Document Representations & Topic Models
Multimodal Computational Methods in Political Science

Tamara Grechanaya [1]

[1]LMU Munich — Computational Social Science MA Program

May, 2026

Course Roadmap

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

So far we've worked with **words**. Today we move to **documents**: how to represent entire texts as vectors, and how to discover latent themes in a corpus without labeled data. This is the most "political science methods" lecture of the course.

Today's Outline

Part I: Document Representations

- From words to documents
- Averaging word embeddings
- TF-IDF weighted averaging
- Doc2Vec
- Using doc vectors for classification & similarity

Part II: Introduction to Topic Models

- Supervised vs. unsupervised methods
- What is a "topic"?
- The generative story
- Why topic models?

Part III: Latent Dirichlet Allocation

- LDA's generative model
- Two distributions: topics and documents
- Inference intuition
- Interpreting LDA output
- Choosing K (number of topics)

Part IV: Politics & Practice

- Structural Topic Model (STM)
- Validation strategies
- Practical: topic model U.K. Commons speeches

Table of Contents

The Challenge: Representing Documents

Lecture 3 gave us vectors for individual words. But political scientists rarely study single words in isolation — we study **documents**: speeches, manifestos, articles, tweets, legislative bills.

The core question

How do we get a **single vector** that represents an entire document, so we can:

- Measure similarity between documents?
- Feed documents into a classifier?
- Cluster documents by topic?
- Compare corpora (e.g., left-wing vs. right-wing media)?

The methods we already know:

- **Bag-of-words** (Lecture 1): each document is a sparse count vector. Already gives us a "document representation" — but in 50,000+ dimensions, with no semantic structure.
- **TF-IDF** (Lecture 1): same, with weights. Still sparse.

The new question: Can we build *dense*, low-dimensional document representations that preserve semantic meaning?

Approach 1: Averaging Word Embeddings

The simplest idea: If each word in the document has a 300-dimensional embedding, take the **average** across all words in the document.

$$\mathbf{v}_{\text{doc}} = \frac{1}{|d|} \sum_{w \in d} \mathbf{v}_w$$

Averaging in action

Speech: "The climate crisis requires decisive government action"
After preprocessing, tokens: `[climate, crisis, requires, decisive, government, action]`
Document vector:

$$\mathbf{v}_{\text{doc}} = \frac{1}{6} (\mathbf{v}_{\text{climate}} + \mathbf{v}_{\text{crisis}} + \mathbf{v}_{\text{requires}} + \mathbf{v}_{\text{decisive}} + \mathbf{v}_{\text{government}} + \mathbf{v}_{\text{action}})$$

Result: a single 300-dimensional vector representing the entire speech.

Surprisingly effective despite its simplicity! Used as a baseline in many papers. Works especially well for short documents (tweets, headlines, single sentences).

Averaging: Strengths and Limitations

Strengths:

- Extremely simple
- Fast to compute
- Low-dimensional (300 instead of 50,000)
- Captures general topic/theme of a document
- Works well for classification tasks
- Meaningful cosine similarities between documents

Limitations:

- **Ignores word order entirely.** "The dog bit the man" and "The man bit the dog" average to similar vectors.
- **Longer documents get washed out.** Averaging hundreds of words dilutes distinctive content.
- **Treats all words equally.** Rare, topic-specific words contribute the same as common ones.
- **Stopwords distort results.** If you haven't removed them, they dominate the average.

Key Concept

The averaging approach is a **strong baseline**: fast, simple, often good enough. Before trying anything more sophisticated, always test whether averaged embeddings already solve your problem.

Approach 2: TF-IDF Weighted Averaging

Fix one of the limitations: weight words by their importance.

$$\mathbf{v}_{\text{doc}} = \frac{\sum_{w \in d} \text{TF-IDF}(w, d) \cdot \mathbf{v}_w}{\sum_{w \in d} \text{TF-IDF}(w, d)}$$

What changes:

- Common words like "the" and "is" get low weights (low IDF)
- Distinctive, topic-specific words get high weights
- The document vector reflects the *most distinctive* word meanings, not the overall average

Illustration

Speech with 200 words. 150 of them are common function words; 50 are substantive.
Plain average: the 150 common words dominate → vector looks like "generic English text".
TF-IDF weighted: the 50 substantive words dominate → vector reflects actual content.

Usually performs better than plain averaging. Still ignores word order. Still a single fixed vector per document.

Approach 3: Doc2Vec (Paragraph Vectors)

Le & Mikolov (2014): extend Word2Vec to learn a vector for each entire document directly. **The core idea:**

- Word2Vec learns word vectors by predicting context words from a target word.
- Doc2Vec adds an extra "document vector" to this setup.
- The document vector is trained jointly with word vectors to predict context words in that document.
- After training, each document has its own learned vector in the same space as words.

Two variants:

- **PV-DM** (Distributed Memory): like CBOW but with a document vector added to the context
- **PV-DBOW** (Distributed Bag of Words): like Skip-gram, where the document vector predicts words

Advantages over averaging:

- Captures some document-level structure
- The document vector is optimized directly for the task
- Each document gets a unique, learned representation

In practice: often marginal improvement over TF-IDF weighted averaging. Less widely used today due to BERT.

Comparing Document Representation Methods

Method	How it works	Dimensionality	When to use
Bag-of-Words	Raw word counts	$\|V\|$ (sparse)	Simple classification, baseline
TF-IDF DTM	Weighted counts	$\|V\|$ (sparse)	Standard text classification
Averaged embeddings	Mean of word vectors	d (e.g., 300)	Quick doc similarity, small corpora
TF-IDF weighted emb.	Weighted mean	d	Better semantic similarity
Doc2Vec	Trained doc vectors	d	Research, custom corpora
BERT (preview)	Contextual, layered	768+	State-of-the-art (Lecture 7)

Practical advice

For most political science applications, **TF-IDF weighted averages of pre-trained embeddings** give you 90% of the quality with 10% of the complexity. Start there; move to fancier methods only if you need them.

A Different Question: Unsupervised Discovery

So far, everything we've done has been about producing representations for downstream tasks. But what if we don't know what we're looking for?

A motivating problem

You have 50,000 parliamentary speeches from the U.K. House of Commons. You want to answer:

- What issues dominate the political agenda?
- How does issue attention change over time?
- Do different parties talk about different things?

You *could* hand-code all 50,000 speeches by topic — but that would take years. And you'd have to decide the topic list in advance.

The idea behind topic models: Let the data tell you what topics are there. No labels, no pre-defined categories. Just: *what are the latent themes in this corpus?*

This is unsupervised learning. The machine learns structure without being told what to look for.

Table of Contents

Supervised vs. Unsupervised Methods

Supervised (Lectures 2, 5, 7)

- Requires labeled training data
- Model learns to predict known categories
- Evaluated by accuracy/precision/recall
- Examples: sentiment, stance, party, topic coding

You know what you want to find. The challenge is training a model to find it at scale.

Unsupervised (Lecture 4)

- No labels required
- Model discovers structure in the data
- Evaluated by interpretability and substantive validity
- Examples: topic models, clustering, dimensionality reduction

You don't know what you'll find — the method surfaces latent patterns for *you* to interpret.

When to use which

Use **supervised** methods when you have a clear classification task and can (affordably) produce labels. Use **unsupervised** methods for exploratory analysis, when labels are impossible to get, or when you want the data to reveal structure you didn't anticipate.

What Is a "Topic"?

Intuitive definition: a topic is a **set of words that tend to appear together** because they are about the same thing.

Hypothetical topics in U.K. Commons speeches

Topic A (Climate Policy):
`climate, energy, emissions, renewable, carbon, environment, sustainability, net-zero`

Topic B (Labour & Social Policy):
`workers, wages, pensions, benefits, inequality, minimum-wage, unions, employment`

Topic C (Foreign Affairs):
`europe, international, alliance, nato, diplomacy, security, cooperation`

Topic D (Immigration):
`immigration, asylum, refugees, border, integration, visa, deportation`

Formally (for the models we'll see today): a topic is a **probability distribution over words**. Each word has some probability of appearing in the topic; topic-related words have high probability.

The Generative Story

Topic models are "generative": they describe an imagined **process** by which documents are created.

Imagine writing a parliamentary speech

An MP wants to give a speech. Here's the (fictional) process:

1. She decides what **mix of topics** her speech will cover: "60% Climate, 30% Economy, 10% Foreign Policy".
2. For each word she's going to say:
   a. She picks a topic from her mix (e.g., roll a weighted die: 60% chance Climate, 30% Economy, 10% Foreign)
   b. Given the topic, she picks a word from that topic's word distribution (e.g., the Climate topic has high probability on "energy", "climate", "carbon", ...)
3. Repeat until the speech is done.

Of course, nobody actually writes like this. But this is a useful *model* of the data: it gives us a mathematical framework to work backward from observed documents to the underlying topics.

The algorithm's job: given the observed speeches (and only the speeches), figure out what topics must have existed and how they were mixed in each speech.

Why Topic Models Matter for Political Science

Topic models are one of the most widely used quantitative text methods in political science, second only to dictionary methods. Hundreds of published papers use them. **What topic models let you do:**

1. **Discover issue dimensions** without pre-specifying them. "What topics appear in congressional debates?" without choosing the list in advance.
2. **Measure attention to issues over time or across actors.** "How much did immigration dominate parliamentary speeches in 2015 vs. 2020?" You get per-document topic proportions.
3. **Compare agendas across groups.** "Do Labour and Conservatives talk about different things, or about the same things differently?"
4. **Track framing.** Within a topic (e.g., immigration), which sub-frames get used by which actors?
5. **Generate hypotheses for further study.** Topic models surface patterns you wouldn't have thought to look for.

Seminal political science applications: Quinn et al. (2010) on the U.S. Senate, Grimmer (2010) on Senate press releases, Roberts et al. (2014) on open-ended survey responses (introducing STM).

Table of Contents

LDA: The Most Popular Topic Model

Latent Dirichlet Allocation (LDA), introduced by Blei, Ng & Jordan (2003), is the workhorse topic model.

Three key entities in LDA:

Documents

Each document is a mixture of topics. The proportions sum to 1. Doc 1: [0.6 Climate, 0.3 Economy, 0.1 Foreign] Doc 2: [0.1 Climate, 0.7 Immigration, 0.2 Security]

Topics

Each topic is a distribution over words. The word probabilities within a topic sum to 1. Climate: [climate: 0.08, energy: 0.05, carbon: 0.03, . . .]

Words

Each word in each document is attributed to one topic (drawn from that document's topic mixture). Word "climate" in doc 1: attributed to the Climate topic.

Two key distributions

LDA learns:

- θ_d: the topic distribution for document d (how much is the document about each topic?)
- ϕ_k: the word distribution for topic k (what words define this topic?)

The LDA Generative Process (Formally)

To generate a corpus of D documents with K topics, LDA says:

1. For each topic $k \in \{1, \ldots, K\}$:
   - Draw a word distribution $\phi_k \sim \text{Dirichlet}(\beta)$
2. For each document $d \in \{1, \ldots, D\}$:
   - Draw a topic distribution $\theta_d \sim \text{Dirichlet}(\alpha)$
   - For each word position $n \in \{1, \ldots, N_d\}$ in the document:
     - ⋆ Draw a topic assignment $z_{d,n} \sim \text{Multinomial}(\theta_d)$
     - ⋆ Draw a word $w_{d,n} \sim \text{Multinomial}(\phi_{z_{d,n}})$

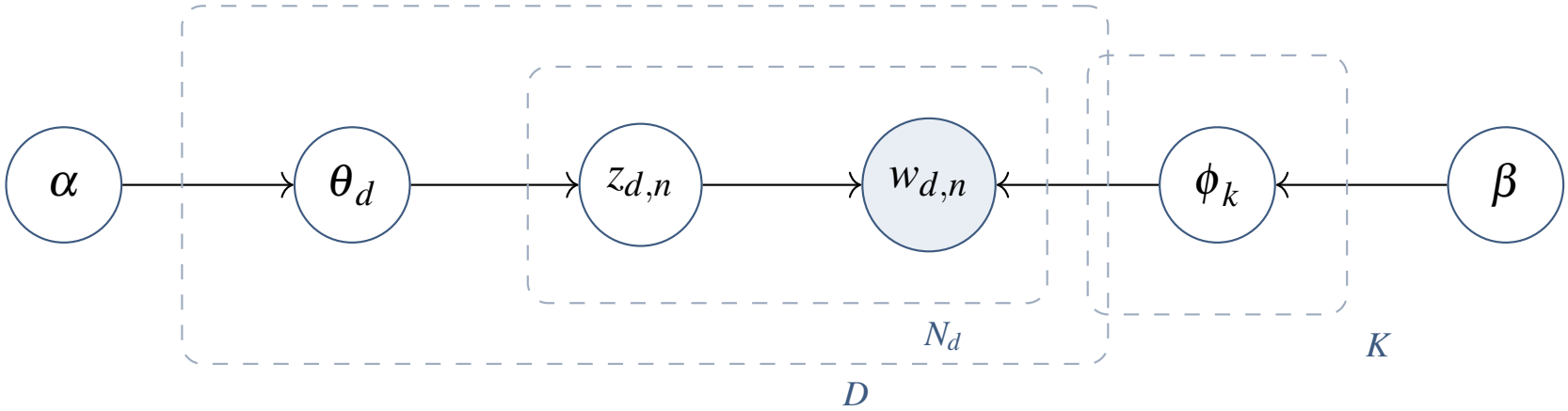
Don't panic at the notation. The key insight is:

- Each topic gets a word distribution (ϕ_k)
- Each document gets a topic distribution (θ_d)
- Each word is assigned to one topic

The **Dirichlet** priors (α, β) are the "Latent Dirichlet Allocation" part of the name. They control how concentrated the distributions are — we'll discuss this.

LDA as a Plate Diagram

The canonical graphical representation of LDA:



Reading the diagram:

- Shaded node ($w_{d,n}$) = observed
- Unshaded nodes = latent (to be inferred)
- Arrows = generative dependency
- Plates = "repeat this for each ..."

In words:

- Repeat for each of K topics: draw word distribution
- Repeat for each of D documents: draw topic proportions
- Repeat for each of N_d words: draw topic, then word

The Dirichlet Priors: What Do They Control?

The hyperparameters α and β control the "shape" of the distributions LDA learns.

α: document-topic prior

- **Small α** (< 1): each document is about **few topics** (concentrated).
- **Large α** (> 1): each document is about **many topics** (spread out).

Rule of thumb: use $\alpha = 50/K$ or let the algorithm estimate it.

β: topic-word prior

- **Small β** (< 1): each topic has **few high-probability words** (sharply defined topics).
- **Large β**: topics spread probability over **many words** (less sharp).

Rule of thumb: $\beta = 0.01$ to 0.1.

Key Concept

You don't need to deeply understand Dirichlet distributions to use LDA effectively. The defaults in standard software (`topicmodels`, `gensim`) usually work fine. Only tune these priors if you have a specific reason.

Inference: Going Backward from Documents to Topics

We observe the documents. We need to infer: the topics, the topic mixtures for each document, and the topic assignment for each word.
This is hard. Exact inference is computationally intractable. In practice, we use approximation methods:

- **Gibbs Sampling:** iteratively reassign each word to topics based on the current state. After many iterations, the samples approximate the true distribution. Used by `topicmodels` in R.
- **Variational Inference:** approximate the true posterior with a simpler distribution, then optimize. Used by `gensim` in Python.

Practical implications:

- Both methods are **stochastic**: you'll get slightly different results each time (set a random seed for reproducibility!)
- Training can be slow for large corpora (minutes to hours)
- You won't implement these yourself — use the packages

For those interested in the math: Blei's original LDA paper (2003) is a classic read. But for applied work, you can treat inference as a black box and focus on interpretation and validation.

What LDA Gives You: The Output

After training, LDA produces two matrices:

1. Topic-word matrix ($K \times |V|$)
Each row is a topic; each column is a word. Entries are $P(\text{word} \mid \text{topic})$.

	climate	energy	workers	wages
Topic 1	0.08	0.07	0.001	0.0005
Topic 2	0.002	0.001	0.06	0.05
Topic 3	0.001	0.001	0.002	0.001
...				

Rows sum to 1. Used to interpret what each topic is about.

2. Document-topic matrix ($D \times K$)
Each row is a document; each column is a topic. Entries are $P(\text{topic} \mid \text{document})$.

	T1	T2	T3	T4	T5
Speech 1	0.62	0.03	0.05	0.20	0.10
Speech 2	0.04	0.55	0.08	0.23	0.10
Speech 3	0.10	0.10	0.70	0.05	0.05
...					

Rows sum to 1. Used to measure topic attention per document.

These two matrices are the output of LDA — everything else (interpretation, visualization, analysis) is built on top of them.

Interpreting Topics: The Top-Words Approach

How do you figure out what a topic is about? Look at the words with the highest probability in that topic.

Hypothetical LDA output for U.K. Commons speeches

Topic 7 (top 10 words by probability):
`climate, energy, carbon, renewable, emissions, environment, net-zero, fuel, green, sustainability`
→ **You label it:** "Climate & Energy Policy"

Topic 12:
`workers, wages, employment, pension, benefits, inequality, unions, minimum, welfare, jobs`
→ **You label it:** "Labour & Social Policy"

Topic 23:
`europe, union, brussels, commission, member, single-market, council, parliament, euro, treaty`
→ **You label it:** "European Integration"

Important: LDA gives you the top words; **you** give the topic a label based on substantive interpretation. Labeling is *your* interpretive contribution as a researcher.

FREX and Highest-Probability Words

Problem with top-probability words: some words appear in *many* topics because they're common overall. This makes topic interpretation ambiguous.

The problem

A topic's top-probability words might include "parliament", "government", "minister" — which don't distinguish it from any other political topic. These words are just frequent everywhere.

Solution: FREX (FRequency + EXclusivity) words. Words that are both:

- **Frequent** within this topic
- **Exclusive** to this topic (rare in other topics)

FREX vs. top-probability

Topic about immigration policy:
Top probability: `government, people, country, parliament, immigration, border, ...`
FREX: `asylum-seeker, deportation, visa, refugee-status, detention, safe-country, ...`

FREX words are much more informative for interpretation. The `stm` package reports FREX by default. Always look at both.

Choosing the Number of Topics: K

LDA requires you to specify the number of topics K in advance. This is one of the hardest decisions in applied topic modeling.

What happens with different K:

- K **too small** (e.g., $K = 5$): topics are very broad, mix unrelated themes together
- K **just right**: topics are interpretable and cover the substantive structure of the corpus
- K **too large** (e.g., $K = 200$): topics become redundant, fragmented, hard to interpret

There is no "correct" K. Different values surface different patterns.

Approaches to choosing K:

1. **Substantive:** pick based on prior knowledge of the corpus and research question
2. **Statistical:** fit multiple K values and use metrics (perplexity, coherence, exclusivity)
3. **Exploratory:** try several values ($K = 10, 20, 50, 100$), compare qualitatively
4. **Practical:** pick the smallest K where topics look distinct and interpretable

Diagnostic Metrics for Topic Models

Several quantitative metrics help you compare different values of K:

- **Perplexity (held-out likelihood):** how well does the model predict held-out documents? Lower is better. Usually *decreases* monotonically with K, so it's not always useful for choosing K.
- **Semantic coherence:** do the top words of each topic tend to co-occur in the same documents? Higher is better. Tends to *decrease* as K grows.
- **Exclusivity:** are the top words of each topic exclusive to that topic? Higher is better. Tends to *increase* with K.
- **Held-out likelihood:** test-set log likelihood. Robust measure of generalization.

The trade-off

Coherence and exclusivity often pull in opposite directions. The "best" K is usually a compromise — you pick a K that has a good balance of both and produces topics you can actually interpret.

Validating Topic Models

Grimmer & Stewart's Principle 4: validate, validate, validate. How?

1. **Face validity:** read the top words for each topic. Do they hang together semantically? Can you label the topic?
2. **Word intrusion test:** for each topic, show annotators the top 5 words plus 1 random "intruder" word. Can they identify the intruder? Good topics are easy; bad topics are hard. (Chang et al. 2009)
3. **Document intrusion test:** for each topic, show annotators 3 documents the model assigns to that topic plus 1 document it doesn't. Can they identify the intruder?
4. **Predictive validity:** do the topic proportions predict something you know (e.g., party, time, speaker)?
5. **Substantive validity:** do the topics match your prior knowledge of the corpus?
6. **Robustness:** do you get similar topics with different random seeds, different preprocessing, different K?

Table of Contents

Beyond LDA: The Structural Topic Model (STM)

LDA has a limitation: it ignores document metadata. But in political science, we almost always have metadata (party, date, speaker, chamber, country).
Structural Topic Model (STM): Roberts, Stewart, Tingley et al. (2014, 2016, 2019).
The key innovation: STM lets document-level covariates *affect* both:

- Which topics a document is about (**topical prevalence**)
- How a topic is discussed within a document (**topical content**)

STM in action

You're modeling U.K. Commons speeches and you have:

- **Prevalence covariates:** party, year, government/opposition status
- **Content covariate:** party (how does each party *discuss* each topic?)

STM can answer: *"Does the Climate topic get more attention after 2015?"*, *"Do Labour and Conservatives use different words when discussing Immigration?"* — all as part of the topic model itself.

Why STM Is the Default in Political Science

STM has become the standard topic model in political science research. Why?

1. **Incorporates covariates directly.** No need to fit topics first and then run a regression of topic proportions on covariates — it's all one coherent model.
2. **Better small-sample behavior.** STM uses initialization tricks (e.g., spectral initialization) that work better than LDA's random initialization for small corpora.
3. **Excellent R package.** The `stm` package (Roberts, Stewart, Tingley) is well-documented, has great diagnostics, and integrates with `quanteda`.
4. **Strong validation tools.** Built-in functions for coherence, exclusivity, topic correlation, word intrusion, and uncertainty estimates.
5. **Active community.** Widely taught, widely cited, widely extended. Many published political science papers use STM.

Recommendation: if you're doing topic modeling in political science, start with STM unless you have a specific reason to use plain LDA.

STM: A Political Example

Roberts, Stewart, Tingley (2019) applied to our course

Imagine fitting an STM to U.K. Commons speeches with:

- **Prevalence:** party, year, government/opposition status
- $K = 40$ **topics**

Possible findings:

- The "Immigration" topic gets significantly more attention from Reform/UKIP than from all other parties
- Attention to the "Climate" topic increases over time, with the largest increase from the Green Party
- The "Economy" topic is discussed very differently by Conservatives ("enterprise", "growth", "deregulation") vs. Labour ("inequality", "austerity", "public services")
- Government speeches cover different topics than opposition speeches (discovery!)

All of these can be estimated and visualized with statistical uncertainty directly from the STM output, without a separate regression step.

Limitations of Topic Models

Topic models are powerful — but they're not magic. Be aware of their limitations.

1. **Bag of words.** LDA and STM ignore word order (like Lecture 1). They can't distinguish "The dog bit the man" and "The man bit the dog".
2. **Sensitivity to preprocessing.** Different stop word lists, stemming choices, and minimum frequency thresholds can change the topics substantially. Document your choices.
3. **Sensitivity to K.** Different K values give genuinely different topic structures. No "correct" K.
4. **Randomness.** Different random seeds give different topics (though they should be qualitatively similar).
5. **Interpretation is subjective.** Two researchers might label the same topic differently.
6. **Short texts struggle.** Topic models need a decent number of words per document. Tweets are often too short; use sentence-aggregated documents or specialized methods.
7. **Validation is essential and non-trivial.** There's no single "correct" answer to check against.

When to Use Topic Models (and When Not To)

Good fit for topic models:

- Exploring a corpus you don't know well
- Discovering latent themes without pre-specified categories
- Measuring attention to issues over time / across actors
- Generating hypotheses for further study
- When you have $> 1{,}000$ medium-length documents
- When interpretation flexibility is welcome

Not a good fit:

- You need to classify documents into pre-defined categories $\rightarrow$ use supervised methods (Lecture 2)
- You need high precision on specific terms $\rightarrow$ use dictionary methods
- Documents are very short (tweets) $\rightarrow$ use embeddings + clustering, or BERTopic
- You need causal or precise numerical inference $\rightarrow$ topic models are too noisy
- Your corpus is small (< 500 docs) $\rightarrow$ topics will be unstable

Key Concept

Topic models are **exploratory and descriptive tools**, not precise measurement instruments. Use them to generate insights and hypotheses — then verify the findings with other methods.

A Brief Note on BERTopic

BERTopic (Grootendorst 2022) is a modern alternative that uses BERT embeddings instead of bag-of-words. **The idea:**

1. Embed each document using a pre-trained BERT model (Lecture 7)
2. Reduce the dimensionality of the embeddings (UMAP)
3. Cluster the embeddings (HDBSCAN)
4. Use class-based TF-IDF to extract the most distinctive words for each cluster

Advantages:

- Captures semantic similarity (not just word overlap)
- Works well on short texts (tweets, headlines)
- Multilingual out of the box

Disadvantages:

- Less statistical foundation than LDA/STM
- Harder to validate
- Python-only (no mature R package)

We'll mention BERTopic again in Lecture 7, once we've covered BERT. For today, focus on LDA and STM.

Table of Contents

Practical Session: Overview

Discovering Topics in U.K. Commons Speeches with STM

What we'll do:

1. Load and preprocess the U.K. Commons corpus (from Lecture 1)
2. Fit a Structural Topic Model with $K = 20$ topics
3. Interpret topics using top-probability and FREX words
4. Measure how topic attention varies by party
5. Run diagnostics to compare different values of K
6. Visualize topics and their covariate effects

Tools we'll use:

- `quanteda` for preprocessing
- `stm` for fitting the topic model
- `ggplot2` for visualization

```
install.packages(c("quanteda", "stm", "stminsights", "
    tidyverse"))
```

Practical: Discussion Questions

Work through these with the corpus you just modeled:

1. **Face validity**: look at the top 10 words (both highest probability and FREX) for each topic. Can you label every topic? Which topics are clear? Which are ambiguous or look like junk?
2. **Party effects**: for a topic where you expect a strong party difference (e.g., Climate for Greens, Immigration for Reform), does the `estimateEffect` plot confirm your expectation?
3. **Temporal trends**: pick a topic you expect to have changed over time (e.g., Brexit around 2016). Does the time trend look plausible?
4. **Robustness**: rerun with a different random seed (if possible) and different K. How stable are the main findings?
5. **What would a supervised classifier have missed?** What topics did the model *discover* that you wouldn't have pre-specified?

Table of Contents

Key Takeaways

1. **Document representations** range from simple (BoW, averaged embeddings) to sophisticated (Doc2Vec, BERT). Start simple; add complexity only when needed.
2. **Topic models** are unsupervised methods that discover latent themes in a corpus without labels. They are the most widely used exploratory text method in political science.
3. **LDA** represents each topic as a distribution over words and each document as a distribution over topics. The model infers these from the observed text.
4. **STM** extends LDA with document-level covariates, making it especially well-suited for political science research where metadata (party, date, speaker) matters.
5. **Choosing** K is not a solved problem. Use diagnostics, substantive knowledge, and interpretability to guide the choice. Report sensitivity to K.
6. **Validation is essential.** Read top words, inspect representative documents, run word intrusion tests, verify covariate effects match prior expectations.
7. **Topic models are descriptive, not measurement-grade.** Use them for exploration and hypothesis generation; verify findings with other methods.

Home Assignment (Optional)

Task: Fit and interpret a topic model for a political corpus. **Instructions:**

1. Choose a corpus (U.K. Commons speeches, U.S. Congressional Record, EU Parliament, party manifestos from the Manifesto Project, or another corpus of your choice). Minimum 1,000 documents.
2. Preprocess using what you learned in Lecture 1. Document your preprocessing choices.
3. Fit an STM with at least two different values of K (e.g., $K = 15$ and $K = 30$). Include at least one covariate.
4. For your chosen K, label all topics. Flag any junk/procedural topics.
5. Pick one substantive finding (e.g., "party X talks about topic Y more than party Z") and report it with the STM's uncertainty estimate.
6. **Validate**: run a word intrusion test on 3–5 topics (show the top 5 words plus one random word to a friend, ask them to pick the intruder; report how often they got it right).
7. Write a **2-page report**: corpus, preprocessing, topic labels, main findings, limitations.

Submit: R script + report. Due before Lecture 5.

Readings

Required:

- Blei, D.M. (2012). "Probabilistic Topic Models." *Communications of the ACM*, 55(4), 77–84.
- Grimmer, J., Roberts, M. E., & Stewart, B.M. (2022). Text as Data, Chapters 12 to 13.

Recommended:

- Roberts, M.E., Stewart, B.M., & Tingley, D. (2019). "stm: An R Package for Structural Topic Models." *Journal of Statistical Software*, 91(2), 1–40.
- Grimmer, J. (2010). "A Bayesian Hierarchical Topic Model for Political Texts." *Political Analysis*, 18(1), 1–35.

Other related readings:

- Quinn, K.M. et al. (2010). "How to Analyze Political Attention with Minimal Assumptions and Costs." *AJPS*, 54(1), 209–228.
- Roberts, M.E. et al. (2014). "Structural Topic Models for Open-Ended Survey Responses." *AJPS*, 58(4), 1064–1082.
- Blei, D.M., Ng, A.Y., & Jordan, M.I. (2003). "Latent Dirichlet Allocation." *JMLR*, 3, 993–1022.

Next week: Neural Networks & Sequence Models

Moving from count-based methods to deep learning: feedforward networks, RNNs, and LSTMs