Recap: Natural Language Processing

DS701 Session 21 — Mon Nov 16, 2026

Today’s plan

Knowledge-Check Review

KC 1: The TF-IDF weight

Write down the three pieces of the TF-IDF weight of a term t in a document d — term frequency, inverse document frequency, and how they combine — and say in one sentence what kind of term ends up with a large weight. What weight does a term that appears in every document of the corpus get, and why is that the point?

KC 2: Why sparse vectors cannot see synonyms

The lecture notes that with one-hot encodings the similarity of two different words is always 0, while word embeddings put words with similar meanings close together (small angle, high cosine). Explain why the sparse representations — one-hot, bag of words, TF-IDF — can never express that "doctor" and "physician" are related, and what an embedding does differently that makes it possible.

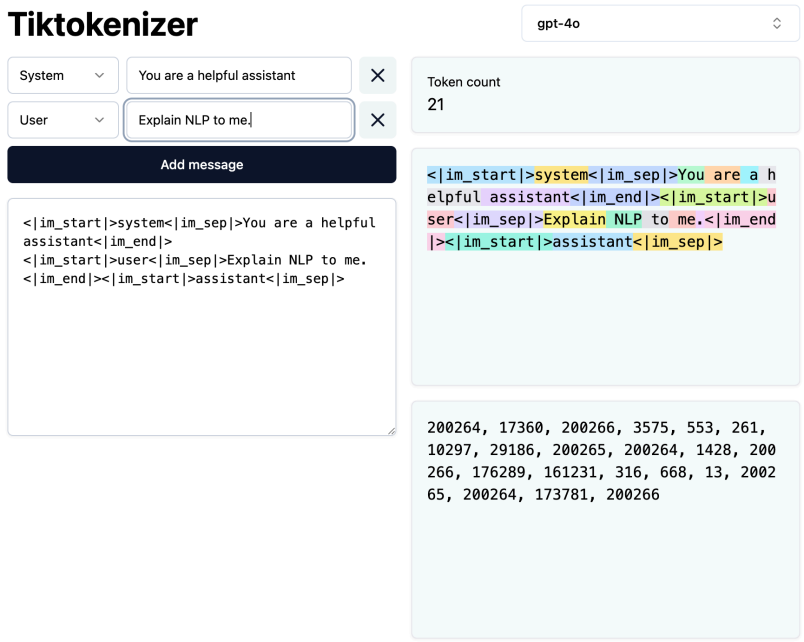
KC 3: Why “bank” gets two vectors

A static embedding (Word2Vec) gives “bank” one vector; a contextual model (BERT) gives “bank” a different vector in “river bank” and in “bank account”. Using the lecture’s description of attention (queries, keys, values, softmax-weighted sum), explain what mechanism lets the second vector depend on the neighbouring words. Then name one task from the lecture where that difference matters and one where a static embedding is good enough.

Highlights

Text → tokens → vectors is the whole game

Models cannot read strings. Every NLP system, from a 1990s search engine to GPT, starts the same way:



Bag of words — and why counts alone mislead

Throw away order and grammar; keep **how many times** each vocabulary word occurs. A document becomes a vector in $\mathbb{R}^V$:

	the	cat	sat	on	mat	dog	log	emu
“The cat sat on the mat.”	**2**	1	1	1	1	0	0	0
“The dog sat on the log.”	**2**	0	1	1	0	1	1	0
“The emu sat on the mat.”	**2**	0	1	1	1	0	0	1

TF-IDF: the formula and what it rewards

$$\text{tf-idf}(t, d) = \underbrace{\text{tf}(t, d)}_{\text{frequent here}} \times \underbrace{\log \frac{N}{\text{df}(t)}}_{\text{rare overall}} \qquad \text{df}(t) = \#\{\text{documents containing } t$$

TF-IDF in practice: what scikit-learn actually computes

`TfidfVectorizer` is **not** the textbook formula. Its defaults:

$$\text{idf}_{\text{sk}}(t) = \log \frac{1 + N}{1 + \text{df}(t)} + 1, \qquad \text{then each document row is divided by its } \ell$$

Document similarity via cosine

$$\cos\theta = \frac{v_1 \cdot v_2}{\|v_1\|\,\|v_2\|}$$

Embeddings: from sparse to dense

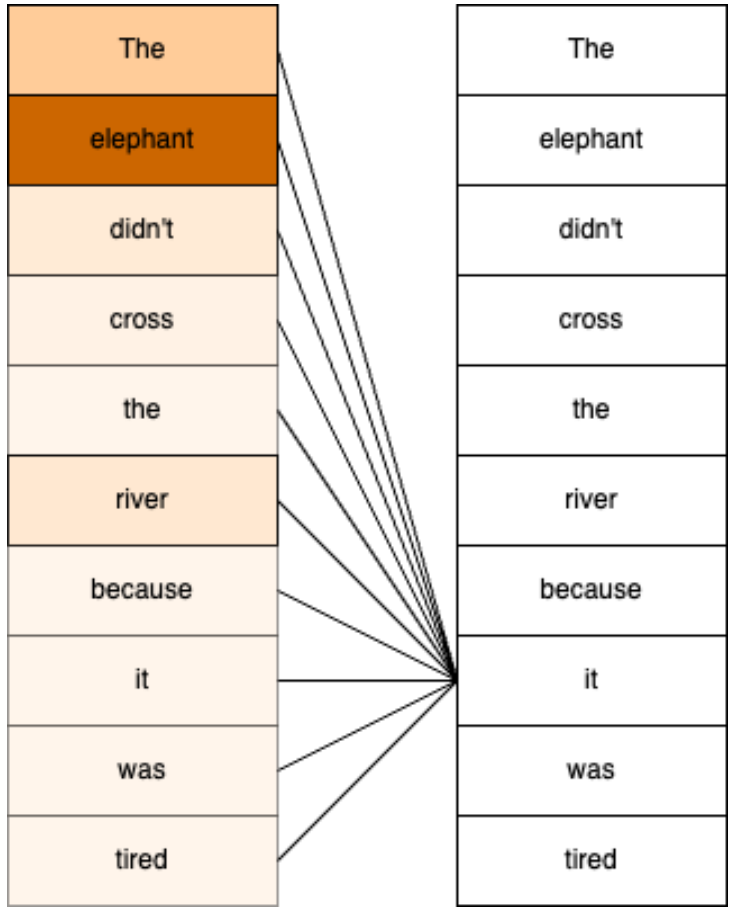
Sparse (one-hot, BoW, TF-IDF): one axis per word, all axes orthogonal, similarity = literal overlap. Vocabulary-sized and mostly zeros.

Dense embedding: a learned vector of a few hundred dimensions per word; *similar meaning* $\Rightarrow$ *small angle*. Individual coordinates are not interpretable; the geometry is (**king − man + woman ≈ queen**).

What attention adds — in one slide

> “The elephant didn’t cross the river because **it** was tired.”

For each token, in parallel over the whole sequence:



The toolkits — pointers, not the point

Need	Reach for	Where
tokenize, count, TF-IDF, cosine, LSA	`sklearn.feature_extraction.text`, `TruncatedSVD`	**today's activity**
classical text processing, corpora, sentiment lexicons	**NLTK**	A5 — NLP Packages
production pipelines: POS, parsing, **NER**, `displacy`	**spaCy**	A5, lecture NER section
topic modeling with transformer embeddings (embed → UMAP → HDBSCAN → c-TF-IDF)	**BERTopic**	A5, lecture
tokenizers and pre-trained transformer models	Hugging Face `transformers`	lecture
applications on top of	LangChain	A5

In-Class Activity

Activity: text to vectors — TF-IDF, similarity, and what embeddings add

Goal: build the text pipeline **by hand** — tokenizer, bag-of-words matrix, TF, IDF, TF-IDF — checking each against `CountVectorizer` / `TfidfVectorizer`; then use cosine similarity to retrieve the nearest document. **Stretch:** measure retrieval accuracy on real newsgroup posts, hit TF-IDF's blind spot (a same-meaning pair scoring exactly 0), and close it with LSA — `TruncatedSVD` on the TF-IDF matrix — before asking what a transformer embedding would add.

- Work in groups of 2–3. Open **your section's** notebook.
- Parts marked **(autograded)** are submitted to Gradescope; the rest is participation.
- Staff will circulate — be ready to explain any part of your work.
- Dependencies: `numpy`, `pandas`, `matplotlib`, `scikit-learn` — no spaCy / NLTK / transformer downloads (the one optional cell that uses them is clearly marked).

Section A1 

Going deeper

Full lecture notes: Natural Language Processing

- Tokenization and Tokens, Token IDs, and Vocabulary — character, word, subword; the Hugging Face tokenizer demo
- Sparse Representations — one-hot, bag of words, the TF-IDF table
- Word Embeddings, Word2Vec, and Contextual vs Static
- The Attention Mechanism and Queries, Keys, and Values; 3 Types of Transformer Models
- Named Entity Recognition with spaCy; Topic Modeling with BERTopic
- The toolkits, with runnable examples: NLP Packages
- The SVD behind LSA: SVD and Low-Rank Approximation and Dimensionality Reduction