But,** as always,choices should depend on context and research question !

Zipf’s Law



Zipf’s law: Empirical law stating that the frequency of each word is inversely proportional to its rank. Source: Wikipedia

Stop words

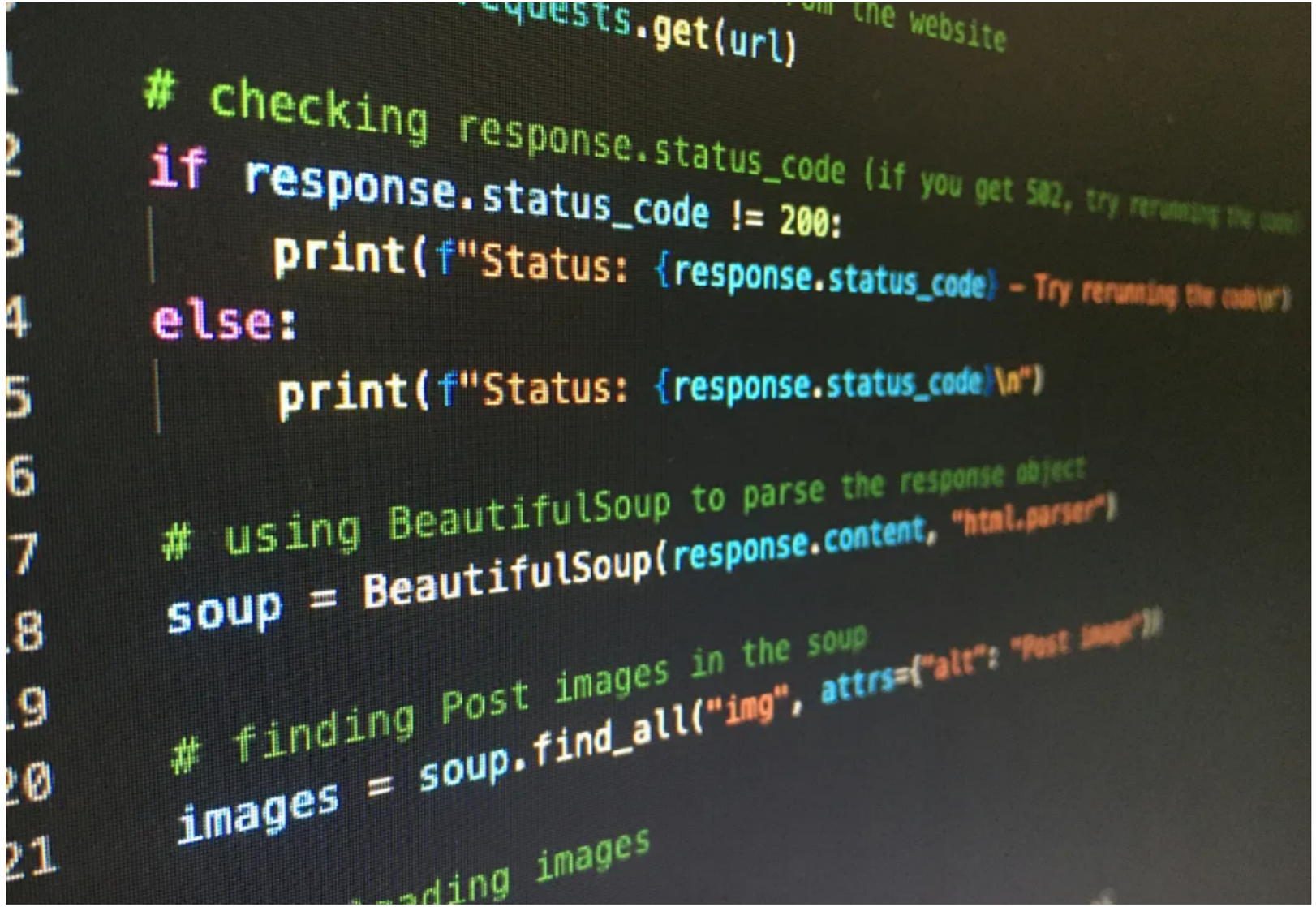
- Stop words: Very common words with little substantive meaning (e.g. “the”, “of”)
- Dominate frequency counts
- Several stop words list are available in different Python libraries, but it is advisable to check whether these stop words also make sense for your specific application

Stemming & Lemmatization

Reducing words to their base form

- **Stemming**:
  - Fast
  - Crude
  - “running” → “runni”
- **Lemmatization**:
  - Slower
  - Linguistically informed
  - Better for social science
- Examples:
  - “runs”, “running” $\rightarrow$ “run”
  - “better” $\rightarrow$ “good”

And now let’s see how all of this works in practice…



Part 2: Classification metrics

Connecting to yesterday

Yesterday we discussed **reliability**—do coders agree?

Today we formalize this with **classification metrics**:

- How do we measure agreement between human and machine?
- How do we measure classifier performance?
- What trade-offs do different metrics capture?

The confusion matrix

	Predicted Positive	**Predicted Negative**
Actually Positive	True Positive (TP)	False Negative (FN)
Actually Negative	False Positive (FP)	True Negative (TN)

Like so much in the social sciences, everything flows from a 2×2 table.

Accuracy: The obvious metric

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

The proportion of all predictions that are correct.

Problem: Can be misleading with imbalanced classes.

Problem 2: Only works as a model-level statistic.

The accuracy trap: An example

Detecting “crisis” language in 1,000 financial reports:

- 50 reports contain crisis language (5%)
- 950 reports don’t (95%)

Classifier A: Always predicts “no crisis” → 95% accuracy!

Classifier B: Correctly identifies 40/50 crises, 10 false alarms → 98% accuracy

Both look good. Only one is useful.

Precision: When false positives are costly

$$\text{Precision} = \frac{TP}{TP + FP}$$

Of all the items we labeled positive, what fraction are actually positive?

High precision = few false alarms

Recall: When false negatives are costly

$$\text{Recall} = \frac{TP}{TP + FN}$$

Of all the items that are actually positive, what fraction did we find?

High recall = few missed cases

Also called **sensitivity** or **true positive rate**.

The precision-recall tradeoff

You usually can’t maximize both.

Aggressive classifier: Predicts “positive” liberally

- ↑ Recall (catches more positives)
- ↓ Precision (more false alarms)

Conservative classifier: Predicts “positive” cautiously

- ↓ Recall (misses some positives)
- ↑ Precision (fewer false alarms)

F1 score: Balancing the tradeoff

$$F_1 = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

The harmonic mean of precision and recall.

- Balances both concerns
- Punishes extreme imbalance
- Widely used as a single summary metric

Worked example

	Predicted Crisis	Predicted Normal
Actually Crisis	40	10
Actually Normal	20	930

Calculate:

- Precision = ?
- Recall = ?
- F1 = ?

Let’s run the numbers!

Worked example: Solutions

- **Precision** = 40 / (40 + 20) = **0.67**
  - Two-thirds of predicted crises are real
- **Recall** = 40 / (40 + 10) = **0.80**
  - We catch 80% of actual crises
- **F1** = 2 × (0.67 × 0.80) / (0.67 + 0.80) = **0.73**

Is this good? It depends on your research question and your field of study (the stakes in the social sciences ≠ clinical trial or astrophysics).

Beyond binary: Multi-class metrics

For multiple categories, calculate per-class then average:

Macro-average: Simple mean across classes

- Treats rare classes equally

Micro-average: Weighted by class frequency

- Treats all instances equally

Weighted average: Weighted by class size

- Common in imbalanced settings

Connecting to inter-coder reliability

Cohen's Kappa for classification:

$$\kappa = \frac{p_o - p_e}{1 - p_e}$$

Same logic as for human coders:

- p_o = proportion where classifier and human agree
- p_e = expected agreement by chance

Use this when comparing automated coding to human gold standard.

Break (15 minutes)

The bag-of-words assumption

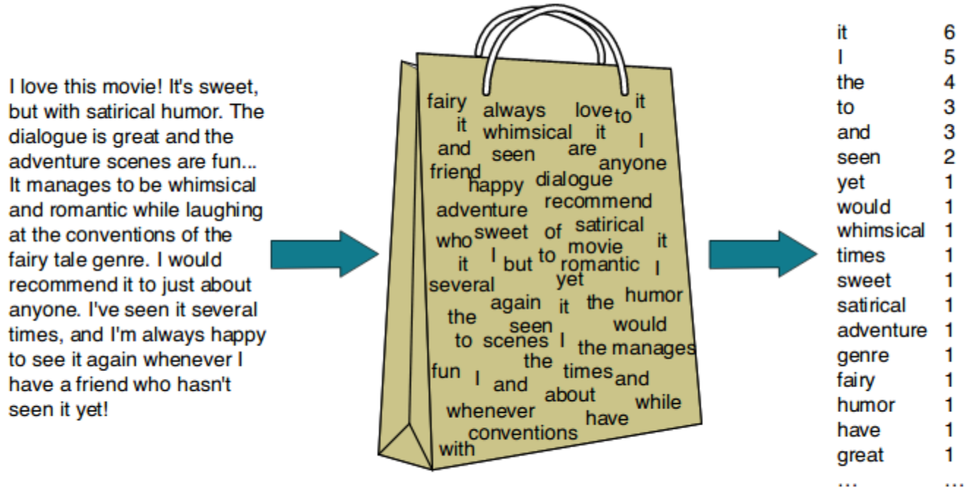
Core idea: A document is an unordered collection of words.

Throw away:

- Word order
- Grammar
- Sentence structure

Keep only:

- Which words appear
- How often



What we lose

“The dog bit the man”

↓

{the: 2, dog: 1, bit: 1, man: 1}

“The man bit the dog”

↓

{the: 2, dog: 1, bit: 1, man: 1}

Identical representations!

What we gain

Mathematical tractability

- Documents become vectors
- We can compute distances, similarities
- Standard ML algorithms apply

Scalability

- Simple operations
- Efficient computation
- Works on millions of documents

Surprising effectiveness

- Many tasks care more about “what words” than “what order”
- Topics, sentiment, authorship often work fine

Building the Document-Term Matrix

Step 1: Define your vocabulary (all unique words)

Step 2: For each document, count each word

Step 3: Arrange as matrix

	inflation	economy	growth	crisis
Doc 1	3	2	1	0
Doc 2	1	5	4	0
Doc 3	7	2	0	5

Properties of the DTM

High dimensionality

- Vocabulary of 10,000+ words = 10,000+ columns
- Can be millions for large corpora

Sparsity

- Most cells are zero
- Any one document uses tiny fraction of vocabulary

Zipf’s law

- Few words are very common
- Most words are very rare

From counts to weights: TF-IDF

Raw counts treat all words equally.

But “the” appearing 10 times ≠ “inflation” appearing 10 times.

TF-IDF reweights by informativeness:

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

TF-IDF: The components

Term Frequency (TF): How often does word t appear in document d?

$$\text{TF}(t, d) = \frac{\text{count}(t, d)}{\text{total words in } d}$$

Inverse Document Frequency (IDF): How rare is word t across all documents?

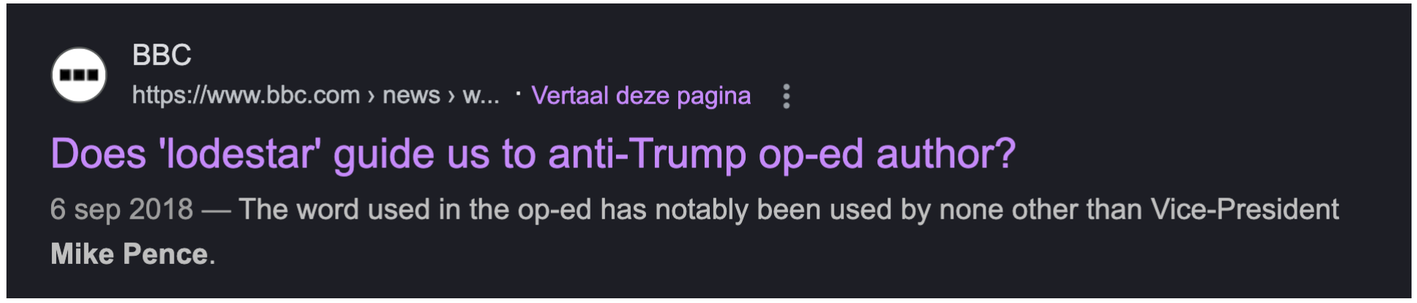
$$\text{IDF}(t) = \log \frac{N}{\text{documents containing } t}$$

TF-IDF intuition

Word	Doc 1 TF	Corpus DF	IDF	TF-IDF
the	0.08	1000/1000	0	0
economy	0.04	800/1000	0.22	0.009
stagflation	0.02	10/1000	4.6	0.092

“Stagflation” has lower TF but much higher TF-IDF.

It’s distinctive.



Summary: DTM representations

Type	Cell values	Use case
Binary	0 or 1	Word presence only
Count	Raw frequency	Simple frequency analysis
Normalized	Count / doc length	Comparable across lengths
TF-IDF	Weighted by distinctiveness	Classification, similarity

Coding Exercise 1

Exercise 1: Building a Document-Term Matrix

Goal: Create DTM and TF-IDF representations of the Brexit corpus

Tasks:

1. Load the Brexit speeches data
2. Create a count-based DTM using `CountVectorizer`
3. Create a TF-IDF matrix using `TfidfVectorizer`
4. Examine vocabulary size and sparsity
5. Compare raw counts vs. TF-IDF for specific words
6. Identify most distinctive words per party

Time: 30 minutes, work in pairs

Break (15 minutes)

Part 3: Frequency analysis in practice

From matrix to analysis

Now we have documents as vectors.

What can we do with them?

- **Frequency analysis**: What words are common/rare?
- **Comparison**: How do vocabularies differ across groups?
- **Similarity**: Which documents are alike?
- **Classification**: Can we predict categories?

Word frequency distributions

Zipf’s Law revisited: word frequency vs. rank

A few words are very common; most words are rare.

Implications:

- The long tail contains distinctive vocabulary
- High-frequency words often uninformative
- Preprocessing choices matter enormously

Comparing vocabularies across groups

Research question: How do Labour and Conservative MPs talk differently about Brexit?

Approach:

1. Separate corpus by party
2. Calculate word frequencies or TF-IDF per group
3. Identify words that distinguish groups

Keyness: Statistical distinctiveness

- Observed frequency in corpus A
- Expected frequency if A and B used same vocabulary

High keyness = significantly more common in A than B

Commonly used in corpus linguistics (Rayson & Garside 2000)

Limitations of frequency analysis

What frequency captures:

- Word salience (what's talked about)
- Vocabulary differences across groups
- Topic presence

What frequency misses:

- Meaning, context, nuance
- Negation ("not good" = good + not)
- Syntax, relationships between words
- Same word, different meanings (polysemy)

When frequency analysis is enough

✓ Measuring visibility/salience of entities or topics

✓ Comparing vocabulary across groups

✓ Tracking word usage over time

✓ Exploratory analysis of corpus content

✓ Simple content analysis at scale

Part 4: The limits of counting

The negation problem

Dictionary says “good” is positive.

But:

- “not good” → negative
- “not very good” → negative
- “wouldn’t say it’s good” → negative
- “good for nothing” → negative

Bag of words sees: {good: 1, not: 1, …}

The polysemy problem

“Bank” means:

- Financial institution (“the bank increased rates”)
- River edge (“sitting by the river bank”)
- Verb for turning (“bank left at the intersection”)
- Reliance (“bank on this strategy”)

Bag of words: bank = bank = bank = bank

The synonymy problem

Different words, same meaning:

- “happy” / “glad” / “joyful” / “pleased”
- “car” / “automobile” / “vehicle”
- “decrease” / “decline” / “drop” / “fall”

Bag of words: all different features

No recognition of semantic similarity

The compositionality problem

Meaning often comes from combination:

- “hot dog” ≠ hot + dog
- “kick the bucket” ≠ kick + bucket
- “white house” ≠ white + house (when capitalized)

Multi-word expressions, idioms, named entities

What we want from a representation

1. **Similar words should be close together**
    - “good” near “excellent”
    - “good” far from “bad”
2. **Context should matter**
    - “bank” in financial context ≠ “bank” in nature context
3. **Relationships should be preserved**
    - king:queen :: man:woman

This is what embeddings provide.

Text classification

Simple, but very good use case!

Goal: Use text classification strategies to predict whether an email is ham (0) or spam (1). Predict classes given the message of a text.

Naive Bayes: $P(\text{Class} \mid \text{Text}) \propto P(\text{Text} \mid \text{Class})\, P(\text{Class})$

- Model word probabilities per class
- We choose the class with highest probability
- Assumption that each word contributes independently to the classification is **wrong** but **useful**

Part 5: Setting the scene

Where we've been

```
Raw text
    → Preprocessing (tokenization, lemmatization, stopwords)
        → Representation (DTM, TF-IDF)
            → Analysis (frequency, similarity, classification)
```

This pipeline underlies most traditional text analysis.

Limitations: no semantics, no context, no compositionality.

Where we're going

Word embeddings (Word2Vec, GloVe, 2013–)

- Dense vectors that capture semantic relationships
- Pre-trained on massive corpora
- king − man + woman ≈ queen

Contextual embeddings (BERT, GPT, 2018–)

- Same word gets different vectors in different contexts
- Pre-trained on next-word prediction
- Fine-tunable for specific tasks

Large Language Models (GPT-4, Claude, 2022–)

- Zero-shot classification through prompting
- In-context learning

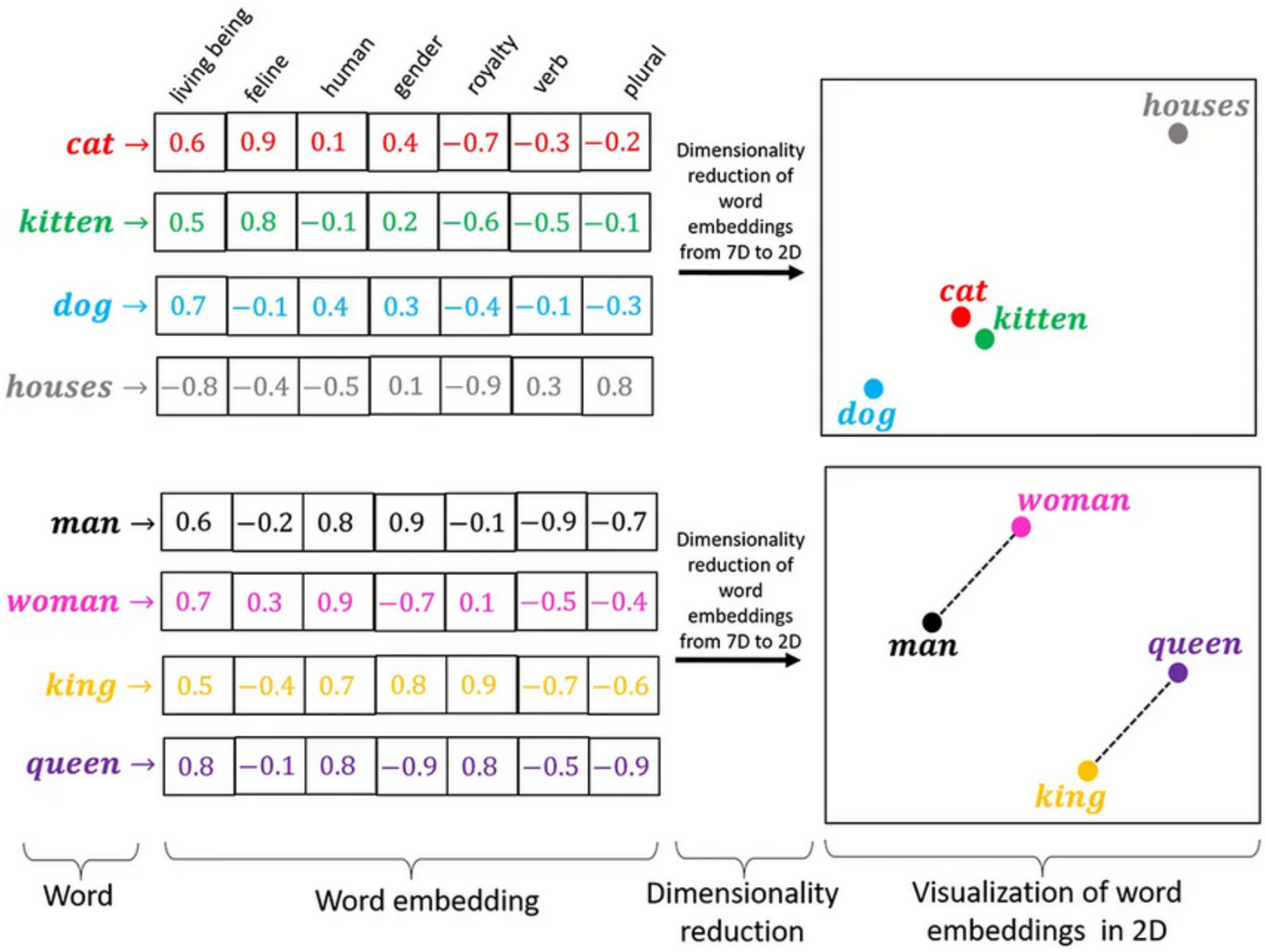
Word embeddings: The key insight

Distributional hypothesis: “You shall know a word by the company it keeps” (Firth 1957)

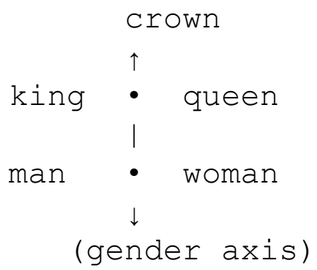
Words appearing in similar contexts have similar meanings.

Train a model to predict context → words with similar contexts get similar vectors.

Word embeddings: Visualization I



Word embeddings: Visualization II



Semantic relationships become geometric relationships.

king − man + woman ≈ queen

From words to documents

Simple approach: Average the word vectors

$$\vec{d} = \frac{1}{n} \sum_{i=1}^{n} \overrightarrow{w_i}$$

Better than bag-of-words because word vectors capture similarity.

“Happy” and “joyful” contribute similarly.

Contextual embeddings: The next step

Problem with static embeddings:

“bank” has one vector, regardless of context.

Solution: Make embeddings depend on context.

BERT, GPT, and other transformers learn contextual representations.

“The river bank was muddy” → different “bank” vector

“The investment bank collapsed” → different “bank” vector

LLMs as classifiers

Traditional approach:

Train a classifier on labeled data.

LLM approach:

Just ask the model!

```
Classify this text as positive, negative, or neutral:
"The fiscal outlook remains challenging despite recent gains."

Answer: Negative
```

A teaser: Same task, different methods

Brexit speech sentiment:

Method	Approach	Performance
Dictionary (VADER)	Count positive/negative words	Baseline
Bag-of-words + SVM	Train classifier on TF-IDF	Better
Fine-tuned BERT	Transfer learning	Much better
LLM prompting	Zero-shot classification	Comparable or better

The tradeoffs

Method	Interpretability	Data needed	Compute	Performance
Dictionary	High	None	Low	Variable
BoW + ML	Medium	Moderate	Low	Good
Embeddings	Low	Less	Medium	Better
LLMs	Low	None	High	Best*

*For many tasks; highly dependent on prompting.

Practical guidance

Start simple, add complexity only if needed.

1. Establish human baseline (inter-coder reliability)
2. Try dictionary methods if concept is well-defined
3. Try supervised learning if you have labeled data
4. Consider embeddings if little labeled data
5. Use LLMs for exploration and hard cases
6. **Validate everything**

Wrapping up

Key takeaways

1. **Classification metrics** (precision, recall, F1) formalize evaluation
2. **Bag-of-words** is simple but surprisingly effective
3. **TF-IDF** emphasizes distinctive words
4. **Similarity and classification** flow from representation
5. **Embeddings and LLMs** address BoW limitations—coming soon!

What’s next: Spring workshop

Machine Learning and LLMs for Text Analysis

- Word embeddings (Word2Vec, GloVe)
- Transfer learning with pre-trained models
- Fine-tuning BERT for classification
- Using LLMs (GPT, Claude) for text analysis
- Prompt engineering for zero-shot classification

Resources for further learning

Books:

- Grimmer, Roberts & Stewart (2022). *Text as Data*
- Jurafsky & Martin (2024). *Speech and Language Processing* (free online)

Papers:

- Gentzkow, Kelly & Taddy (2019). “Text as Data” *JEL*
- Grimmer & Stewart (2013). “Text as Data” *Political Analysis*

Tutorials:

- scikit-learn text tutorial
- spaCy course (free)
- Hugging Face NLP course (free)

Thank you!

Questions?

References

Firth, J. R. (1957). A synopsis of linguistic theory 1930-55. *Studies in Linguistic Analysis*.

Gentzkow, M., Kelly, B., & Taddy, M. (2019). Text as data. *Journal of Economic Literature*, 57(3), 535-574.

Grimmer, J., Roberts, M. E., & Stewart, B. M. (2022). *Text as Data: A New Framework for Machine Learning and the Social Sciences*. Princeton University Press.

Grimmer, J., & Stewart, B. M. (2013). Text as data: The promise and pitfalls of automatic content analysis. *Political Analysis*, 21(3), 267-297.

Rayson, P., & Garside, R. (2000). Comparing corpora using frequency profiling. *Proceedings of the Workshop on Comparing Corpora*.