Lecture 4 — Document Representations & Topic Models

Tamara Grechanaya
Multimodal Computational Methods for Political Science

Summer Semester 2026

Contents

Learning objectives

By the end of this lecture, you should be able to:

1. Explain three different ways to build a single vector representation for an entire document, and the trade-offs between them.
2. Distinguish supervised from unsupervised methods, and explain when each is appropriate.
3. Explain what a "topic" is in the context of a topic model, intuitively and formally as a probability distribution over words.
4. Walk through the generative story behind Latent Dirichlet Allocation (LDA) without getting lost in the math.
5. Interpret the two output matrices that LDA produces — the topic-word matrix and the document-topic matrix — and use top words and FREX words to label topics.
6. Choose a sensible number of topics (K) using diagnostics (coherence and exclusivity) plus substantive judgment.
7. Understand why the Structural Topic Model (STM) is widely used in political science and how it extends LDA with covariates.
8. Validate a topic model using face validity, word intrusion tests, and robustness checks.

1 Part I — From words to documents

1.1 The challenge

Lecture 3 gave us vectors for individual words. But political scientists rarely study single words in isolation — we study documents: speeches, manifestos, newspaper articles, tweets, legislative bills. To do almost anything substantive with these documents — measure their similarity, classify them by party, cluster them into themes, scale them on an ideological dimension — we need a single vector that represents the entire document.

We already have one such representation from Lecture 1: the bag-of-words (or TF-IDF weighted) vector. Each document becomes a row in the Document-Term Matrix, with one column per word in the vocabulary. This works, and it is the foundation of the classifiers we built in Lecture 2.

But the bag-of-words representation has the same fundamental limitations we discussed in Lecture 3. It is high-dimensional (50,000+ columns) and sparse (mostly zeros). It treats every word as completely independent of every other word. And it has no notion of semantic similarity: a document that uses "healthcare" and a document that uses "NHS" look as different from each other as a document about healthcare and a document about foreign policy.

Word embeddings give us dense, semantically meaningful vectors for individual words. The question for this lecture is: can we build dense, semantically meaningful vectors for *entire documents*?

1.2 Approach 1: Averaging word embeddings

The simplest approach is also surprisingly effective. If each word in a document has a 300-dimensional embedding, just take the average across all the words in the document:

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

The result is a single 300-dimensional vector that represents the entire document. Two documents with similar word content will have similar averaged vectors. A speech about climate policy will have an averaged vector that lies somewhere near the climate-related region of the embedding space, because most of its words will be climate-related and they will all pull the average in that direction.

This approach is fast, simple, and used widely in practice. It is often a strong baseline that more sophisticated methods barely beat. The strengths are obvious: it's much lower-dimensional than bag-of-words (300 dimensions instead of 50,000), it captures semantic content rather than just word identity, and it generalizes across synonyms (because "healthcare" and "NHS" have similar embeddings, documents using either word will have similar averaged vectors).

But averaging has clear limitations. It ignores word order entirely — even more dramatically than bag-of-words, since we now have only one number per dimension, not even per-word counts. Long documents tend to have averaged vectors that look like "generic political text" because the distinctive content gets diluted across many words. And every word contributes equally to the average, which means common words like "government" and "policy" can dominate over distinctive, topic-specific words that actually carry the meaning.

1.3 Approach 2: TF-IDF weighted averaging

We can fix the third problem by weighting the average. Multiply each word's embedding by its TF-IDF weight before averaging, then divide by the sum of TF-IDF weights:

$$\mathbf{v}_{\text{doc}} = \frac{\sum_{w \in d} \text{TF-IDF}(w, d) \cdot \mathbf{v}_w}{\sum_{w \in d} \text{TF-IDF}(w, d)} \tag{2}$$

The intuition is straightforward. Common words have low TF-IDF (because they appear in many documents) and therefore contribute little to the weighted average. Distinctive, topic-specific words have high TF-IDF and dominate the weighted average. The resulting document vector reflects the *most distinctive* words rather than the average word.

In practice, TF-IDF weighted averaging usually outperforms plain averaging, especially for longer documents. It is the standard recommendation when you need a quick document representation and don't want to deal with anything more complex.

1.4 Approach 3: Doc2Vec

Le and Mikolov (2014) extended Word2Vec to learn vectors for entire documents directly. The idea, called Doc2Vec or Paragraph Vectors, is the same self-supervised trick as Word2Vec: train a neural network on a prediction task, and the document vectors emerge as a byproduct.

Specifically, Doc2Vec adds an extra "document vector" alongside the word vectors. During training, the network is given context words plus the current document's vector, and it tries to predict the next word. The document vector is updated jointly with the word vectors. After training, each document has its own learned vector in the same space as the word vectors.

Two variants exist (PV-DM and PV-DBOW, paralleling CBOW and Skip-gram in Word2Vec), but the conceptual idea is the same. Doc2Vec captures some document-level structure that simple averaging misses, because the document vector is optimized directly to predict words in that specific document.

In practice, Doc2Vec offers only marginal improvements over TF-IDF weighted averaging in most political science applications. It has fallen somewhat out of favor since the rise of BERT-based document embeddings (which we will see in Lecture 7). For this course, you should know that Doc2Vec exists, but TF-IDF weighted averaging or BERT embeddings are more commonly used.

1.5 When to use which

For most political science applications, the practical recommendation is: **start with TF-IDF weighted averages of pre-trained word embeddings**. They give you 90% of the quality of more sophisticated methods with 10% of the complexity. Move to Doc2Vec or BERT embeddings only if you have a specific reason to.

But this lecture is not really about document vectors. It's about the question that motivates the rest of the lecture: what if you don't know what you are looking for in your corpus?

1.6 A different question: unsupervised discovery

Up to this point, every method we have covered has been about producing representations or predictions for a specific known task. We classified documents into known categories (Lecture 2). We measured similarity between known concepts (Lecture 3). The substantive question was already specified.

But often in research, we don't know what we're looking for. Imagine you have 50,000 parliamentary speeches from the 2017–2019 UK Parliament, and you want to know:

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

You could hand-code all 50,000 speeches by topic — but that would take years of work, and you'd have to decide your topic list in advance. The whole point of the analysis is that you don't know what topics are there.

This is the problem topic models were designed to solve: discover latent themes in a corpus *without* labels. Let the data tell you what topics are there. This is **unsupervised learning** — the machine learns structure without being told what to look for.

2 Part II — Introduction to topic models

2.1 Supervised vs. unsupervised methods

Before diving into topic models, let's situate them within the broader landscape of methods.

Supervised methods (Lectures 2, 5, 7) require labeled training data. A human (or a metadata source) provides the correct answer for each training example. The model learns to predict labels for new, unseen examples. Performance is evaluated objectively against held-out labels using metrics like precision, recall, and F1. You know what you want to find — the challenge is training a model to find it at scale.

Unsupervised methods (Lecture 4) require no labels. The model discovers structure in the data on its own. Performance is harder to evaluate because there is no objective ground truth. You don't know exactly what you'll find — the method surfaces latent patterns for you to interpret.

When should you 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 obtain, or when you want the data to reveal structure you didn't anticipate. Topic models fall squarely in the unsupervised camp.

2.2 What is a "topic"?

Intuitively, a topic is a set of words that tend to appear together because they are about the same thing. If you read a parliamentary speech that contains "climate," "energy," "emissions," "renewable," and "carbon," you would say it is about climate policy. If another speech contains "wage," "workers," "union," "social," and "rights," you would say it is about labor and social policy. The clusters of co-occurring words define the topics.

More formally, in the topic models we will study today, a topic is a **probability distribution over the entire vocabulary**. Each topic assigns some probability to every word in the corpus. Topic-related words have high probability; unrelated words have low probability.

Here's a concrete example of what an LDA topic might look like (top 10 words by probability, from a hypothetical model fit to UK Commons speeches):

- **Topic 7**: climate (0.08), energy (0.07), carbon (0.04), emissions (0.04), renewable (0.03), green (0.03), fossil (0.02), pollution (0.02), environment (0.02), nuclear (0.02)

You would label this topic "Climate and Energy Policy." The label is your contribution as a researcher; the model gives you the words.

2.3 The generative story

Topic models work by imagining a *generative process* — a fictional procedure by which documents are created — and then working backward from the observed documents to figure out what must have generated them.

Here is the generative story imagined by LDA, told as if an MP were writing a speech:

The MP first decides what mix of topics the speech will cover. Maybe she wants 60% climate, 30% economy, 10% foreign policy. This is the document-topic distribution. Then, for each word in the speech, she does two things. First, she rolls a weighted die to pick which topic the next word will come from — the die has 60% probability for climate, 30% for economy, 10% for foreign. Once she has picked a topic (say, climate), she draws a word from that topic's word distribution. The climate topic has high probability on words like "energy," "emissions," "carbon," and so on. So the next word she writes is likely one of those.

She repeats this process — pick topic, then pick word — for every word in the speech.

Of course, no MP actually writes this way. The generative story is not meant to be a realistic model of how speeches are produced. It is a useful *mathematical* model that lets us work backward: given observed speeches, what topic-word distributions and document-topic distributions must have existed to produce them?

This backward inference is what the LDA algorithm does. We give it a corpus of documents; it gives us back the topics and the topic mixtures.

2.4 Why topic models matter for political science

Topic models are one of the most widely used quantitative text methods in political science. Hundreds of published papers use them. They let you do four things that are difficult or

impossible with supervised methods.

Discover issue dimensions without pre-specifying them. The Manifesto Project hand-codes manifesto sentences into 56 pre-defined policy categories. Topic models can discover the latent issue structure of a corpus without any pre-defined list — and they sometimes find dimensions that the pre-defined codes miss.

Measure attention to issues over time or across actors. Once you have fitted a topic model, you have, for each document, the proportion devoted to each topic. You can plot how much attention each party gave to migration over time, or compare the salience of EU integration in different countries' parliaments.

Compare agendas across groups. Do Greens and Conservatives talk about different things, or about the same things differently? Topic models can answer both questions.

Generate hypotheses for further study. Topic models surface patterns you wouldn't have thought to look for. Reading the top words of a topic that the model discovered (one you didn't predict) often suggests new research questions.

The seminal political science applications include Quinn et al. (2010) on the U.S. Senate, Grimmer (2010) on Senate press releases, and Roberts et al. (2014) introducing the Structural Topic Model on open-ended survey responses.

3 Part III — Latent Dirichlet Allocation (LDA)

3.1 The most popular topic model

LDA, introduced by Blei, Ng, and Jordan in 2003, is the standard topic model. Its math involves Dirichlet distributions and probabilistic inference, but the conceptual structure is straightforward and worth understanding even if the math is intimidating.

LDA has three key entities:

Documents are mixtures of topics. Each document has a distribution over topics: a list of probabilities that sum to 1, telling you what fraction of the document is about each topic. Document 1 might be (60% climate, 30% economy, 10% foreign policy). Document 2 might be (10% climate, 70% migration, 20% security). Each document gets its own mixture.

Topics are distributions over words. Each topic has a distribution over the vocabulary: a list of probabilities that sum to 1, telling you how characteristic each word is of that topic. The climate topic might give probability 0.08 to "climate," 0.07 to "energy," 0.04 to "emissions," and tiny probabilities to everything else.

Words are attributed to topics. In LDA's generative model, each word in each document is generated from one specific topic, drawn from that document's topic mixture. The word "energy" in Document 1 is attributed to the climate topic; the word "tax" in the same document might be attributed to the economy topic.

The model's job, given the observed documents, is to figure out two things: the word distributions for each topic ($\boldsymbol{\phi}_k$ for topic k) and the topic distributions for each document ($\boldsymbol{\theta}_d$ for document d).

3.2 The generative process formally

Without getting too deep into the math, here is what LDA assumes happens when documents are generated. For each topic k, draw a word distribution $\boldsymbol{\phi}_k$ from a Dirichlet distribution with parameter $\boldsymbol{\beta}$. Then for each document d, draw a topic distribution $\boldsymbol{\theta}_d$ from a Dirichlet

distribution with parameter $\boldsymbol{\alpha}$. Then for each word position in the document, first draw a topic z from the document's topic distribution, then draw a word from that topic's word distribution.

Don't worry too much about the Dirichlet distributions. They are a mathematical convenience — they are the natural way to put probabilities on probability distributions, which is what we need here. The two key parameters are:

$\boldsymbol{\alpha}$: the document-topic prior. Small α means each document is concentrated on few topics. Large α means each document spreads probability over many topics.

$\boldsymbol{\beta}$: the topic-word prior. Small β means each topic has sharply defined high-probability words. Large β means topics spread probability over many words.

You will rarely need to tune these by hand. The defaults in standard software (`topicmodels` in R, `gensim` in Python, `stm` in R) work well in practice.

3.3 Inference: working backward

Given a corpus, LDA needs to infer the topics and the topic mixtures. This is a hard problem. Exact inference is mathematically intractable — there are too many possible assignments of words to topics to enumerate. So we use approximation methods.

Gibbs sampling is one such method. It iteratively reassigns each word to topics: for each word in each document, ask "given the current state of all other word assignments, what is the probability of assigning this word to each topic?" and then sample a new assignment. After many iterations, the samples converge toward the true posterior distribution.

Variational inference is another approach. It approximates the true posterior with a simpler distribution and optimizes the approximation. Variational inference is faster than Gibbs sampling and is what `stm` uses.

The practical implications for you are that both methods are stochastic — you'll get slightly different results each time you run the model unless you set a random seed. Training can be slow for large corpora, ranging from minutes for small datasets to hours for large ones. And you don't need to implement these methods yourself — the packages do it for you. Treat inference as a black box and focus on interpretation and validation.

3.4 What LDA gives you: the output

After training, LDA produces two matrices.

The topic-word matrix has K rows (one per topic) and $|V|$ columns (one per word). Entry (k, w) is $P(\text{word } w \mid \text{topic } k)$. Each row sums to 1. This matrix is what you use to interpret topics — by looking at the highest-probability words in each row.

The document-topic matrix has D rows (one per document) and K columns (one per topic). Entry (d, k) is $P(\text{topic } k \mid \text{document } d)$. Each row sums to 1. This matrix is what you use for downstream analysis — measuring topic prevalence per document, comparing topic attention across groups, tracking topics over time.

Everything you do with a topic model — interpretation, visualization, regression on topic proportions — is built on top of these two matrices.

3.5 Interpreting topics: top words and FREX

The most natural way to interpret a topic is to look at the words with the highest probability in that topic. In `topicmodels` and `stm`, this is what you get when you call `terms()` or

`labelTopics()`.

But there's a subtlety. Some words have high probability in one topic *and* in many other topics. They are frequent overall, so they appear in lots of topic distributions. These words are not very useful for distinguishing the topic. The top probability words for a political topic might include "government," "policy," "country," "people" — accurate but uninformative.

A better metric is **FREX**: words that are both **fre**quent within the topic and **ex**clusive to that topic. FREX words are rare in other topics, so they are diagnostic of *this* topic specifically.

Compare for a hypothetical migration topic:

- **Top probability words**: country, people, policy, government, immigration, must, should, important
- **FREX words**: refugee, asylum, border, deportation, hostile, sanctuary, undocumented, removal

The FREX list is much more informative. The `stm` package reports FREX automatically alongside top probability words. **Always look at both** when interpreting topics.

Another useful diagnostic is to find the documents that the model assigns most strongly to each topic and read them. If the highest-loading documents on a topic genuinely seem to be about that topic, the topic is real and well-formed. If they look like a random mix, the topic is junk.

3.6 Choosing the number of topics

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

Different values of K surface different patterns:

- K **too small** (say, $K = 5$): topics are very broad and mix unrelated themes together. You might get a single "domestic policy" topic that conflates climate, economy, and social issues.
- K **just right**: topics are interpretable and cover the substantive structure of the corpus.
- K **too large** (say, $K = 200$): topics fragment into redundant or barely-distinguishable variations. You might get five different "climate" topics that look almost the same.

There is no universally correct K. Different values reveal different aspects of the corpus structure. The practical approaches are:

Substantive judgment. Use prior knowledge of the corpus and your research question. If you're studying a parliamentary corpus and you know there are roughly 20 major policy areas, $K \approx 20$ is a reasonable starting point.

Statistical diagnostics. Fit models with different K values and compare metrics like semantic coherence (do top words tend to co-occur?) and exclusivity (are top words exclusive to that topic?). The `searchK()` function in `stm` automates this.

Exploratory comparison. Try several values ($K = 10, 20, 50$) and compare them qualitatively. Pick the smallest K where topics look distinct and interpretable.

The trade-off you'll see in diagnostics is that semantic coherence tends to *decrease* as K grows (because each topic is built from fewer words), while exclusivity tends to *increase*. The "best" K is usually a compromise between the two.

3.7 Validating topic models

Grimmer and Stewart's Principle 4 — validate, validate, validate — is especially important for topic models because the output is so easy to over-interpret. Several validation strategies are standard practice.

Face validity. Read the top words and FREX words for each topic. Do they hang together semantically? Can you label every topic? If a topic looks like word salad, treat it with suspicion.

Word intrusion. For each topic, show annotators the top 5 words plus 1 random "intruder" word from a different topic. Can they reliably identify the intruder? Good topics make the intruder obvious; bad topics don't. (Chang et al. 2009)

Document intrusion. For each topic, show annotators 3 documents the model assigns strongly to that topic plus 1 document it doesn't. Can they identify the intruder? This tests whether the document-topic assignments are coherent.

Predictive validity. Do the topic proportions predict something you know? If a topic looks like "labour and unions," it should appear more in Labour speeches than Conservative speeches. If it doesn't, something is wrong.

Substantive validity. Do the topics match what you know about the corpus? This isn't a weakness — it's how you check that the model isn't hallucinating. If a topic model on a parliamentary corpus produces no topic recognizably about Brexit, that's a red flag.

Robustness. Do you get similar topics with different random seeds, slightly different pre-processing, or nearby values of K? If a topic disappears when you change the seed, it wasn't real.

4 Part IV — Structural Topic Model (STM)

4.1 Why LDA isn't enough for political science

LDA treats all documents as exchangeable. It ignores any metadata you might have — party, year, speaker, country. But in political science, we almost always have metadata, and we almost always care about it. We want to know whether topic prevalence varies by party, whether it changes over time, whether it differs across countries.

You *can* fit LDA, then run a separate regression of topic proportions on covariates. But this two-step approach is statistically awkward — the topic proportions from the first step are estimated with uncertainty, and the regression in the second step doesn't account for that uncertainty.

The Structural Topic Model (STM), developed by Roberts, Stewart, Tingley and colleagues, integrates the two steps. Document-level covariates can affect both topic prevalence (how much each document is about each topic) and topic content (how each topic is discussed in different groups). The whole thing is one coherent model with proper uncertainty quantification.

4.2 How STM extends LDA

In LDA, each document's topic distribution $\boldsymbol{\theta}_d$ is drawn from a Dirichlet distribution with a fixed parameter $\boldsymbol{\alpha}$. In STM, the parameter depends on document covariates. If party and year are prevalence covariates, then a Conservative speech in 2019 has a different prior over topics than a Labour speech in 2017. The data informs which covariates make a difference and how much.

STM can also let topic content depend on covariates. Within the same topic, two parties might use different vocabulary. Conservatives discussing "the economy" might emphasize "growth," "competitiveness," and "enterprise"; Labour discussing the same topic might emphasize "inequality," "exploitation," and "fairness." Content covariates capture these within-topic differences.

4.3 Why STM is the default in political science

STM has become the standard topic model in political science research for several reasons.

First, it integrates the model with the analysis you actually want to do. The covariate effects on topic prevalence come with proper standard errors, so you can say "the migration topic was significantly more prevalent in AfD speeches than in CDU speeches, with effect size 0.12 (95% CI: 0.08–0.16)" rather than just eyeballing differences in topic proportions.

Second, it has better small-sample behavior than LDA. STM uses a "spectral initialization" that is deterministic given the same seed, which makes results reproducible and reduces the need to fit many models with different seeds.

Third, the `stm` package in R is excellent. It is well-documented, integrates seamlessly with `quanteda`, has built-in diagnostics for choosing K and interpreting topics, and produces high-quality plots out of the box.

Fourth, the political science community has converged on it. Most published papers using topic models in major political science journals over the past few years use STM. There's a critical mass of examples, tutorials, and best practices.

The practical recommendation for this course and beyond: if you're doing topic modeling in political science, start with STM unless you have a specific reason to use plain LDA.

4.4 A political example

Here's the kind of analysis STM enables. Suppose you fit an STM with $K = 30$ to UK Commons speeches from 2017–2019, with party and year as prevalence covariates, and party as a content covariate.

You might find that the "migration" topic is significantly more prevalent in Conservative speeches than in Labour speeches, with the gap larger in 2019 than in 2017. Within the migration topic, Conservatives use more words about "border" and "control," while Labour uses more words about "refugees" and "vulnerable." The "Brexit" topic is prevalent across all parties but discussed differently — Conservatives emphasize "deal" and "delivery"; Labour emphasizes "chaos" and "shambles."

All of this can be estimated and visualized with statistical uncertainty, directly from the STM output. You don't need to fit topics first and then run regressions — the model does it all together.

4.5 A brief note on BERTopic

The newest entrant to topic modeling is BERTopic (Grootendorst, 2022), which uses BERT embeddings instead of bag-of-words counts. The pipeline is: embed each document with a pre-trained BERT model, reduce the dimensionality with UMAP, cluster the embeddings with HDBSCAN, and use class-based TF-IDF to label each cluster.

BERTopic captures semantic similarity between documents (not just word overlap), works well on short texts like tweets, and is multilingual out of the box (because BERT models are

multilingual). It's a strong choice for some applications.

But it has weaknesses too: it's harder to validate than LDA/STM, it has no statistical foundation comparable to the LDA generative model, and the R support is minimal (it's primarily a Python library). For political science applications today, STM remains the safer choice. We'll mention BERTopic again briefly in Lecture 7, after we've covered BERT.

5 Part V — Limitations and the practical session

5.1 Limitations of topic models

Topic models are powerful, but they're not magic. Be aware of what they can and cannot do.

They are bag-of-words. LDA and STM ignore word order entirely. They cannot distinguish "the bill passed" from "the bill failed."

They are sensitive to preprocessing. Different stop word lists, stemming choices, and minimum frequency thresholds can change the topics substantially. Document your preprocessing carefully and ideally show that your main findings are robust.

They are sensitive to K. Different values give genuinely different topic structures. Report sensitivity to K in your analysis.

They have randomness. Different random seeds give different topics, though they should be qualitatively similar. Set a seed for reproducibility.

Interpretation is subjective. Two researchers might label the same topic differently. The labels are your contribution as a researcher; they are not "found" by the algorithm.

They struggle on short documents. Topic models need a decent number of words per document to estimate topic distributions reliably. Tweets are often too short. For short texts, consider sentence aggregation, BERTopic, or other methods.

Validation is essential and non-trivial. There is no objective ground truth, so validation requires careful design (face validity, word intrusion, predictive checks, robustness).

5.2 When topic models are a good fit

Topic models are a good fit when you want to explore a corpus you don't know well, discover latent themes without pre-specified categories, measure attention to issues over time or across actors, generate hypotheses for further study, and you have at least a few hundred medium-length documents.

They are a poor fit when you need to classify documents into specific pre-defined categories (use supervised methods instead — Lecture 2), when you need precise dictionary-style measurement of specific terms, when documents are very short (tweets — try BERTopic or simpler clustering), or when you need precise causal inference (topic models are too noisy for that).

The unifying point is: topic models are descriptive and exploratory, not measurement-grade. Use them to generate insights, then verify the most important findings with other methods.

5.3 The practical session

The practical session for this lecture fits a Structural Topic Model to UK House of Commons speeches and walks through the standard analysis workflow. The complete R code is provided in a separate script (`Lecture4_Practical.R`).

The workflow proceeds through several steps. First, we load the same UK Commons corpus from Lecture 1 and apply preprocessing — including stemming, which works well for topic models even though we preferred lemmatization in earlier lectures. Second, we convert the `quanteda` DFM to STM's required format using `convert(..., to = "stm")`. Third, we fit an STM with $K = 20$ topics, using party and year as prevalence covariates and spectral initialization for reproducibility. Fourth, we inspect the topics using `labelTopics()`, which reports four different metrics including FREX. Fifth, we manually label each topic based on its top words and representative documents. Sixth, we estimate covariate effects with `estimateEffect()` to see how topic prevalence varies by party and over time. Seventh, we run `searchK()` with several values of K to compare diagnostics and judge whether $K = 20$ was a reasonable choice. Throughout, we discuss the validation strategies and what each step is checking.

The discussion questions at the end of the practical focus on the things we most need to internalize: face validity (can you label every topic?), substantive interpretation (do covariate effects match prior expectations?), robustness (does changing K or the seed change the main findings?), and what topic models add over supervised approaches (what topics did you find that you wouldn't have pre-specified?).

Key takeaways

1. **Document representations** range from simple (bag of words, averaged embeddings) to sophisticated (Doc2Vec, BERT). For most political science applications, TF-IDF weighted averages of pre-trained embeddings are an excellent starting point.

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 both from the observed text.

4. **STM extends LDA** by letting document-level covariates affect topic prevalence and content. It has become the standard topic model in political science research.

5. **Choosing** K is not a solved problem. Use diagnostics (coherence and exclusivity), substantive knowledge, and interpretability to guide the choice. Always report sensitivity to K.

6. **Validation is essential.** Use face validity, FREX words, representative documents, word intrusion tests, predictive checks, and robustness analysis.

7. **Topic models are descriptive, not measurement-grade.** Use them for exploration and hypothesis generation; verify important findings with other methods.

Required reading

- Blei, D.M. (2012). "Probabilistic Topic Models." *Communications of the ACM*, 55(4), 77–84. (The most accessible introduction to LDA, by its creator.)

- Grimmer, J., Roberts, M. E., & Stewart, B.M. (2022). *Text as Data*, Chapters 12 to 13.

Recommended reading

- Grimmer, J. (2010). "A Bayesian Hierarchical Topic Model for Political Texts: Measuring Expressed Agendas in Senate Press Releases." *Political Analysis*, 18(1), 1–35.
- 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.

Other related reading

- Blei, D.M., Ng, A.Y., & Jordan, M.I. (2003). "Latent Dirichlet Allocation." *Journal of Machine Learning Research*, 3, 993–1022. (The original LDA paper, more technical.)
- Quinn, K.M., Monroe, B.L., Colaresi, M., Crespin, M.H., & Radev, D.R. (2010). "How to Analyze Political Attention with Minimal Assumptions and Costs." *American Journal of Political Science*, 54(1), 209–228.
- Roberts, M.E., Stewart, B.M., Tingley, D., Lucas, C., Leder-Luis, J., et al. (2014). "Structural Topic Models for Open-Ended Survey Responses." *American Journal of Political Science*, 58(4), 1064–1082.

Looking ahead

In **Lecture 5**, we move from count-based methods to *neural* methods. We'll cover the basics of feedforward neural networks for text, then introduce sequence models (RNNs and LSTMs) that can finally take word order into account. This is the conceptual bridge from classical methods to the deep learning revolution that has transformed NLP since 2017. By the end of Lecture 5, you'll have built and trained a simple neural text classifier, ready for the more sophisticated architectures in Lectures 6 and 7.