Text as Data II: *NLP Basics*

DSIER [/d□ˈza□ər/] — Summer 2025

Irene Iodice

Bielefeld University

Learning goals for today

- Move beyond bag-of-words counts.
- Describe classic representations (n-grams, topic models, Word2Vec, GloVe, fastText).
- Interpret the skip-gram (Word2Vec) objective and its use of positive/negative sampling.
- Distinguish static vs contextual embeddings and outline Transformer self-attention.
- Compute and benchmark document similarity with modern embeddings.
- Applications:
  - AIPNET
  - Innovation clusters

Section 1

Why move beyond simple word counts?

Limitations of the bag-of-words (BoW) approach

- **Ignores order**: “bank raises rates” vs “rates raise bank”.
- **Misses synonyms**: “job”, “employment”, and “work” are treated as unrelated.
- **Very sparse**: thousands of columns, mostly zeros, make models hard to train.
- Newer methods turn each word (or document) into a short, dense vector that captures meaning.

Limitations of the bag-of-words (BoW) approach

- **Ignores order**: “bank raises rates” vs “rates raise bank”.
- **Misses synonyms**: “job”, “employment”, and “work” are treated as unrelated.
- **Very sparse**: thousands of columns, mostly zeros, make models hard to train.
- Newer methods turn each word (or document) into a short, dense vector that captures meaning.

Why economists care

Better text features often sharpen forecasts (e.g. central-bank tone $\rightarrow$ bond yields) and let us study questions about narrative change.

Example uses in economics

- **Forecasting**: Use news or earnings-call transcripts to predict GDP or inflation.
- **Policy uncertainty**: Compare dictionary vs embedding-based indices (Baker et al., 2016).
- **Sentiment**: Track investor mood in filings or tweets (Ke et al., 2019).
- **Skills**: Match job ads to O*NET tasks with embeddings (Hansen et al., 2021).

Why text matters for economists

- Digital text — news, patents, job ads — is exploding; numbers alone miss these “soft” signals (Gentzkow et al., 2019).
- NLP turns messy words into *variables* we can graph, regress, and test.
- Early tools counted words but ignored context.
- Modern models capture meaning and nuance → richer measures of innovation, policy tone, skills, and more.

From word counts to large language models

- **Then**: BoW ≈ word counts.
- **Now**: transformer models such as BERT or GPT weigh each word in its context
  : transformer models such as BERT or GPT weigh each word in its context.
  ”attention operations that allow models to weigh the importance of different words in a context-dependent manner (Vaswani et al. 2017)”
- Domain-tuned versions (FinBERT, ClimateBERT) already beat generic models in finance and climate text.
- ChatGPT-style assistants make advanced NLP almost plug-and-play.

Section 2

Representations

Step 1 – Convert documents into tokens

- Break text into words (or sub-words).
- Build a vocabulary that maps each token to an ID.
- Basic cleaning: lower-case, remove stop-words, maybe stem.
- The result can be stored in a Document-Term Matrix (DTM).

Bag-of-Words Reload

- Count how often each word appears; word order is ignored.
- Simple and transparent — that's why many early economics papers used it.
- Downsides:
  - Doesn't know that "weak" and "tepid" are similar.
  - Can't tell "statistics lie" from "cats lie".
  - No sense of grammar or context.

Making BoW a bit better

1. **Topic models**: group words that often appear together.
2. **n-grams**: keep short phrases like “weak growth”.
3. **Dependency parsing**: pull out subject–verb–object triples.

Making BoW a bit better

1. **Topic models**: group words that often appear together.
2. **n-grams**: keep short phrases like “weak growth”.
3. **Dependency parsing**: pull out subject–verb–object triples.

Still, these tricks stay *sparse*, and they miss deeper context.

Example: why order and meaning matter

Synonomy: economic growth is weak but long-term productivity trends are strong.
economic growth is tepid but long-term productivity trends are strong.

Polysemy: one word having multiple meanings.
economic statistics lie about current well-being.
the happy cats lie on the bed.

Word Order:
economic growth is weak but long-term productivity trends are strong.
economic growth is strong but long-term productivity trends are weak.

So what?

- Older text methods gave useful summaries.
- But they missed meaning, nuance, and grammar.
- That set a ceiling on predictive power.
- Newer models break that ceiling by using context.

Subsection 1

N-grams & dimension reduction

From single words to short phrases

- **Unigrams**: single words.
- **Bigrams / trigrams**: keep 2- or 3-word phrases like “capital gains tax”.
- Adds context but also more columns (sparser data).
- Trade-off: more context vs heavier data.

From single words to short phrases

- **Unigrams**: single words.
- **Bigrams / trigrams**: keep 2- or 3-word phrases like “capital gains tax”.
- Adds context but also more columns (sparser data).
- Trade-off: more context vs heavier data.

R example – get bigrams

```
library(tidytext)
reuters |>
  unnest_tokens(bigram, text, token = "ngrams", n = 2) |>
  count(bigram, sort = TRUE)
                bigram    n
1               of the  439
2                to be  422
```

Classic *n*-gram Language Models

Language Models: “given everything I’ve already read, what word (or token) is most likely to come next?”

- **Markov assumption (order $n-1$):** $P(w_t \mid w_{t-n+1}^{t-1})$ depends only on the previous $n-1$ words.
- **Relative-frequency estimation:** $P(\text{rates} \mid \text{raises}) = \dfrac{\#(\text{‘raises rates’})}{\#(\text{‘raises’})}$.
- Sparsity explodes as n or vocabulary V grows ($|V|^n$ possible sequences).
- **Smoothing (Kneser–Ney)** redistributes probability mass to unseen n-grams $\Rightarrow$ avoids zero-probability phrases.
- **Economics use-case:** next-word perplexity spikes can flag anomalous financial filings or authorship shifts.

Subsection 2

Word embeddings

Word Embeddings: A Semantic Map

- Word embeddings are like coordinates on a semantic map.
- Instead of a giant lookup table with thousands of orthogonal features, we use a dense vector space (typically 100–300 dimensions).
- In this space, similar words are located close to each other.

Example:

$$\cos(\vec{\rho}_{\text{growth}}, \vec{\rho}_{\text{expansion}}) \approx 0.9 \quad \text{(very similar)}$$
$$\cos(\vec{\rho}_{\text{growth}}, \vec{\rho}_{\text{inflation}}) \approx 0.3 \quad \text{(partially related)}$$

- Cosine similarity captures analogies: $\text{king} - \text{man} + \text{woman} \approx \text{queen}$.

Distributional semantics – “You shall know a word by the company it keeps.”

- **Word2Vec** (Mikolov et al. 2013): CBOW & Skip-Gram architectures.
- **GloVe** (Pennington et al. 2014): factorizes global co-occurrence matrix.
- fastText: extension of Word2Vec, each word as a bag of character n-grams

Distributional semantics – “You shall know a word by the company it keeps.”

- **Word2Vec** (Mikolov et al. 2013): CBOW & Skip-Gram architectures.
- **GloVe** (Pennington et al. 2014): factorizes global co-occurrence matrix.
- fastText: extension of Word2Vec, each word as a bag of character n-grams

Training embeddings in R (`text2vec`)

```
library(text2vec)
iter <- itoken(text_corpus, tokenizer = word_tokenizer)
vectorizer <- vocab_vectorizer(create_vocabulary(iter, ngram = c(1,1)))

tcm <- create_tcm(iter, vectorizer, skip_grams_window = 5)
fit <- GlobalVectors$new(rank = 200, x_max = 10)
word_vec <- fit$fit_transform(tcm)
```

Word embeddings: the basic idea

- A word embedding is a vector $\rho_v \in \mathbb{R}^K$ for vocabulary term v.
- Usually $K \ll V$ (vocabulary size) → low-dimensional but expressive (Dense, low-dim vectors help models learn patterns like: “jobs” and “employment” often appear in similar contexts.
- Design goal: $\cos(\rho_v, \rho_{v'})$ is large $\rightarrow$ words v and v' share meaning.

Word embeddings: the basic idea

- A word embedding is a vector $\rho_v \in \mathbb{R}^K$ for vocabulary term v.
- Usually $K \ll V$ (vocabulary size) → low-dimensional but expressive (Dense, low-dim vectors help models learn patterns like: “jobs” and “employment” often appear in similar contexts.
- Design goal: $\cos(\rho_v, \rho_{v'})$ is large $\rightarrow$ words v and v' share meaning.

Key takeaway

We embed words in a continuous space where *distance ≈ semantic dissimilarity*.

“You shall know a word by the company it keeps”

- Linguistic principle behind Word2Vec.
- Define a context window of length $2L$ around each token.

$$C(w_{d,t}) \;=\; [\, w_{d,t-L}, \ldots, w_{d,t-1}, w_{d,t+1}, \ldots, w_{d,t+L} \,]$$

- The model learns vectors so that **target words** predict **context words**.

Word2Vec: skip-gram objective

- Task: given target word w, decide if comparison word w' is in its context.
- Positive pairs: (w, w') with $w' \in C(w)$ Negative pairs: (w, w') with random w'.
- Parameters:
  - ρ_w – target embedding
  - $\gamma_{w'}$ – context embedding

$$\Pr\left[w' \in C(w)\right] = \frac{\exp(\rho_w^\top \gamma_{w'})}{1 + \exp(\rho_w^\top \gamma_{w'})}$$

Word2Vec: Predicting Context Words

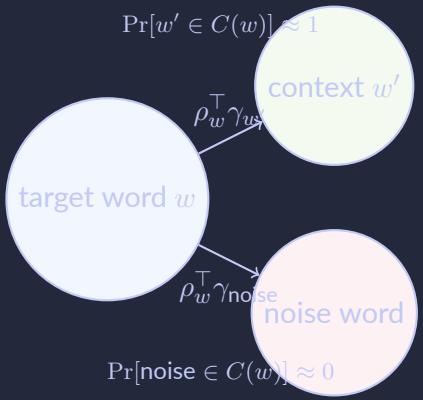
Probability model:

$$\Pr[w' \in C(w)] = \frac{\exp(\rho_w^\top \gamma_{w'})}{1 + \exp(\rho_w^\top \gamma_{w'})} = \sigma(\rho_w^\top \gamma_{w'})$$

- ρ_w: embedding of target word w
- $\gamma_{w'}$: embedding of context word w'
- $\rho_w^\top \gamma_{w'}$: similarity (dot product)
- Sigmoid $\sigma(x)$ squashes this into [0,1]

Goal:

- High score for true context words
- Low score for random (negative) words



Training Word2Vec embeddings: workflow

Goal: learn ρ and γ so real neighbours score high, random pairs score low.

1. Initialise vectors with small random values.
2. For each sentence:
   - Pick centre word w and window $C(w)$ (± 2).
   - Positive pairs: (w, w'), $w' \in C(w)$
   - Negative pairs: (w, w'), w' random
3. Objective (y=1 for positive pairs and 0 for negative pairs)

$$\ell = \sum_{(w,w')} y \log \sigma(\rho_w^\top \gamma_{w'}) + (1-y) \log\left[1 - \sigma(\rho_w^\top \gamma_{w'})\right]$$

4. SGD update: nudge $\rho_w, \gamma_{w'}$ along the gradient; repeat for many epochs.

Intuition

Pull vectors together when words co-occur; push apart otherwise.

Outcome

Words that share contexts end up near each other in K-D space $\Rightarrow$ embeddings capture meaning.

A 2-D toy example of ρ and γ

Tiny vocabulary: $\{\text{king}, \text{queen}, \text{man}, \text{woman}\}$, $K = 2$

$$\rho_{\text{king}} = [0.8, 0.1], \quad \gamma_{\text{man}} = [0.95, -0.1]$$

$$\Pr(\text{“man” near “king”}) = \frac{\exp(0.8 \times 0.95 + 0.1 \times -0.1)}{1 + \exp(0.76)} \approx 0.68$$

68% chance that “man” is predicted to appear near “king”.

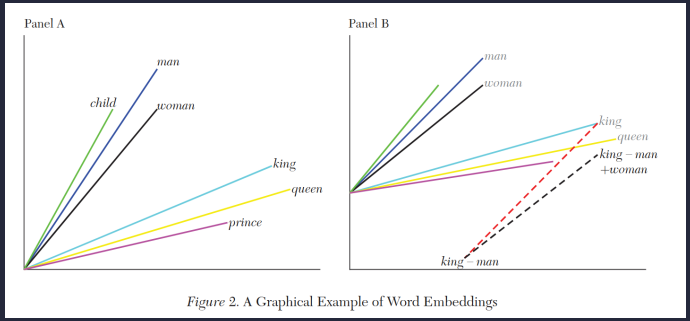


Figure 2. A Graphical Example of Word Embeddings

Practical tips for using embeddings

1. **Corpus matters**: train on domain text (e.g., SEC 10-K, central-bank speeches) to capture jargon and semantics.
2. **Dimension K**: 100–300 is common; larger adds nuance but risks overfitting and slows downstream models.
3. **Window size L**: smaller L emphasizes syntax; larger l emphasises topical similarity.
4. **Debias or diagnose**: always explore projection directions for gender, race, geography before causal inference.
5. **Downstream validation**: treat embeddings as features—benchmark on prediction or similarity tasks tied to your economic question.

How GloVe learns embeddings

Training objective:

$$\min_{\rho_i, \gamma_j} \sum_{i,j} f(W_{i,j}) \left(\rho_i^\top \gamma_j - \log W_{i,j}\right)^2$$

- Goal: match dot products of embeddings to log-counts of word co-occurrence.
- $f(W_{i,j})$ is a weighting function to downweight rare pairs:

$$f(x) = \begin{cases} (x/x_{\max})^\alpha & \text{if } x < x_{\max} \\ 1 & \text{otherwise} \end{cases}$$

- This avoids overfitting to rare word pairs and stabilizes learning.

Comparison

Comparison to Word2Vec

GloVe uses global co-occurrence counts
Word2Vec predicts local context word-by-word.

Feature	Word2Vec	GloVe
Uses...	Local context windows	Global co-occurrence counts
Learns via...	Prediction	Matrix factorization
Goal	Predict context	Reconstruct co-occurrence matrix

Summary Comparison of Word2Vec and GloVe

R example – get bigrams

```r
# 1. Load the text2vec package ---------------------------------
library(text2vec)         # provides tokenisers, vectorisers, and GloVe

# 2. Turn the raw text into an iterator of tokens -------------
#    - text_corpus : character vector (one element = one document)
#    - word_tokenizer : splits each document into a vector of words
iter <- itoken(text_corpus,
               tokenizer = word_tokenizer,
               progressbar = FALSE)   # iterator yields one tokenised doc at a time

# 3. Build a vocabulary and a vectoriser ----------------------
#    - create_vocabulary() scans the iterator and collects unique tokens
#    - here ngram = c(1,1) means “use unigrams only”
vocab <- create_vocabulary(iter, ngram = c(1, 1))
vectorizer <- vocab_vectorizer(vocab)  # converts the vocab into a “vectoriser” object

# 4. Construct a Term-Co-occurrence Matrix (TCM) --------------
#    - skip_grams_window = 5   → look 5 words to the left & right of each token
#    - the TCM counts how often two words co-occur within that window
iter <- itoken(text_corpus, tokenizer = word_tokenizer, progressbar = FALSE)  # re-initialis
tcm  <- create_tcm(iter, vectorizer, skip_grams_window = 5)
```

R example GLOVE

```r
# 5. Train a GloVe model (Global Vectors for Word Representation)
#    - rank = 200         → dimensionality of the word vectors
#    - x_max = 10         → controls weighting of frequent / infrequent co-occurrences
fit <- GlobalVectors$new(rank = 200, x_max = 10)

# 6. Fit the model and get the word embeddings ----------------
#    - fit_transform() learns two matrices: main (ρ) and context (γ) embeddings
#    - it returns the main matrix (ρ);
word_vec <- fit$fit_transform(tcm)     # rows = words, cols = 200-dim embeddings

# word_vec is now a matrix you can:
#   • average into sentence / document vectors
#   • compute cosine similarity between words
#   • feed into downstream ML tasks
```

Limitations of Static Embeddings

- **One vector per word (no context awareness)**
  - Cannot handle *polysemy* — words with multiple meanings.
  - **Example:**
    - "He sat by the **bank** of the river."
    - "She deposited money in the **bank**."
  - Word2Vec/GloVe assigns the *same vector* to both, despite different meanings.

- **Ignores word order (sentence structure lost)**
  - Sentence embeddings often average word vectors → structure is erased.
  - **Example:**

  Economic growth is weak, but productivity is strong.
  Economic growth is strong, but productivity is weak.

  - These sentences mean opposite things, but averaging static embeddings gives similar vectors.

- **Motivation for contextual embeddings**
  - different vectors for the same word depending on context
  - They preserve both **meaning** and **sentence structure**.

Subsection 3

Contextual embeddings

From static to contextual embeddings

- **Static** (Word2Vec, GloVe): one vector ρ_v per word type — the same in every sentence.
- **Contextual** (BERT, RoBERTa, GPT): a fresh vector $\rho_{d,n}$ for each token position *after* seeing its neighbours.
- Limitation solved: polysemy and word–order sensitivity now captured.

Self-attention update (single layer)

$$\rho'_{d,n} = \sum_{u=1}^{N_d} w_{n,u}\, \rho^0_{d,u}, \qquad \sum_u w_{n,u} = 1$$

$w_{n,u}$ are learned relevance weights; deeper Transformers stack many such layers.

Understanding Self-Attention Indices

$$\rho'_{d,n} = \sum_{u=1}^{N_d} w_{n,u} \cdot \rho^0_{d,u}$$

What the indices mean:

- $\mathbf{d}$: index of the input sequence (e.g. sentence or document)
- $\mathbf{N_d}$: number of tokens in sequence d
- $\mathbf{n}$: position of the *current* token we are updating
- $\mathbf{u}$: position of a *context* token being attended to

Example sentence ($d = 1$):

"NVIDIA's share price hit an all-time high."

- $N_1 = 7$ tokens
- To compute updated vector for token $n = 7$ ("high"):
  Sum over all tokens $u = 1, \ldots, 7$, weighted by $w_{7,u}$

Interpretation: Each token's new representation is a context-aware combination of the entire sequence.

Two Core Prediction Tasks in Transformers

1. Autoregressive (Next-Word Prediction)

$$\Pr[w_{N+1} \mid w_1, w_2, \ldots, w_N]$$

- Model predicts the next word given all previous words.
- Used in **GPT**, great for text generation.
- *Example:* "The cat sat on the __" → likely: "mat".

2. Bidirectional (Masked Language Modeling)

$$\Pr[w_t \mid \text{all other words except } w_t]$$

- Model fills in a missing word using both left and right context.
- Used in **BERT**, good for understanding full context.
- *Example:* "NVIDIA's share price hit an all-time `[MASK]`."
  - Likely guess: "high" or "record".

How Self-Attention Works (Intuition)

Goal: Each word gets a new vector that reflects the full sentence context.

Step 1: Score how much attention each word pays to every other word

$$e_{n,u} = \frac{(Q\rho_{d,n})^\top (K\rho_{d,u})}{\sqrt{K}}$$

- Q, K: learned matrices for comparing word meanings.

Step 2: Turn scores into weights using softmax

$$w_{n,u} = \frac{\exp(e_{n,u})}{\sum_{u'} \exp(e_{n,u'})}$$

- Higher $w_{n,u}$ = more attention from token n to token u.

Step 3: Compute the new vector for token n

$$\rho'_{d,n} = \sum_u w_{n,u} \cdot (V\rho_{d,u})$$

- V: another learned matrix for the value transformation.

Why It Works

- Captures long-distance links (e.g., “he” ↔ “runs”).
- Ignores unimportant words (e.g., “the”, “and”).
- Resulting vectors encode rich context and syntax.

Mini Example

“He **runs** fast” vs. “He **runs** a company” → Different meanings for “runs” from context!

Which words get the highest attention?

Sentence:

*"...driven by **AI** adoption, **NVIDIA**'s share price hit an all-time [MASK]."*

Token	Attention $w_{\texttt{[MASK]},u}$	Intuition
NVIDIA	0.40	issuer name highly predictive
AI	0.35	theme of innovation
driven	0.10	verb context
by	0.05	function word
a/of...	< 0.05	low-information stopwords

Take-home: attention distributes probability mass to the truly informative tokens.

Why context matters in economics text

- Static vectors blur nuances: “rates **rise**” = “**rise** rates”.
- Contextual vectors separate monetary vs river “bank”; hawkish vs dovish “tone”.
- Empirical gains: finer sentiment, better similarity, sharper policy-shock identification.

Bottom line: attention-based embeddings power today’s LLMs (BERT, GPT) — and their economic applications.

Subsection 4

Generative language models

Transformer-based LLMs — Key Ideas

- Modern models (BERT, GPT, Claude, Gemini...) are built on the **Transformer** architecture.
- Train on massive text corpora (e.g. Wikipedia, Common Crawl) with billions/trillions of tokens.
- Input text is split into sequences of up to N tokens (e.g. $N = 512$ for BERT, $N = 128{,}000$ for GPT-4).
- Each token gets an initial embedding h_t^0 that encodes:
  - the identity of the word/token,
  - its **position** in the sequence.
- Stack L layers of self-attention and feed-forward transformations:

$$h_t^1 = \text{NonLinear}\left(\sum_u \alpha_{t,u} h_u^0\right), \quad \ldots, \quad h_t^L$$

BERT vs GPT — Masked vs Generative LLMs

BERT (2018)

- **Bidirectional encoder**: predicts [MASK] in middle of text.
- 15% of tokens are masked during training.
- Trained on Wikipia + books (3.3B words).
- Good for understanding, classification, etc.

GPT (2018–2023+)

- **Autoregressive decoder**: predicts next token only.
- No masking — always moves forward.
- Trained on web-scale corpora.
- Good for generation, reasoning, conversation.

Key difference: BERT sees both past + future; GPT only sees the past.

How Do We Adapt Pretrained LLMs?

- Big models like BERT or GPT are trained on tons of general text — like giving them a giant reading list.
- But we often want them to do something specific (e.g., read economics papers, classify policy documents).
- We can **adapt** them without starting from scratch:
  - **Domain adaptation:** Let the model keep reading — but now only economics papers, so it gets the jargon.
  - **Supervised fine-tuning:** Show it examples with correct answers, like flashcards. It learns the task (e.g., classify texts).
- We can adapt models efficiently, using smart tricks:
  - **Freeze most layers:** Only train the final layers.
  - **Adapters:** Add small "plugin" layers to specialize without changing the full model.
  - **LoRA:** Update just a tiny part of the model's weights to save memory.

Key idea: Start with a smart generalist — teach it just enough to be useful for your task.

From Language Models to Assistants

- Models like GPT-3.5, GPT-4, and Claude don't just predict text — they follow instructions and talk like assistants.
- This is done through a process called **alignment** — teaching the model to respond the way **humans prefer**.
- **How alignment works:**
  - Give the model good examples of questions and answers.
  - Use human feedback (likes/dislikes) to teach the model what's helpful — this is called RLHF.
- **Why it matters:** Pretrained models know a lot, but they don't know how to help you — alignment makes them useful.
- **For economics and social science:** We want models to answer our research questions clearly, not just generate fluent text.

Result: Aligned models become helpful, goal-directed assistants — not just autocomplete machines.

Frame Title

```r
library(reticulate)
use_virtualenv("r-reticulate", required = TRUE)
transformers <- import("transformers")
pipeline <- transformers$pipeline
classifier <- pipeline("sentiment-analysis")
classifier("I love working with large language models!")
```

```
$name
[1] "POSITIVE"

$score
[1] 0.9998
```

Section 3

Similarity & clustering

Document vectors

- **Bag-of-words (BoW)**: raw & tf-idf weighted $\mathbf{x}_d$, cosine similarity $\frac{x_{d'} \cdot x_{d''}}{\|x_{d'}\| \|x_{d''}\|}$.
- **Average word embeddings**: $\frac{1}{N_d} \sum_{n=1}^{N_d} \rho_{w_{d,n}}$ or tf-idf weighted average.
- **SBERT sentence embeddings**: out-of-the-box 768-dim vectors.
- **Vector dimension reduction**: PCA (LSA), NMF, LDA on $\mathbf{X}$ to get lower-K topic distributions.
- **Clustering**: k-means, hierarchical (Ward's linkage) using cosine or Euclidean distance.

Document vectors

- **Bag-of-words (BoW)**: raw & tf-idf weighted x_d, cosine similarity $\frac{x_{d'} \cdot x_{d''}}{\|x_{d'}\| \|x_{d''}\|}$.
- **Average word embeddings**: $\frac{1}{N_d} \sum_{n=1}^{N_d} \rho_{w_{d,n}}$ or tf-idf weighted average.
- **SBERT sentence embeddings**: out-of-the-box 768-dim vectors.
- **Vector dimension reduction**: PCA (LSA), NMF, LDA on $\mathbf{X}$ to get lower-K topic distributions.
- **Clustering**: k-means, hierarchical (Ward's linkage) using cosine or Euclidean distance.

Central-bank speeches example

- Embed 8 years of ECB speeches with SBERT.
- Cluster with hierarchical Ward linkage; identify "inflation-hawk" vs "climate-risk" clusters.
- Compute shift in cluster proportions pre- and post-COVID to capture thematic changes.
- **Reference**: Hansen, Ioannidis, McMahon (2023).

Measuring semantic similarity

- **Cosine similarity** between vectors $\mathbf{v}, \mathbf{w}$: $\cos(\theta) = \dfrac{v^\top w}{\|v\|\|w\|}$.
- Cosine $\in [-1, 1]$, robust to document length.
- **Applications**:
  - *Pairwise* firm-filing similarity as proxy for product competition (Hoberg & Phillips 2016).
  - *Clustering* news articles into stories (Cagé et al. 2020).
  - *Nearest neighbors* for recommendation, anomaly detection.
- **Distance metrics**: Euclidean, Manhattan, but cosine preferred for high-dim sparse spaces.

Comparing document similarity measures (Ash & Hansen 2023)

- Alternative pipelines produce different similarity ranks:
  1. Raw BoW counts vs tf-idf BoW.
  2. Average pretrained GloVe vs corpus-trained GloVe vs Word2Vec.
  3. Dimensionality reduction: LSA vs NMF vs LDA.
- **Finding**: Pairwise correlations of cosine similarities often ≈ 0.6–0.8, not 1.0.
- **Implication**: Downstream regression coefficients (e.g. linking similarity to sector overlap or return correlations) differ by pipeline choice.
- **Takeaway**: Validate choices via ex-ante benchmarks or human-annotated samples.

Clustering in practice

- **k-means**: choose k, initialize centroids, iterate assignment– update–assignment.
- **Hierarchical clustering**:
  - Agglomerative (bottom-up): start with each doc as cluster, merge by minimal distance (Ward's).
  - Divisive (top-down): start with all docs, split recursively.
- **Choosing k or number of clusters**:
  - Elbow/knee method on within-cluster sum of squares.
  - Silhouette score.
  - Economic interpretability: e.g. industry categories (Hoberg & Phillips 2016 used 10 clusters to replicate NAICS2).
- **Visualization**: dendrograms, t-SNE, UMAP on embedding space.

Section 4

Applications in Economics

Subsection 1

Four measurement problems (Ash & Hansen 2023)

Problem I: Measuring document similarity

- **Goal**: Quantify how "close" two documents are in meaning.
- **Approaches**:
  1. **BoW-based**: raw counts or tf-idf $\rightarrow$ cosine.
  2. **Embeddings**: average word vectors $\rightarrow$ cosine.
  3. **Topic model sharing**: cosine on topic loadings (LDA).
- **Economics examples**:
  - *Industry overlap*: Hoberg & Phillips (2010, 2016) use BoW tf-idf on product-descriptions in 10-K.
  - *Patent–patent*: Novelty and influence (Kelly et al. 2021).
  - *Syllabi vs research*: Biasi & Ma (2022), measure gaps between teaching and frontier.

Problem II: Concept detection

- **Goal**: Detect presence or intensity of an economic concept in text (e.g. policy uncertainty, sentiment, skills).
- **Methods**:
  1. **Dictionaries / pattern matching**: Baker et al. (2016) – economic + uncertainty + policy term sets.
  2. **Topic model filtering**: Identify relevant topics (LDA) then apply dictionary in subset.
  3. **Embedding-augmented lexicons**: Seed sets $\rightarrow$ nearest neighbors in embedding space $\rightarrow$ expanded term set (Hanley & Hoberg 2019).
  4. **Supervised classification**: Human-annotated sample $\rightarrow$ train BERT or random forest $\rightarrow$ scale up (Hansen et al. 2023).
- **Economics examples**:
  - *Economic policy uncertainty* (EPU): Baker et al. (2016).
  - *Skills demand*: Compare job postings to O*NET (Hansen et al. 2021).
  - *Remote work*: Fine-tune BERT on annotated postings (Hansen et al. 2023).

Problem III: How concepts relate

- **Goal**: Quantify co-occurrence or semantic association between concepts (e.g. gender and emotion, class and politics).
- **Methods**:
  1. **Local co-occurrence counts**: Count windows with terms from two dictionaries (Apel & Blix Grimaldi 2014).
  2. **WEAT (Embedding association test)**: Project words onto axes defined by attribute sets (Caliskan et al. 2017, Kozlowski et al. 2019).
  3. **Syntactic patterns**: Extract dependency triples (actor–verb–patient) to capture directed relationships (Ash et al. 2023).
- **Economics examples**:
  - *Gender attitudes*: Use WEAT on judge opinions—Ash et al. (2020b).
  - *Sentiment × topics*: Larsen & Thorsrud (2019) – apply sentiment dictionary within LDA topics.
  - *Narrative networks*: Dependency triples among Congressional speeches (Ash et al. 2023).

Problem IV: Associating text with metadata

- **Goal**: Use document text to predict or explain metadata (e.g. political bias, firm returns).
- **Methods**:
  1. **Supervised BoW**: LASSO/logistic on term counts (Gentzkow & Shapiro 2010).
  2. **Topical regression**: Structural topic models regressing topics on covariates (Roberts et al. 2014).
  3. **Transformer fine-tuning**: Fine-tune BERT to predict labels (Bana 2022).
- **Economics examples**:
  - *Media slant*: Train on Congressional speeches $\rightarrow$ predict newspaper article ideology (Widmer et al. 2020).
  - *Stock returns*: Use LDA or embeddings to predict firm returns from MD&A sections (Davis et al. 2020).
  - *Wage prediction*: BERT on job postings to predict salaries (Bana 2022).

Econometric considerations

- **Measurement error**: Text-derived measures have sampling model uncertainty—downstream regressions ignore it.
- **Data-generating process**: BoW $\rightarrow$ multinomial; LDA / LSA $\rightarrow$ latent factor. Must account for dependencies when regressing.
- **Joint modeling**: Supervised topic models (sLDA, STM) incorporate covariates into topic formation (Mcauliffe & Blei 2007; Roberts et al. 2014).
- **Identification threats**: Spurious correlations—e.g. remote work predictions confounded by occupation changes during recession.
- **Validation**: Human-annotated holdouts, concept benchmarks, ex ante test sets—limited in economics vs NLP.

Validation challenges (Ash & Hansen 2023)

- **Divergent pipelines**: Cosine similarities from different pipelines often correlate ≈ 0.6–0.8 (not 1.0).
- **Downstream impacts**: Regression coefficients linking similarity to economic covariates (sector, returns, size) vary significantly by pipeline.
- **No common benchmarks**: Unlike GLUE/MultiNLI in NLP, economics lacks standardized tasks (e.g. domain-specific analogy tests, expert-coded similarity judgments).
- **Towards battery of tasks**: Propose creating standardized economic NLP benchmark datasets (e.g. risk factor similarity, policy event detection, firm-stock co-movement).

Section 5

Application: AI-Generated Production Networks

Overview of AIPNET (Fetzer et al. 2024)

- **Goal**: Construct a granular input–output network over 5,000 product nodes using generative AI.
- **Motivation**: Traditional IO tables (e.g. WIOD, EORA) miss fine-grained product linkages; AIPNET recovers detailed inter-dependencies.
- **Key contributions**:
  1. A two-step "build–prune" pipeline leveraging prompt-tuned large language models to classify directed edges between product categories.
  2. A public dataset of AI-generated production links, enabling analysis of shifts in product and country network positions during the twenty-first century.
  3. An empirical application: measuring spillovers from the 2017 Qatar blockade through AIPNET.

Build–Prune Methodology

- **Step 1: Build**
  - Generate an initial set of candidate edges between each ordered pair of products (5,000 $\times$ 5,000 = 25 million pairs).
  - Use an ensemble of prompt-tuned generative AI classifiers (e.g. GPT-4) to predict whether product i is an input to product j based on textual definitions and known IO schemas.
  - For each pair (i, j), record the probability $p_{i \to j}$ that an edge exists.
- **Step 2: Prune**
  - Re-evaluate all candidate edges in light of the full "Build" output distribution, correcting spurious links and enforcing global consistency (e.g. no self-loops, acyclicity within certain product clusters).
  - Retain edges with posterior probability above a calibrated threshold τ to control for false positives.
- **Output**: A directed AIPNET adjacency matrix $A_{ij} \in \{0, 1\}$ over 5,000 product categories.

LLM Query Examples for AIPNET

- **Prompt 1 (Direct yes/no):**
  "Given the following two products: • Product i: [insert desc] • Product j: [insert desc] Would Product i typically be used as an input in manufacturing Product j? Answer 'Yes' or 'No,' and briefly explain why."
- **Prompt 2 (Probability score):**
  "Here are two HS codes and their official descriptions: – HS 1006: Rice, semi-milled or wholly milled, whether or not polished or glazed. – HS 100630: Rice, husked (brown). Based on these, is 'husked (brown) rice' (HS 100630) used as an input to produce 'rice, semi-milled' (HS 1006)? Provide a probability (0–100%) and a one-sentence justification."
- **Prompt 3 (Enumerate inputs):**
  "List all raw materials or intermediate goods that are commonly used to produce 'smartphone assembly' (product j). Then indicate whether 'semiconductor wafers' (product i) belong in that list, and why."
- **Prompt 4 (Edge-case check):**
  "Consider Product i: 'Engine parts for combustion vehicles' and Product j: 'Electric vehicle batteries.' Would 'engine parts for combustion vehicles' ever feed into 'electric vehicle batteries'? Explain whether there is any production link (Yes/No)."
- **Prompt 5 (Consistency refinement):**
  "We have high-confidence inputs for 'automobile assembly' (product j) that include: • Steel sheets • Engine blocks • Transmission gearboxes Could 'tire rubber compounds' (product i) also serve as an input to 'automobile assembly'? If so, indicate why; if not, explain why it's excluded."

Step 2 – *Prune* candidate edges

1. **Edge-specific re-scoring**
   For each ordered HS pair (i, j) ask an LLM:
   "Is `i` typically used as an input to make `j`? Return a 0–100 % probability."
2. **Ensemble averaging**

   Repeat the query L times with different prompts/seeds and set $\bar{r}_{ij} = \frac{1}{L} \sum_{\ell=1}^{L} r_{ij}^{(\ell)}$.
3. **Keep / drop rule**
   - *Threshold*: keep edge if $\bar{r}_{ij} \geq \varphi$, or
   - *Top-k*: keep the k highest-scoring edges overall.
4. **Economic sanity filters**
   - Remove self-loops $(i = i)$.
   - Collapse reciprocal duplicates, keep stronger direction.
   - Break short production cycles by deleting the weakest link.
   - Drop edges that imply "backwards" flow (e.g. `T-shirt` $\rightarrow$ `cotton`).

What pruning achieves

- **Scale**: from $\sim$25 million candidates $\rightarrow$ $\approx$**1 million** high-confidence links (mean out-degree ≈ 240 across 5 021 HS-6 products).
- **Validity**: surviving binary adjacency explains 40–55 % of the variance in official U.S. Mexican I-O coefficients — random graphs do $\ll$ this.
- **Robustness**: removing as little as 10 % of the pruned edges cuts the I-O fit by half, showing the kept links carry real economic signal.

Intuition

The prune step is a second-pass ensemble classifier: *"Ask a narrower question about every tentative edge, average the answers, then keep only the links that stay highly probable & economically plausible."*

Network Structure and Descriptive Statistics

- **Network density**: Approximately 1.2 million edges remain after pruning, yielding an average out-degree of ≈240—much finer than traditional sector-level IO.
- **Product centrality**: Use eigenvector centrality to rank products by their "upstream" importance.
  - Top "upstream" nodes: "Basic chemicals," "Engine parts," "Semiconductor wafers."
  - Top "downstream" nodes: "Consumer electronics," "Automobile assembly," "Medical devices."
- **Country-level mapping**: Map each country's export basket onto the AIPNET product space to compute a "development index" reflecting proximity to central products.
- **Temporal evolution**: Compare country indices in 2000 vs. 2020 to document "network upward mobility" (e.g. Vietnam's shift into higher-value electronics).

(Fetzer et al. 2024)

Application: 2017 Qatar Blockade Spillovers

- **Natural experiment**: In June 2017, Saudi Arabia, UAE, Bahrain, and Egypt imposed a blockade on Qatar, abruptly disrupting regional trade in hydrocarbons and re-exports.
- **Hypothesis**: Shock to Qatar's exports propagates through AIPNET to trading partners via intermediate inputs—measured as changes in export volumes of affected products.
- **Empirical strategy**:
  1. Identify Qatar's top-10 exported products in 2016 (e.g. LNG, petrochemicals).
  2. Trace downstream "neighbors" in AIPNET (products that use those as inputs).
  3. For each partner country c and month t, construct a weighted "exposure index":

$$\text{Exposure}_{c,t} = \sum_{p \in \mathcal{P}_Q} A_{p \to q} \, \Delta\text{Exports}^{Q}_{p,t} \times \frac{\text{Imports}_{c,p,\,t-1}}{\sum_{p'} \text{Imports}_{c,p',\,t-1}},$$

     where $\Delta\text{Exports}^{Q}_{p,t}$ is the month-on-month change in Qatar's exports of product p; $A_{p \to q} = 1$ if p is input to q.
  4. Regress $\%\Delta\text{Exports}_{c,q,t}$ on $\text{Exposure}_{c,t}$ with country $\times$ product and time-fixed effects.
- **Result**: A 1 standard-deviation higher exposure predicts a 2.3

Key Takeaways from the Application

- **AIPNET enables micro-level IO analysis**: By recovering fine-grained product linkages via generative AI, we can trace shocks at the product level rather than broad SIC/HS aggregations.
- **Spillover quantification**: The 2017 Qatar blockade caused non-trivial "second-round" effects across more than 20 countries—beyond the direct GCC trade partners—highlighting the global reach of localized disruptions.
- **Policy relevance**: Policymakers can use AIPNET to simulate counterfactual shocks (e.g. sanctions on specific inputs) and estimate welfare or trade losses at a detailed product–country level.
- **Extensions**:
  - Incorporate firm-level data to assess heterogeneous firm responses to network shocks.
  - Combine AIPNET with NLP-derived "trade sentiment" measures (e.g. news embeddings) to predict real-time vulnerability.

Section 6

Application: NLP + Innovation

NLP + Innovation: A natural match

Why NLP helps:

- Innovation is hard to measure — it's intangible and abstract.
- Yet innovators leave behind *text*: patents, papers, reports, filings.
- NLP lets us convert these textual traces into structured, analyzable data.

Types of documents:

- Patents (technical content, claims, novelty)
- Scientific articles (knowledge flow, frontier topics)
- Corporate filings, earnings calls (strategy, R&D signals)

NLP + Innovation: A natural match

Why NLP helps:

- Innovation is hard to measure — it's intangible and abstract.
- Yet innovators leave behind *text*: patents, papers, reports, filings.
- NLP lets us convert these textual traces into structured, analyzable data.

Types of documents:

- Patents (technical content, claims, novelty)
- Scientific articles (knowledge flow, frontier topics)
- Corporate filings, earnings calls (strategy, R&D signals)

NLP lets us: classify, summarize, cluster, track trends, and extract signals from unstructured innovation text.

How economists use NLP to study innovation

- **Text similarity** to trace knowledge diffusion (e.g. science → patents)
- **Topic modeling** to detect emerging technologies or frontier fields
- **Sentiment analysis** of earnings calls to gauge innovation optimism
- **Named entity linking** to match firms, inventors, or institutions
- **Temporal analysis** to identify shifts in focus or language over time

Why it matters:

- Stronger tools for evaluating firm-level innovation
- Richer data for understanding the drivers of technological progress

Domain-adapted LLMs in economics

- **FinBERT:** Trained on financial filings and earnings calls.
  - Improves prediction of stock market reaction to sentiment.
- **SciBERT:** Fine-tuned on scientific texts.
  - Used for tracking science–patent linkages and tech emergence.
- **LegalBERT, ClimateBERT:** Support legal and environmental text analysis.

Recent applications:

- Hansen et al. (2023): BERT-based classifier for detecting remote work language in job ads.
- Bana et al. (2022): Fine-tunes BERT to predict wages from job description text.

Further reading and resources

- **Review articles**:
  - Ash & Hansen (2023), *Text Algorithms in Economics*, AER: overview of methods, tasks, challenges.
  - Gentzkow & Shapiro (2019), *Text as Data in Economics*, *Review of Economic Studies*: earlier review of text-as-data methods.
- **Software**:
  - `text2vec`, `tm`, `quanteda` in R.
  - `spacyr`, `udpipe`s for annotation.
  - `reticulate` + `sentence_transformers` for BERT/SBERT in R.
- **Datasets**:
  - SEC 10-K filings (EDGAR).
  - ECB, Fed transcripts (FOMC).
  - O*NET tasks, Burning Glass job postings.
  - Congressional Record (GovInfo API).

References

To be finished