Transfer Learning & Fine-Tuning BERT
Multimodal Computational Methods in Political Science

Computational Social Science Program [1]

[1]LMU Munich — Computational Social Science MA Program

Summer Semester 2026

Course Roadmap: The Finish Line

1 Text as Data: Foundations & Preprocessing
2 Classical Text Classification
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 ← *Today*

Today is the culmination of the course. We bring together the Transformer architecture (Lecture 6), classification (Lecture 2), and the practical question every researcher faces: **how do I actually use a pre-trained language model on my political science task?** You'll fine-tune BERT on UK Commons speeches and compare it against everything we've built.

Today's Outline

Part I: Transfer Learning

- The pre-train / fine-tune paradigm
- Why it works so well

Part II: How BERT Was Pre-trained

- Masked Language Modeling
- Next Sentence Prediction
- What BERT actually learns

Part III: Fine-Tuning BERT

- The [CLS] token & classification head
- Training procedure & hyperparameters
- Common mistakes

Part IV: BERT in Practice

- Compute requirements
- When BERT beats LR (and when not)
- Validation

Part V: Beyond BERT

- RoBERTa, DeBERTa, multilingual variants
- NLI-based classifiers
- Large language models for classification

Part VI: Practical & Course Wrap-Up

Table of Contents

The Old Way: Training from Scratch

Until 2018, the typical NLP pipeline looked like this:

1. Start with a randomly-initialized neural network.
2. Collect labeled data for your specific task (e.g., 5,000 hand-coded political speeches).
3. Train the network on this labeled data until validation loss stops improving.
4. Apply the trained model.

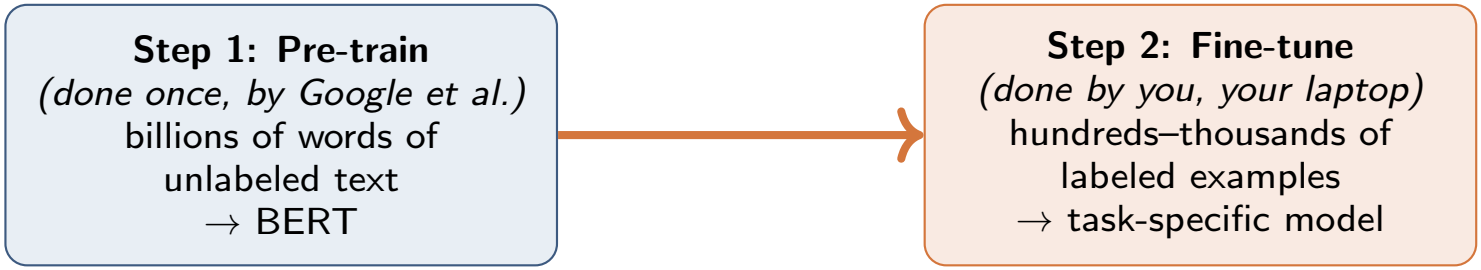
Why this is hard:

- The model has to learn *everything* from your labeled data: vocabulary, grammar, syntax, semantics — *and* your specific task.
- Neural networks need a lot of data. With 5,000 labels, they overfit easily.
- Logistic regression with TF-IDF (Lecture 2) often beats a from-scratch neural network at small to medium scale.

This is why Lecture 5's simple neural classifier barely beat Lecture 2's logistic regression.

The New Way: Pre-Train, then Fine-Tune

Since 2018, the paradigm has shifted to:



The crucial insight: the pre-training has *already taught the model about language*. Vocabulary, grammar, syntax, semantic relationships, factual knowledge — all of it is encoded in the pre-trained weights.

When you fine-tune, you're not teaching the model from scratch. You're **specializing** a model that already knows enormous amounts about language for your specific task.

This is the deepest reason why modern NLP works so well with so little labeled data. The model has done 99% of the learning before you ever see it.

Why Transfer Learning Works

Two complementary explanations:

1. Linguistic knowledge generalizes across tasks.
A model that knows what "increase" usually relates to (cost, taxes, support, momentum, etc.) doesn't need to relearn that knowledge for every new task. The contextual embeddings BERT produces are useful for sentiment analysis, topic classification, named entity recognition, and stance detection alike.

2. The model has seen the words you care about.
BERT was pre-trained on ∼3.3 billion words from Wikipedia + BookCorpus. Almost every word in your political corpus has been seen *thousands of times* in some context. The model has learned representations that capture word meaning, syntactic patterns, and semantic relationships from this massive exposure.

Key Concept

Fine-tuning is teaching a polyglot a new dialect, not teaching a baby a language. The hard part — learning language at all — has been done. You only need to specialize the model for your task.

The Economy of Transfer Learning

	Pre-training	**Fine-tuning**
Data needed	3+ billion words of unlabeled text	A few hundred to a few thousand labeled examples
Compute needed	Hundreds of GPUs for days / weeks	One GPU for minutes / hours
Done by	Big tech companies, well-resourced labs	Anyone with a laptop and Colab
Cost	Millions of dollars	Free (Colab) or pennies
Frequency	Done once per model	Done many times, per task

The democratization of NLP

You don't have to pre-train. Google, Facebook, Microsoft, and Anthropic have already done it, and they release the resulting models for free. You only do the fine-tuning step, which is computationally trivial.

This is what makes modern NLP accessible to political science researchers — a graduate student with a laptop can fine-tune a model that captures linguistic knowledge from billions of words.

Table of Contents

BERT: Bidirectional Encoder Representations from Transformers

BERT (Devlin, Chang, Lee & Toutanova, 2018) was a turning point in NLP.

The architecture: the Transformer encoder we built in Lecture 6.

- **BERT-base**: 12 layers, 12 attention heads, 768-dim embeddings, 110M parameters
- **BERT-large**: 24 layers, 16 attention heads, 1024-dim embeddings, 340M parameters

The innovation: a new pre-training task — **Masked Language Modeling** — that produced extraordinarily useful representations.

The result: BERT, fine-tuned on any standard NLP benchmark, set new state-of-the-art results across the board — often by a large margin. The paper became one of the most-cited in NLP history, and BERT remains the workhorse of computational text analysis in political science today.

We will fine-tune BERT in today's practical session.

Pre-training Task 1: Masked Language Modeling (MLM)

The idea: randomly hide some words in a sentence, and train the model to guess them from context.

Masked Language Modeling

Original: "The Prime Minister announced new climate measures"

Masked: "The Prime Minister [MASK] new climate [MASK]"

Model's job: predict the original words at the masked positions.

- Position 4: should be "announced" (or similar verb)
- Position 7: should be "measures" / "policies" / "targets"

Why this teaches BERT so much:

- To fill in "[MASK]" correctly, the model needs to understand the surrounding words.
- Because attention is bidirectional, the model can look at words *after* the mask, too.
- Doing this billions of times on diverse text teaches the model word meaning, syntactic structure, and even some world knowledge ("Prime Minister" tends to "announce" or "propose" rather than "cook" or "swim").

MLM: The Mechanics

The masking procedure (during pre-training):

- Select 15% of the tokens randomly.
- Of those: 80% are replaced with [MASK], 10% with a random word, 10% kept unchanged.

Why this weird 80/10/10 split?
It's a regularization trick. If we always used [MASK], the model would only learn good representations for [MASK] tokens (which never appear at fine-tuning time). The random and unchanged tokens force the model to maintain good representations for *every* token, regardless of whether it might be masked.

The training signal:
For each masked position, the model produces a probability distribution over the full vocabulary. The loss is cross-entropy against the true word — the same cross-entropy from Lecture 5, just applied per-position over a 30,000-word vocabulary instead of a few classes.

This is self-supervised learning. The labels come from the text itself — no human annotation needed. That's how you scale to billions of words.

Pre-training Task 2: Next Sentence Prediction (NSP)

The original BERT also had a second pre-training task: given two sentences, decide whether the second one follows the first in the original text.

NSP

Sentence A: "The bill passed in the House of Commons."
Sentence B: "The opposition vowed to challenge it in the Lords."
Label: IsNext (yes, this follows)
Sentence A: "The bill passed in the House of Commons."
Sentence B: "Tigers are large carnivorous mammals."
Label: NotNext

The motivation: NSP was meant to teach the model about discourse-level relationships — which is useful for question-answering and natural language inference.

Verdict, 5 years later: subsequent research (RoBERTa, ALBERT) showed NSP doesn't help much. Modern BERT variants drop it. But it's part of the original BERT and shapes how the model uses special tokens.

The two-sentence input format and the [CLS]/[SEP] tokens (which we'll see in a moment) come from NSP design.

The BERT Tokenizer and Special Tokens

BERT uses a special tokenizer: WordPiece. Rare words are split into subwords.

WordPiece tokenization

"The Bundestag passed legislation on globalization."

Tokenized: `[CLS], the, bun, ##des, ##tag, passed, legislation, on, global, ##ization, ., [SEP]`

The ## marks continuation of a previous word. Rare words become multi-token sequences. This way, the model handles *any* word, even ones never seen during training.

Two special tokens are always added:

- **[CLS]** at the start of every input. After processing, the [CLS] token's representation summarizes the entire sequence — used for classification.
- **[SEP]** at the end of each sentence. Used to separate sentence pairs (for NSP) and to mark the end of a single sentence.

BERT's input has a maximum length of **512 tokens**. Longer documents must be truncated or split. This will matter in the practical.

What Does BERT Learn?

Empirically, BERT's pre-training produces representations that encode:

- **Surface features** (lower layers): tokenization patterns, capitalization, basic word identity.
- **Syntactic features** (middle layers): part-of-speech, syntactic dependencies, phrase structure. BERT learns syntax *without ever seeing a parse tree*.
- **Semantic features** (upper layers): word sense disambiguation, semantic roles, coreference, entailment.
- **World knowledge** (distributed across layers): factual associations ("Paris" is the capital of "France"), social and cultural patterns.

Key Concept

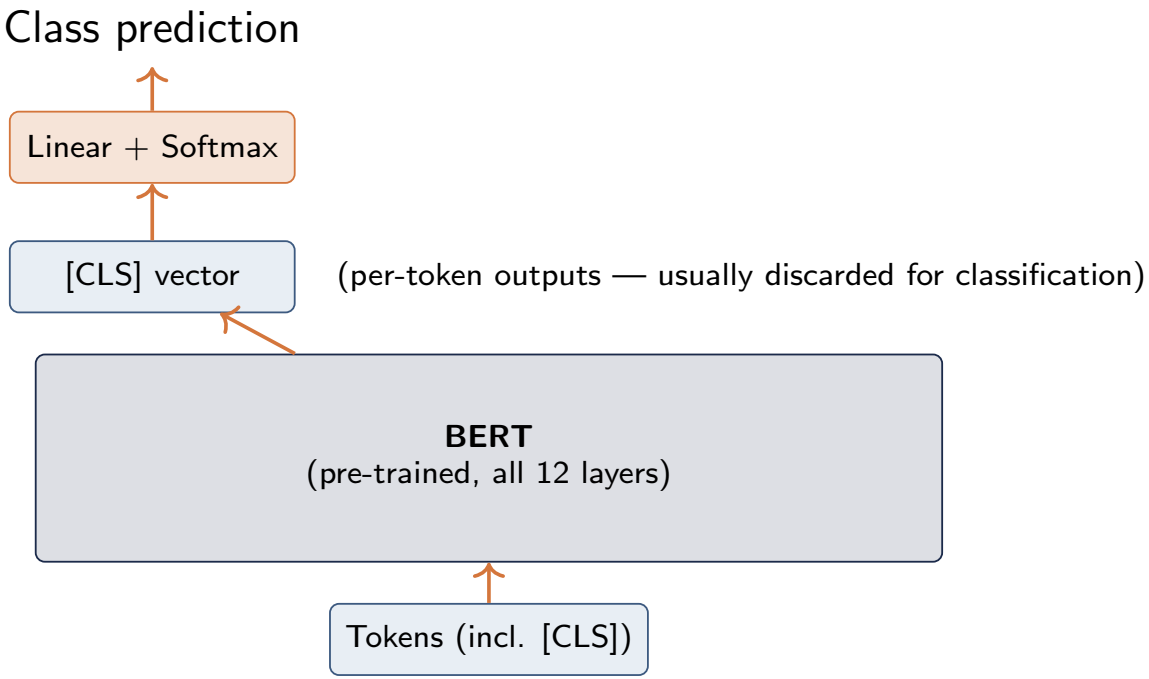
This means BERT's representations are useful for **many tasks, not just one**. Fine-tuning gives you "free" linguistic knowledge that you would otherwise have to learn from labeled data.

Caveat: BERT also encodes biases from its training data (gender, race, religion — see Caliskan-style WEAT applied to BERT). When you fine-tune, these biases come along for the ride.

Table of Contents

The Fine-Tuning Architecture

To fine-tune BERT for classification, we add a small "head" on top:



What's new vs. Lecture 6:

- Take BERT's output at the **[CLS] position** (a single 768-d vector summarizing the entire sequence).
- Add a small **classification head**: a linear layer + softmax over K classes.
- The head is randomly initialized; everything else is pre-trained.

This is the standard recipe. The classification head is tiny ($768 \times K + K$ parameters). BERT itself contributes 110M parameters of pre-trained knowledge.

The Fine-Tuning Procedure

The training loop is essentially the same as in Lecture 5, with two crucial differences:

1. **Start from pre-trained weights** (not random ones).
2. **Use a small learning rate** (typically 2×10^{-5} to 5×10^{-5}).

Why a small learning rate?
The pre-trained weights are already very good. Large updates would destroy that knowledge. We're gently adjusting BERT for our task, not retraining it from scratch.

Typical hyperparameters (from the original BERT paper, still works today):

- Learning rate: 2×10^{-5}, 3×10^{-5}, or 5×10^{-5}
- Batch size: 16 or 32
- Number of epochs: 2–4 (NOT 10–50 as in Lecture 5)
- Optimizer: AdamW with weight decay
- Warmup: linear learning-rate warmup over the first 10% of training, then linear decay

Just 2–4 epochs. BERT learns your task very quickly because most of the work is already done.

To Freeze or Not to Freeze?

Question: during fine-tuning, should the BERT weights themselves be updated, or only the classification head?

Full fine-tuning (the standard)
Update all of BERT's parameters along with the classification head.

Pros:

- Highest accuracy.
- Adapts BERT to your domain.

Cons:

- More compute (110M parameters updating).
- Risk of "catastrophic forgetting" if data is too small.

Frozen BERT + head
Treat BERT as a fixed feature extractor; only train the classification head.

Pros:

- Very fast.
- Tiny memory footprint.
- Robust with very small datasets.

Cons:

- Usually 2–5% lower accuracy.
- Can't adapt to specialized domains.

Default: full fine-tuning. Switch to frozen only if you have severe compute constraints or very little data.

Compromise: parameter-efficient methods (LoRA, adapters) update only a small fraction of weights. Good middle ground for limited compute.

Common Fine-Tuning Mistakes

Mistakes I see students (and researchers!) make:

1. **Learning rate too high.** A learning rate of 10^{-3} (good for from-scratch training) destroys BERT's pre-trained weights. Always use 10^{-5} range for fine-tuning.
2. **Training for too many epochs.** Past 4–5 epochs, BERT typically overfits hard. Use early stopping based on validation loss.
3. **Forgetting to evaluate on a held-out test set.** The fine-tuning loop reports validation metrics; you still need a separate test set you only touch once.
4. **Using the wrong tokenizer.** Each pre-trained model has its own tokenizer. `bert-base-uncased` requires its own tokenizer, not GPT-2's. Mixing them silently corrupts inputs.
5. **Ignoring the 512-token limit.** Long documents must be truncated or split into chunks. Pretending they fit produces silent errors.
6. **Comparing without baselines.** "My BERT got 87% accuracy!" is meaningless until you show that logistic regression on the same task got 84%.
7. **Not setting a random seed.** BERT fine-tuning is stochastic. Different seeds can give 2–3% accuracy differences. Run 3+ seeds and report the average.

Table of Contents

When BERT Beats Logistic Regression (and When It Doesn't)

The honest comparison, based on published political science studies:

BERT clearly beats LR when:

- Task requires understanding **context** or **negation** (stance detection, irony)
- Word order matters (sentiment with complex syntax)
- Documents are short (tweets, headlines) where TF-IDF has limited signal
- Classes are **semantically subtle** (different framings of the same topic)
- You have moderate labeled data (1,000–10,000 examples)

LR is competitive or better when:

- Task is mostly about **word frequency** (topic classification, party prediction)
- Documents are long (rich TF-IDF signal)
- Training data is very abundant (LR is already strong; BERT only marginal)
- Interpretability is essential (you need to know *which words* matter)

Always run a logistic regression baseline. If LR gets 85% and BERT gets 86%, the gain often isn't worth 100× the compute cost.

Computational Requirements: A Reality Check

What you actually need to fine-tune BERT-base:

Setup	Time per epoch (5k speeches)	Notes
CPU only (laptop)	1–3 hours	Workable but slow
Free Colab GPU (T4)	2–5 minutes	Highly recommended
Colab Pro (V100/A100)	30–60 seconds	Smooth iteration
Dedicated GPU	seconds	Research-scale work

Practical advice for your projects:

- Develop on Colab's free GPU. It's enough for the entire course.
- Save your fine-tuned model: `model.save_pretrained("./my_bert")`. Reload later instead of refitting.
- For BERT-large, you may need Colab Pro or a smaller batch size.
- Memory more often than time is the bottleneck. Reduce batch size if you run out.

Validating a Fine-Tuned BERT

The validation principles from Lecture 2 apply — with extra care.

1. **Held-out test set.** Set aside 10–20% of labeled data **at the start**. Do not touch it during model development. Evaluate exactly once at the end.
2. **Report all standard metrics:** accuracy, precision, recall, F1, confusion matrix. Per-class metrics for multi-class problems.
3. **Run multiple seeds.** BERT fine-tuning is stochastic. Report mean $\pm$ std over 3–5 seeds. A 1% difference between models with high variance across seeds is meaningless.
4. **Compare against baselines.** Always include LR with TF-IDF. Optionally include from-scratch neural net (Lecture 5). Without these, your BERT result is uncalibrated.
5. **Inspect errors.** Look at the documents BERT misclassifies. Is there a pattern? Length, language, ambiguity? Often informative.
6. **Probe for bias.** If your task could be socially sensitive (sentiment toward groups, hate speech, etc.), check whether BERT's errors are systematically worse for some groups than others.

Table of Contents

BERT Variants

BERT was the first widely-used encoder-only Transformer. Many improvements followed.

- **RoBERTa** (Facebook, 2019): same architecture as BERT, but trained on more data, longer, with optimized hyperparameters. NSP removed. Usually **1–2% better** than BERT on most tasks. Default starting point today.
- **DistilBERT**: smaller and faster (66M parameters), distilled from BERT. About 97% of BERT's accuracy at 40% the size. Good when compute matters.
- **DeBERTa** (Microsoft, 2020): introduces disentangled attention. Best single-model performance on many benchmarks. Slightly more expensive than BERT.
- **ALBERT**: parameter-sharing variant. Smaller memory footprint.
- **ELECTRA**: pre-trained with a different self-supervised objective (replaced-token detection). More compute-efficient pre-training; comparable fine-tuning performance.

Practical recommendation today: start with `roberta-base` for English tasks. Move to DeBERTa-v3 if you need state-of-the-art and have the compute.

Multilingual Models

For non-English text or cross-country comparisons:

- **Multilingual BERT (mBERT)**: BERT trained on 104 Wikipedia languages simultaneously. Cross-lingual transfer surprisingly good — you can fine-tune on English labels and apply to German text.
- **XLM-RoBERTa**: RoBERTa trained on 100 languages, much more data than mBERT. **Better than mBERT** on virtually every cross-lingual task. Default multilingual choice.
- **Language-specific BERTs**: GermanBERT, FlauBERT (French), AraBERT (Arabic), etc. Usually better than multilingual models if you only care about one language.

Cross-lingual political research

A common research design: fine-tune XLM-R on English UK parliamentary speeches with labeled stance, then **apply directly to French National Assembly debates** without any French labels. The model transfers what it learned.

This is genuinely transformative for European political science: it makes cross-national text analysis far cheaper than manual translation.

NLI-Based Classifiers (Zero-Shot, Almost)

An exciting recent approach for political science: NLI-based classifiers. See Laurer et al. (2024) in *Political Analysis*.

The idea: reformulate classification as Natural Language Inference (entailment). A pre-trained NLI model judges whether a hypothesis is entailed by a premise.

NLI-style classification

Premise (your document): "The minimum wage increase will burden small businesses and harm employment."

Hypothesis (your class definition): "This text expresses a conservative economic view."

NLI model's job: entails, contradicts, or neutral?

Translates back to: classified as "conservative" with high confidence.

The advantage: you can change the hypothesis without retraining. You define classes by *describing them in natural language*. Allows zero-shot or few-shot classification with surprisingly strong performance.

Laurer et al. show NLI-based classifiers often match fine-tuned BERT performance with *far fewer* labeled examples — a major win for political scientists with small labeled datasets.

Large Language Models (GPT-4, Claude, etc.)

LLMs can classify political text from prompts, with little or no training data.

Zero-shot classification with an LLM

Prompt:
"Classify the following UK parliamentary speech as expressing a Government or Opposition perspective. Speech: [text]. Answer in one word: Government or Opposition."

The LLM produces an answer directly — no fine-tuning needed.

When this works well:

- Conceptual tasks (the LLM "understands" what you're asking)
- When labeled data is unavailable or too expensive
- For exploratory research and pilot studies

When to be cautious:

- Cost per call adds up at scale (millions of API calls = thousands of $)
- Outputs are non-deterministic; reproducibility is harder
- Validation against hand-coded labels is *essential* — LLMs hallucinate confidently
- Privacy / GDPR concerns for sensitive text

Best practice: use LLMs as a fast first pass, then validate on hand-coded data, then fine-tune BERT if needed for production.

Recent Political Science Applications

Selected published studies using BERT-family models:

- **Widmann & Wich (2022)** in *Political Analysis*: detect emotion in German political text with fine-tuned BERT. Shows BERT clearly outperforms dictionaries and embeddings.
- **Bestvater & Monroe (2023)** in *Political Analysis*: sentiment vs. stance detection in social media. Argues that off-the-shelf sentiment classifiers conflate the two; calls for task-specific fine-tuning.
- **Laurer et al. (2024)** in *Political Analysis*: NLI-based classifiers as a transfer learning approach for political text. Strong results with very small labeled datasets.
- **Licht (2023)** in *Political Analysis*: cross-lingual transfer for ideological scaling using multilingual transformers.
- **Spirling & colleagues**: extensive work on validation, calibration, and pitfalls of BERT-based measurement in political science.

Common themes: BERT-family models offer real gains over classical methods for nuanced tasks — but only when researchers validate carefully and report honestly. The principles from Lecture 1 (Grimmer & Stewart) apply with full force.

Table of Contents

Practical: Fine-Tuning BERT on UK Commons

The Course Culmination

What we'll do: fine-tune BERT on the **same** Government vs. Opposition task from Lecture 2, then compare:

1. Logistic regression baseline (Lecture 2)
2. Simple neural classifier (Lecture 5)
3. Fine-tuned BERT (today)

Setup:

- Python + `transformers` + `datasets` + `evaluate`
- Google Colab (free GPU) strongly recommended
- Model: `bert-base-uncased` (110M params — runs on free Colab)
- Data: UK Commons speeches, 5,000 labeled by Gov/Opp

Expected runtime on Colab: $\sim$5 minutes per epoch, $\times$3 epochs $\approx$ 15 minutes total. See `Lecture7_Practical.py` (or the Colab notebook version) for complete code.

Practical: Discussion Questions

1. How does fine-tuned BERT compare with logistic regression (Lecture 2)? Is the gap meaningful?
2. Repeat the experiment with different random seeds (e.g., `seed=42, 123, 456`). How much do results vary? Does this change your interpretation?
3. Look at the documents BERT *misclassifies*. Are they hard cases? What features made them difficult?
4. Try a **harder** task: predict the specific party (Con / Lab / LibDem / SNP), not just Gov/Opp. Does BERT's advantage grow?
5. Try with very little data: 200 labels, then 500, then 1,000, then 5,000. How does BERT vs. LR change as labels grow? (Hint: BERT often wins more decisively in the low-data regime.)

The big-picture lesson: fine-tuned BERT is not always better than LR, but **when it's better, the gain is real and worth the cost**. Knowing when each tool wins is methodological maturity.

Table of Contents

What You Have Learned

Over seven lectures, you have built the complete toolkit of modern computational text analysis for political science:

- **Lecture 1:** preprocess any political text corpus, justify every choice, build a Document-Term Matrix.
- **Lecture 2:** train and properly evaluate classical classifiers; interpret their weights.
- **Lecture 3:** use word embeddings to measure semantic similarity and detect biases.
- **Lecture 4:** discover latent themes with topic models; estimate covariate effects with STM.
- **Lecture 5:** build neural networks for text; understand RNNs and LSTMs.
- **Lecture 6:** understand the Transformer architecture from the ground up.
- **Lecture 7:** fine-tune BERT on real political data; compare against everything that came before.

More importantly, you have learned the methodological principles that apply across all these methods: preprocessing is a modeling decision; classical methods are strong baselines; validation is essential; transparency about choices and limitations matters. **You can now read and replicate state-of-the-art political science research that uses text data.**

The Five Principles You Should Carry Forward

1. **Always try the classical methods first.** Logistic regression with TF-IDF is a strong baseline. If it solves your problem, you don't need BERT.
2. **Preprocessing is a modeling decision.** Document every choice. Show robustness to alternatives.
3. **Validate everything.** Hand-code a test set. Read the documents your model misclassifies. Report confidence intervals and seed variation.
4. **Computational methods augment humans, they do not replace them.** You still need substantive expertise to ask good questions and interpret answers.
5. **Be honest about limitations.** A clear, honest analysis with modest claims is better than an inflated one with hidden caveats. The reviewers will thank you. So will future-you, reading your old papers.

These principles are the methodology. The techniques (BERT, LDA, etc.) will be replaced; the principles won't.

What Comes Next

This course gives you foundations. Here is the path forward.

Immediate next steps:

- Apply these methods in your thesis or research project.
- Replicate a published computational text analysis paper in political science (excellent learning exercise).
- Read recent papers in *Political Analysis*, *Political Communication*, and the workshops at NLP+CSS.

Topics this course did NOT cover (worth learning later):

- Generative LLMs in depth (GPT-4, Claude, prompt engineering, RAG)
- Causal inference with text (text as treatment or outcome)
- Topic models beyond LDA/STM (dynamic topic models, hierarchical models)
- Network methods on text data (citation networks, co-mention graphs)

The field is moving fast. The methods you learn next year will partly replace what you learned this year. But the principles from this course will not.

Final Words

Thank you for taking this course.

Text analysis is one of the most exciting and consequential methodological developments in contemporary political science. The questions that can now be answered with text data — about democratic discourse, polarization, framing, representation, accountability, misinformation — are questions that matter for the world we live in.

You have the tools. Use them carefully. Use them transparently. Use them in service of good research questions.

Good luck with your final projects, your theses, and your research careers.

For questions about your final project: office hours through the end of the semester. Don't hesitate.

Readings

Required:

- Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding." *NAACL 2019*. https://arxiv.org/abs/1810.04805
- Hugging Face NLP Course, Chapter 3: Fine-tuning a pretrained model. https://huggingface.co/learn/nlp-course/chapter3

Recommended (political science applications):

- Widmann, T. & Wich, M. (2022). "Creating and Comparing Dictionary, Word Embedding, and Transformer-Based Models to Measure Discrete Emotions in German Political Text." *Political Analysis*.
- Laurer, M., van Atteveldt, W., Casas, A., & Welbers, K. (2024). "Less Annotating, More Classifying: Addressing the Data Scarcity Issue of Supervised Machine Learning with Deep Transfer Learning and BERT-NLI." *Political Analysis*.
- Bestvater, S.E. & Monroe, B.L. (2023). "Sentiment Is Not Stance: Target-Aware Opinion Classification for Political Text Analysis." *Political Analysis*.

End of course. Good luck on your final projects.