

# Table of Contents

- 1 Introduction to NLP
- 2 Conventional Text Analysis
- 3 Basics of Topic Modeling
- 4 Introduction to Embeddings

# Introduction to NLP

# What is Natural Language Processing?

- Natural Language Processing (NLP) is a subfield of artificial intelligence focused on enabling computers to understand, interpret, and generate human language
- It sits at the intersection of:
  - ▶ Computer Science
  - ▶ Artificial Intelligence
  - ▶ Linguistics
- Goal: Bridge the gap between human communication and computer understanding
- Enables analysis of text data at scales that is **impossible** for manual human analysis

# The Three Dimensions of Language

- **Syntax:** Rules governing **sentence structure**
  - ▶ Word order, grammatical structure
  - ▶ “The cat chased the mouse” vs. “The mouse chased the cat”
- **Semantics:** The **meaning** of words and sentences
  - ▶ Word definitions, relationships between concepts
  - ▶ “Bank” can refer to a financial institution or the edge of a river (that is, there are two meanings here)
- **Pragmatics:** How **context** influences meaning
  - ▶ Meaning depends on the situation, speaker intentions, and shared knowledge
  - ▶ Includes social norms, cultural background, and conversational context
  - ▶ **Example:**
    - ★ “It’s cold in here” could be a neutral observation or an **indirect request** to close a window
    - ★ “Can you pass the salt?” is a request, not a question about ability
- **Key point:** NLP systems must address all three dimensions to fully understand language

# Why Does NLP Matter for the Social Sciences?

- Language is central to social interaction, culture, and meaning-making
- Social scientists study:
  - ▶ Historical documents
  - ▶ Interview transcripts
  - ▶ Social media content
  - ▶ News media
  - ▶ Policy documents
  - ▶ Organizational texts
- Modern societies produce far more text than researchers can manually analyze
- NLP enables systematic examination of large-scale textual data, **revolutionizing qualitative data analysis!**

# Evolution of NLP: From Rules to Deep Learning

- **1950s-1960s: Rule-Based Systems**

- ▶ Language processing via hand-crafted rules and pattern matching
- ▶ Brittle systems — unable to handle linguistic variation

- **1970s-1980s: Knowledge-Based NLP**

- ▶ Shift to meaning representation using structured knowledge (e.g., graph-based knowledge representations)
- ▶ Limited by the need for manually encoded world knowledge

- **1990s-2000s: Statistical Revolution**

- ▶ Machine learning models trained on large text datasets
- ▶ Outperformed rules but lacked deep contextual understanding

- **2010s-Present: Neural Revolution**

- ▶ Deep learning captures context and generates fluent text
- ▶ Achieves human-like results but **remains data-dependent**

**Key theme:** NLP evolved from **explicit rules** → **structured knowledge** → **statistical learning** → **neural models**, driven by advances in data and computing.

# 1950s-1960s: Rule-Based Paradigm (1)

**Philosophical foundation:** Language can be understood through the use of formal rules **Key developments:**

- **1950:** Computer scientist and mathematician **Alan Turing** proposes the “Imitation Game” (i.e., the Turing Test)
- **1954:** IBM-Georgetown experiment translates 60 Russian sentences into English
  - ▶ First public demonstration of **machine translation** (MT)
  - ▶ Used hand-crafted grammar rules and a vocabulary of 250 words
- **1957:** Noam Chomsky's *Syntactic Structures* introduces transformational grammar
  - ▶ **Transformed** linguistics by proposing innate language faculty
  - ▶ Established formal methods for analyzing syntax
  - ▶ Influenced computational linguistics, formal language theory, and programming language development

# 1950s-1960s: Rule-Based Paradigm (2)

## More Key developments:

- **1966:** ELIZA chatbot simulates a psychotherapist using pattern matching
- **1966: ALPAC** (Automatic Language Processing Advisory Committee) report reveals limitations of machine translation
  - ▶ Government-commissioned evaluation of MT research
  - ▶ Concluded that MT was slower, less accurate, and twice as expensive as human translation
  - ▶ Led to significant funding cuts for MT research

## Technical aspects of the rule-based paradigm:

- Hand-crafted grammar rules and dictionaries, pattern matching and template-based responses
- Formal language theory for **parsing** (that is, analyzing sentence structure to identify grammatical components)

**Limitations:** Brittle systems, inability to handle linguistic variation, rule explosion, lack of real understanding!



# Example: The ELIZA Chatbot (1)

- Created in 1966 by Joseph Weizenbaum at MIT and named after Eliza Doolittle (who learned “proper” English) from the play *Pygmalion*
- Simulated a **Rogerian psychotherapist**
  - ▶ A therapeutic approach based in part on reflecting patients’ statements back to them
  - ▶ Ideal for simulation as it relies on rephrasing rather than providing new information
- One of the first programs to process natural language **and** create the illusion of understanding through clever response mechanisms
- **How ELIZA works:**
  - ① Identified keywords in user input
  - ② Applied transformation rules based on patterns
  - ③ Used pre-written templates for responses
  - ④ Redirected questions back to the user
- **Try it yourself:** <https://www.masswerk.at/elizabot/eliza.html>

## Example: The ELIZA Chatbot (2)

```
Welcome to
      EEEEE LL      IIII ZZZZZZ  AAAA
      EE   LL      II      ZZ  AA  AA
      EEEEE LL      II      ZZ  AAAAAA
      EE   LL      II      ZZ  AA  AA
      EEEEE LLLLLL IIII ZZZZZZ  AA  AA

Eliza is a mock Rogerian psychotherapist.
The original program was described by Joseph Weizenbaum in 1966.
This implementation by Norbert Landsteiner 2005.

ELIZA: Is something troubling you ?
YOU:   I am feeling tired.
ELIZA: How long have you been feeling tired ?
YOU:   For one week.
ELIZA: Tell me more about that.
YOU:   I don't feel like it.
ELIZA: Do you wish to be able to feel like it ?
YOU:   Huh?
```

### ELIZA's Legacy:

- Demonstrated both potential and limitations of pattern matching
- Created the “**ELIZA effect**” – people’s tendency to anthropomorphize computers
- Weizenbaum himself became concerned about people’s emotional attachment to the program

# 1970s-1980s: Knowledge-Based Era (1)

**Key insight:** Language understanding requires world knowledge, not just rules

- **From syntax-centered to semantics and pragmatics**

- ▶ Earlier systems focused primarily on grammatical rules
- ▶ New focus on meaning and context

- **From pattern matching to knowledge representation**

- ▶ Pattern matching: Simple templates that match text patterns (like ELIZA)
- ▶ Knowledge representation: Structured formats for storing information about concepts and their relationships

- **From isolated sentences to connected discourse**

- ▶ Earlier systems processed single sentences independently
- ▶ New systems tracked context across multiple sentences in conversation

**Fundamental challenge:** The “knowledge acquisition bottleneck” (termed by Edward Feigenbaum) captures the conclusion that manually encoding all necessary world knowledge is infeasible

# 1970s-1980s: Knowledge-Based Era (2)

## Key approaches to representing knowledge:

- **Semantic networks**

- ▶ Graph structures where concepts are nodes and relationships are edges
- ▶ Example: "Dog" —is-a → "Animal" —has-part→ "Legs"
- ▶ Used to represent taxonomies and relationships between concepts

- **Frames** (Minsky 1975)

- ▶ Data structures with "slots" representing attributes of stereotyped situations
- ▶ Example: Restaurant frame has slots for menu, tables, waitstaff, etc.
- ▶ Allow default values and inheritance from more general frames

- **Expert systems**

- ▶ Domain-specific knowledge bases with inference rules
- ▶ Separate knowledge base from inference engine
- ▶ Example: MYCIN system in the 1970s for medical diagnosis

# 1990s-2000s: Statistical Revolution

**Paradigm shift:** From rules and knowledge to statistics and data **Catalysts of the statistical revolution:**

- Limitations of knowledge-based systems became clear
  - ▶ Knowledge acquisition bottleneck
  - ▶ Difficulty handling ambiguity and exceptions
- Increased availability of digital text (beginning of the World Wide Web)
  - ▶ Emergence of large digital corpora
  - ▶ Access to real-world language examples
- Growth in computing power enabling training on larger datasets
  - ▶ More affordable memory and processing power
  - ▶ Ability to process millions of examples

**Philosophy:** “Let the data speak” such that we learn patterns from examples rather than encoding rules

- Focus on what works empirically rather than linguistic theory
- Use probability to handle uncertainty and ambiguity
- Embrace machine learning as the primary methodology

# Major Breakthroughs and Case Study: IBM's Translation System

## Major breakthroughs:

- **1990-1993:** IBM Models for statistical machine translation
- **1992:** Statistical part-of-speech tagging (Brill tagger by Eric Brill)
- **1996:** Statistical named entity recognition
- **2003:** Statistical sentiment analysis and topic modeling (LDA or Latent Dirichlet Allocation by Blei, Ng, and Jordan)

## Case Study: IBM's Statistical Machine Translation (1990-1993)

- Developed by Peter Brown, Stephen Della Pietra, Robert Mercer at IBM Research
- Radically different from previous rule-based translation systems
- Used parallel corpora (same text in multiple languages) for training
- Core idea: Find the most likely English sentence given a foreign sentence
- Combined translation probabilities with language model probabilities

# From Theory to Practice: Progress and Limitations

## A Famous Example:

*"The spirit is willing but the flesh is weak"*

↓ *Translate to Russian and back*

*"The vodka is good but the meat is rotten"*

## Why this example matters:

- Initially used to criticize early machine translation attempts
- Demonstrates the challenge of preserving idiomatic meaning across languages, notwithstanding the successes of purely statistical approaches
- Highlights the need for both statistical patterns and semantic understanding

## Impact:

- Demonstrated data-driven approaches could outperform expert-crafted rules
- Inspired Frederick Jelinek's famous remark: "Every time I fire a linguist, the performance of our speech recognition system goes up."
- Set the foundation for modern neural machine translation systems

# 2010s-present: The Neural Revolution

**Paradigm shift:** From feature engineering to representation learning **Key technological developments:**

- **Word embeddings** (2013): Word2Vec, GloVe for dense vector representations of words
- **Recurrent Neural Networks:** For modeling text sequences
- **Transformers** (2017): Self-attention based architecture
- **Pre-trained language models** (2018+): BERT, GPT, etc.



**Key milestones:**

- **2013:** Word2Vec introduces efficient neural word embeddings
- **2017:** The “Attention is All You Need” paper introduces the **Transformer**
- **2018:** Emergence of BERT and other contextual language models
- **2020+:** GPT-3 and other massive language models demonstrate emergent capabilities

**Impact:** Achieved state-of-the-art performance across NLP tasks, enabling capabilities approaching human-like language understanding and generation.



# Example: GPT (Generative Pre-trained Transformer) Models

## Revolutionary language model family:

- Developed by OpenAI beginning in 2018
- Based on the transformer architecture with a decoder-only design
- Each generation increased dramatically in scale:
  - ▶ GPT-1 (2018): 117 million parameters
  - ▶ GPT-2 (2019): 1.5 billion parameters
  - ▶ GPT-3 (2020): 175 billion parameters
  - ▶ GPT-4 (2023): Parameters not disclosed, but significantly larger

## Emergent capabilities:

- Natural language understanding without explicit programming
- Complex reasoning across diverse domains
- Coding abilities and problem-solving
- Following subtle instructions
- Creative content generation

# Things to Know

- **NLP is an AI subfield** at the intersection of computer science, linguistics, and artificial intelligence, aiming to enable machines to interpret and **generate human language**
- Effective language processing involves **syntax**, **semantics**, and **pragmatics** to capture structure, meaning, and context
- **Social scientists** leverage NLP to analyze large textual datasets, bridging the gap between human communication and computational methods
- NLP's **historical evolution** spans:
  - ▶ **Rule-based** (1950s-1960s): Hand-crafted grammar and pattern matching
  - ▶ **Knowledge-based** (1970s-1980s): Structured representations of world knowledge
  - ▶ **Statistical** (1990s-2000s): Data-driven approaches using machine learning
  - ▶ **Neural** (2010s-present): Deep learning, **transformers**, and large language models
- Landmark systems like **ELIZA** and **GPT** reveal both the promise and challenges of language understanding
- The field continues to advance through **representation learning**, **attention mechanisms**, and **pre-trained models** that push performance to near-human levels

# Conventional Text Analysis

# Overview of Conventional NLP

- Conventional NLP refers to **pre-deep learning** approaches (approximately 1950s-2010)
- Key characteristics:
  - ▶ **Rule-based** systems
  - ▶ **Statistical** methods
  - ▶ **Feature engineering**
  - ▶ Shallow machine learning models
- Heavily dependent on **preprocessing**:
  - ▶ Manual **feature design**
  - ▶ Linguistic knowledge incorporation
  - ▶ Domain-specific rules
- Limitations led to the development of deep learning approaches

# The NLP Pipeline

- Processing text usually involves several **sequential steps**:
  - ▶ **Sentence segmentation**
  - ▶ **Tokenization**
  - ▶ **Parts of Speech (POS) tagging**
  - ▶ **Stemming and lemmatization**
  - ▶ Identification of **stop words**
  - ▶ **Feature extraction** (BOW, TF-IDF, etc.)
    - ★ Process of converting preprocessed text into numerical vectors
    - ★ Transforms words/tokens into features that algorithms can process
    - ★ Creates structured representations that capture relevant patterns in text
    - ★ Essential for applying machine learning algorithms to text data
- These steps help transform **unstructured text** into **structured numerical representations**
- Each step provides different information about the text

# Why Preprocess Text?

- Computers can't directly understand **natural language**
- We need to convert text into **numerical representations**
- Preprocessing helps:
  - ▶ **Standardize** text (e.g., lowercase, remove accents)
  - ▶ **Remove noise** (e.g., stop words, special characters)
  - ▶ **Reduce dimensionality** (e.g., stemming, lemmatization)
  - ▶ **Extract meaningful features**
- Better preprocessing leads to better model performance

# Text Normalization Methods

Method	Description	Illustration
Casing	Convert text to lowercase to avoid variations of words based on their casing.	Teddy bear → teddy bear
Accents	Remove accents from the text. Useful technique for accent-prone languages (e.g. French, German, Spanish, Italian).	knuddelbär → knuddelbar
Unicode	Remove out-of-the-ordinary symbols.	teddy bear© → teddy bear

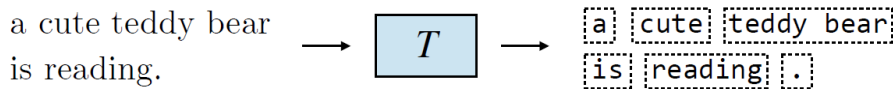
- **Casing:** Converting all text to lowercase to avoid variations based on case
- **Accent removal:** Removing accents from characters in languages like French, German, Spanish
- **Unicode normalization:** Removing special symbols and unusual characters

# Sentence Segmentation

- The process of dividing text into **individual sentences**
- Traditionally: Split text based on **punctuation marks** (., !, ?)
- Challenges:
  - ▶ **Abbreviations** (e.g., "Dr.", "U.S."), Quotes with multiple sentences, and poorly formatted text
  - ▶ These lead to incorrect sentence boundaries that cascade into downstream errors
  - ▶ Misidentified sentences affect tokenization, POS tagging, and semantic analysis
  - ▶ Can significantly impact tasks like summarization, translation, and question answering
- Modern approaches use **machine learning** to handle these cases:
  - ▶ Supervised learning models trained on annotated text
  - ▶ Classification algorithms that consider context, not just punctuation
  - ▶ Features include surrounding words, characters, and their patterns
  - ▶ Models like **Conditional Random Fields (CRFs)** and neural networks can identify sentence boundaries accurately



# Tokenization



- **Tokenization** is the process of breaking text into individual **tokens** (words, characters, or subwords)
- Word tokenization: Split text on whitespace and punctuation
- Example:
  - ▶ "Machine learning algorithms are used in applications"
  - ▶ Tokens: ["Machine", "learning", "algorithms", "are", "used", "in", "applications"]
- Tokenization is essential because:
  - ▶ It creates the **basic units** (tokens) for all subsequent NLP tasks
  - ▶ Algorithms can't process raw text—they need **discrete units**
  - ▶ It enables counting, indexing, and statistical analysis of text
  - ▶ It's the **foundation** for building vocabulary and feature representations

# Parts of Speech (POS) Tagging

- Assigns **grammatical categories** to each token
- Common POS tags:
  - ▶ **NN**: Noun (machine, learning)
  - ▶ **VB**: Verb (run, compute)
  - ▶ **JJ**: Adjective (good, large)
  - ▶ **RB**: Adverb (quickly, very)
- Example:
  - ▶ "Machine (NN) learning (NN) algorithms (NNS) are (VBP) used (VBN) in (IN) applications (NNS)"
- Helps with understanding sentence structure and word meaning in context
- Social science applications of POS tagging:
  - ▶ Political speech analysis: Compare verb/adjective usage across political parties
  - ▶ Gender studies: Analyze differences in adjective usage in descriptions of **men vs. women**
  - ▶ Social media research: Identify emotional content through adjective and adverb usage
  - ▶ Historical text analysis: Track changes in language use and formal structures over time

# Stemming and Lemmatization

- Stemming and lemmatization operate on tokens after tokenization
- The workflow is typically: **Tokenize** → **POS tag** → **Lemmatize**
- Both reduce words to their **base or root form**
- **Stemming:**
  - ▶ Simple, **rule-based truncation** of words
  - ▶ Faster but less accurate
  - ▶ Example: "running", "runner", "runs" → "run"
  - ▶ May produce non-dictionary words: "machine" → "machin"
- **Lemmatization:**
  - ▶ Uses **vocabulary and morphological analysis**
  - ▶ Context-aware, returns dictionary words
  - ▶ Example: "better" → "good", "was" → "be"
  - ▶ Computationally more expensive but more accurate

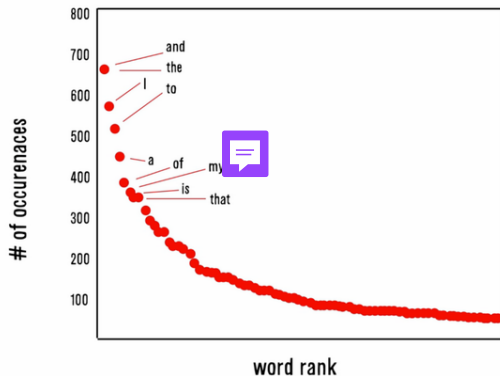
# Stop Words Removal

- **Stop words** are common words that often add little semantic value:
  - ▶ Articles: a, an, the
  - ▶ Prepositions: in, on, at
  - ▶ Conjunctions: and, but, or
  - ▶ Pronouns: I, you, he, she
- Example:
  - ▶ Original: "The machine learning algorithms are used in a variety of applications"
  - ▶ After removal: "machine learning algorithms used variety applications"
- Domain-specific stop words may be needed:
  - ▶ Medical texts: "patient," "treatment," "hospital" (when analyzing specific conditions)
  - ▶ Legal documents: "court," "law," "case" (when analyzing particular legal arguments)
  - ▶ Technical manuals: "system," "user," "process" (when analyzing specific features)
  - ▶ Academic papers: "study," "research," "analysis" (when analyzing specific findings)
- Caution: Sometimes stop words are important for certain tasks (sentiment analysis, question answering)
- Removing stop words in modern NLP:

# An Aside on Zipf's Law

- **Zipf's Law**, discovered in 1935 by the linguist George Zipf, states that in a given corpus, the frequency of any word is inversely proportional to its rank in the frequency table
- **Mathematically:**  $f \propto \frac{1}{r}$ , where:
  - ▶  $f$  is the frequency of the word and  $\propto$  means “is proportional to”
  - ▶  $r$  is the rank of the word in the frequency table
- **Implications:**
  - ▶ A few words (like “the”, “is”, “and”) occur very frequently
  - ▶ Many words occur rarely
  - ▶ The distribution follows a **power law**
- **Visual Representation:**
  - ▶ When plotted on a log-log scale, word frequency versus rank forms a straight line with a slope of approximately -1
- **Relevance to Text Analysis:**
  - ▶ Highlights the need for techniques like TF-IDF to adjust for the “long-tailed” distribution of word frequencies (or removing stopwords)

# Zipf's Law: Word Frequency in Romeo and Juliet



The x-axis is rank while y-axis is word counts. If both axes were logged, then this would be roughly straight line with a slope of -1. The key idea here is that the more informative, rare words form part of the long tail, and that these words can be overshadowed by common but less informative words.

# From Tokens to Numbers: Text Representation

- After preprocessing, we need to convert tokens into **numerical format**
- This transformation enables:
  - ▶ Application of **mathematical operations**
  - ▶ Use of **machine learning algorithms**
  - ▶ Computation of **similarity** between documents or words
- Main text representation methods:
  - ▶ **One-hot encoding**: Binary vectors representing presence of words
  - ▶ **Bag of Words (BOW)**: Frequency-based representation of a document
    - ★ Often organized into a **Term-Document Matrix (TDM)**
  - ▶ **TF-IDF**: Weighting terms by importance
- For the next set of slides we will examine a consistent example that entails three short documents:
  - ▶ Doc1: "The cat sat on the mat"
  - ▶ Doc2: "The dog chased the cat"
  - ▶ Doc3: "The mat was new"

# One-Hot Encoding: Introduction

- One-hot encoding represents each unique word as a **binary vector**
  - ▶ It is called “one-hot encoding” because only one element is “hot” (1) while all others are “cold” (0)
- Key properties:
  - ▶ Vector length equals **vocabulary size**
  - ▶ **1 at the position** corresponding to the word, **0 elsewhere**
  - ▶ Each word has its own **unique representation**
- Used for:
  - ▶ Converting unstructured data (words) to numerical format
  - ▶ Building vocabulary indices (mapping words to unique numeric IDs) contextual models)
- Each word is treated as a **discrete entity** with no inherent relationship to other words

Example: Vocab = {cat, dog, mat}  $\Rightarrow$  "cat" = [1,0,0], "dog" = [0,1,0], "mat" = [0,0,1]



# One-Hot Encoding: Example

After removing stop words ("the"), our vocabulary from the three documents:

Word	cat	chased	dog	mat	new	on	sat	was
Index	1	2	3	4	5	6	7	8

- One-hot encoding for individual words:
  - ▶ "cat" = [1, 0, 0, 0, 0, 0, 0, 0]
  - ▶ "mat" = [0, 0, 0, 1, 0, 0, 0, 0]
  - ▶ "dog" = [0, 0, 0, 0, 1, 0, 0, 0]
- Note that each vector has as many elements as the total number of words (i.e., the vocabulary)
- If the word "cat" appears twice in a document, then it is given two identical one-hot encoding vectors<sup>1</sup>
- Problems with one-hot encoding:
  - ▶ **Sparse vectors:** Mostly zeros, computationally inefficient
  - ▶ **No semantic meaning:** "cat" and "dog" (both animals) are as different as "cat" and "new"
  - ▶ **Dimensionality:** Vocabulary size determines vector length, so it's not very **parsimonious**

---

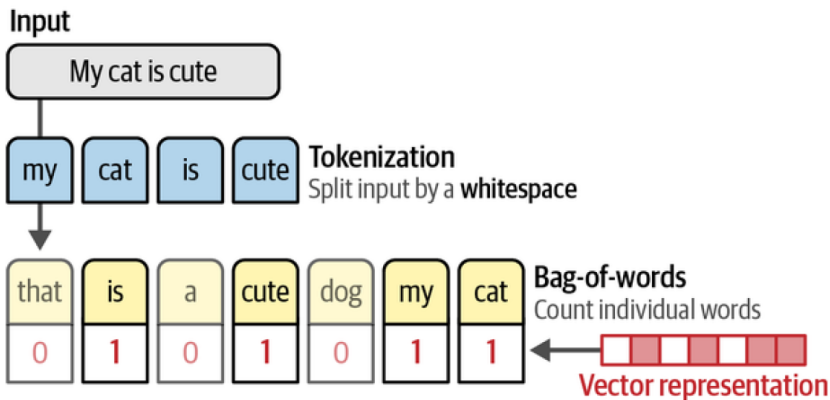
<sup>1</sup>Thus, summing these vectors yields the frequency count for the word in the document, rather than merely indicating its presence. Se

# Bag of Words (BOW): Introduction

- BOW represents each document as an **unordered collection** of words
- Key properties:
  - ▶ **Counts word occurrences** in a document
  - ▶ **Ignores grammar** and word order (hence it's a “bag of words”)
  - ▶ **Represents each document** as a vector of word counts
- BOW for our example documents (after stop word removal):
  - ▶ Doc1: {"cat": 1, "sat": 1, "on": 1, "mat": 1}
  - ▶ Doc2: {"dog": 1, "chased": 1, "cat": 1}
  - ▶ Doc3: {"mat": 1, "was": 1, "new": 1}
- BOW vectors can be viewed as the **sum of one-hot vectors** for all words in a document
  - ▶ Each unique word in the vocabulary is assigned a one-hot vector (a binary vector where only one position is 1 and all others are 0)
  - ▶ A document's BOW representation is obtained by summing the one-hot vectors of all words it contains
  - ▶ This results in a count vector, where each entry represents the frequency of a word in the document

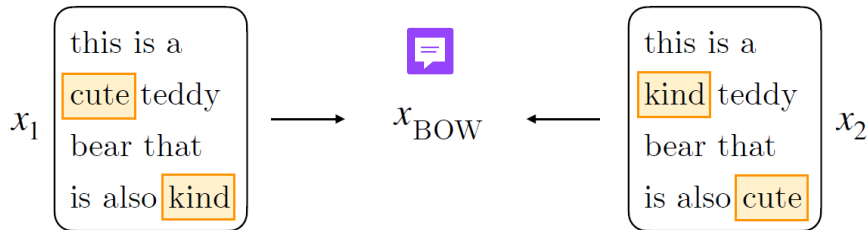


## Bag of Words: Tokenization → Vector of Counts



Tokenization is crucial. Historically tokens are based on **words** although in principle tokens could be larger/smaller units of text.

# Bag of Words: No Distinction!



Both documents are treated in the same way with respect to the two highlighted words!

# Term-Document Matrix (TDM): A Multi-Document BOW


- A TDM organizes **multiple BOW vectors** into a **matrix**, which is simply like a spreadsheet
- Each row represents a **term** in the vocabulary
- Each column represents a **document**
- Cells contain the **count of each term in each document**
- A TDM is essentially a **collection of BOW vectors**
- The TDM inherits BOW properties:
  - ▶ No word order preserved
  - ▶ Only frequency information captured
  - ▶ High dimensionality (that is, many rows) and sparsity (lots of zeros)

# Term-Document Matrix: Example


Term	Doc1	Doc2	Doc3
cat	1	1	0
sat	1	0	0
on	1	0	0
mat	1	0	1
dog	0	1	0
chased	0	1	0
was	0	0	1
new	0	0	1

- This array shows the **term-document matrix** or TDM for our three example documents:
  - ▶ Doc1: "The cat sat on the mat"
  - ▶ Doc2: "The dog chased the cat"
  - ▶ Doc3: "The mat was new"
- Each column is the BOW vector for a document
- This representation allows for **cross-document analysis** (that is, we can compare different documents by seeing how the columns differ)

# Limitations of BOW and TDM

- **Ignores word order** and context
  - ▶ "The dog chased the cat" vs. "The cat chased the dog" have identical BOW representations
- **Sparse vectors** for large vocabularies
  - ▶ Most entries in the matrix are zero 
  - ▶ Computationally inefficient
- No **semantic relationships** between words
  - ▶ "automobile" and "car" are treated as completely different
- **Equally weights** all words (i.e., all words are treated as equal)
  - ▶ Common words like "the" or "and" receive the same weight as distinctive terms, such as "virus" for biomedical research or "socioeconomic" for sociological research
  - ▶ As a result, domain-specific important terms that help us understand how documents differ from one another don't stand out!

# Beyond Simple Word Counts: Introducing TF-IDF

- Word counts in a Term-Document Matrix (TDM)  capture basic information but treat all words as equally important
- A frequent word in one document may not be informative if it appears frequently in many other documents
- We need a way to:
  - ▶ Emphasize words that are important within a document
  - ▶ Reduce the weight of words that appear everywhere and provide little distinction
- **Solution:** Transform word counts into weighted values that capture importance, which leads to Term Frequency-Inverse Document Frequency or **TF-IDF**



# Measuring Word Importance Within a Document: TF

- Some words appear more often than others in a document!
- **Term Frequency (TF)** captures how often a term appears relative to document length:

$$TF(t, d) = \frac{\text{Number of times term } t \text{ appears in document } d}{\text{Total number of terms in document } d}$$

- **Intuition:** If a word appears more frequently in a document, it might be important for that document
- **Link to the TDM:** Instead of raw counts, we “normalize” by dividing by document length
- **Example:**
  - ▶ Document: “the cat sat on the mat”
  - ▶ Raw counts: {“the”: 2, “cat”: 1, “sat”: 1, “on”: 1, “mat”: 1}
  - ▶ TF values: {“the”: 2/6, “cat”: 1/6, “sat”: 1/6, “on”: 1/6, “mat”: 1/6}



# Measuring Word Rarity Across Documents: IDF

- Some words appear in nearly every document (e.g., "the", "and"), making them uninformative
- **Inverse Document Frequency (IDF)** downweights frequent words and highlights rare ones:

$$IDF(t) = \log \left( \frac{\text{Total number of documents}}{\text{Number of documents containing term } t} \right)$$

- **Intuition:** Words that appear in fewer documents are more unique and receive a higher weight
- **Link to TDM:** This adjusts counts to reflect how rare a term is across all documents
- **Example:**

- ▶ If "mat" appears in only 2 out of 100 documents:

$$IDF(\text{"mat"}) = \log \left( \frac{100}{2} \right) = \log(50)$$

- ▶ If "the" appears in all 100 documents:

$$IDF(\text{"the"}) = \log \left( \frac{100}{100} \right) = \log(1) = 0$$

# Combining TF and IDF for Weighted Importance: **TF-IDF**

- TF captures importance within a document, but IDF adjusts for common words
- **TF-IDF** combines both:

$$TF\text{-}IDF(t, d) = TF(t, d) \times IDF(t)$$

- **Intuition:**
  - ▶ If a word appears frequently in a document (high TF) but rarely elsewhere (high IDF), it gets a **high** TF-IDF score
  - ▶ If a word appears everywhere (low IDF), even if frequent in a document, it gets a **low** TF-IDF score
- **Example:**
  - ▶ "mat" has a moderate TF (1/6 in a document) but a high IDF (log(50))
  - ▶ "the" has a higher TF (2/6) but an IDF of 0
  - ▶  $TF\text{-}IDF(\text{"mat"})$  is much higher than  $TF\text{-}IDF(\text{"the"})$ , the latter of which is  $2/6 \times 0 = 0$

# TF-IDF: Example Calculation (1)

Using our three documents outlined previously, let's calculate the TF for each word in each document, as well as the IDF for each word:

Term	TF in Doc1	TF in Doc2	TF in Doc3	IDF
cat	$\frac{1}{4} = 0.25$	$\frac{1}{3} \approx 0.33$	0	$\log(\frac{3}{2}) \approx 0.41$
mat	$\frac{1}{4} = 0.25$	0	$\frac{1}{3} \approx 0.33$	$\log(\frac{3}{2}) \approx 0.41$
dog	0	$\frac{1}{3} \approx 0.33$	0	$\log(\frac{3}{1}) \approx 1.10$
new	0	0	$\frac{1}{3} \approx 0.33$	$\log(\frac{3}{1}) \approx 1.10$

Recall that the TF is simply:

- $TF(t, d) = \frac{\text{Number of times term } t \text{ appears in document } d}{\text{Total number of terms in document } d}$

And that the IDF is as follows, which tells us how rare a word is across all documents:

- $IDF(t) = \log\left(\frac{\text{Total number of documents}}{\text{Number of documents containing term } t}\right)$

## TF-IDF: Example Calculation (2)

Now that we have our TF scores as well as the IDF scores, let's multiply calculate  $TF \times IDF$  to obtain the TF-IDF scores for each word in each document:

Term	TF-IDF Doc1	TF-IDF Doc2	TF-IDF Doc3
cat	$0.25 \times 0.41 \approx 0.10$	$0.33 \times 0.41 \approx 0.14$	0
mat	$0.25 \times 0.41 \approx 0.10$	0	$0.33 \times 0.41 \approx 0.14$
dog	0	$0.33 \times 1.10 \approx 0.36$	0
new	0	0	$0.33 \times 1.10 \approx 0.36$

- Remember that higher TF-IDF scores indicate more importance
- Notice how "dog" and "new" (appearing in only one document) have **higher TF-IDF scores**, while "cat" and "mat" (appearing in multiple documents) have **lower scores**

# TF-IDF: Benefits

- TF-IDF provides several advantages over basic BOW:
  - ▶ **Downweights common words** that appear in many documents
  - ▶ **Highlights distinctive words** that characterize specific documents
  - ▶ **Preserves importance** of frequently occurring terms within a document
  - ▶ **Maintains the BOW structure** while improving term weighting
- Still has limitations:
  - ▶ Does not address word order issues
  - ▶ Does not capture semantic relationships between words
  - ▶ Relies on exact term matching (synonyms are still separate)
- TF-IDF improves information retrieval and document comparison substantially

# Application: Document Similarity with Cosine Similarity

- For any vector  $\mathbf{a} = [a_1, a_2, \dots, a_n]$  and vector  $\mathbf{b} = [b_1, b_2, \dots, b_n]$  Cosine similarity measures the **angle between vector representations** of documents:

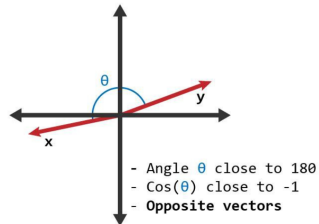
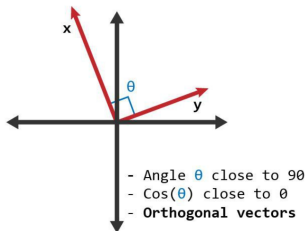
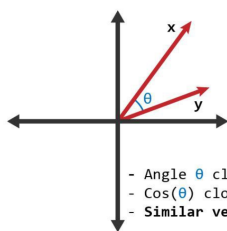
$$\text{Cosine Similarity} = \frac{\mathbf{a} \cdot \mathbf{b}}{\|\mathbf{a}\| \|\mathbf{b}\|} = \frac{\sum_{i=1}^n a_i b_i}{\sqrt{\sum_{i=1}^n a_i^2} \sqrt{\sum_{i=1}^n b_i^2}}$$

- Properties:**

- ▶ A cosine similarity of 1 means the vectors point in the **same direction**
  - ▶ A cosine similarity of 0 means the vectors are **orthogonal** (no similarity)
  - ▶ A cosine similarity of -1 means the vectors point in **opposite directions**
- This formula normalizes the vectors, removing the effect of their magnitude (or length) and focusing solely on their **direction**
- Note that when using word counts, which are non-negative, the cosine similarity ranges from 0 to 1 inclusive
- If the vectors  $\mathbf{a}$  and  $\mathbf{b}$  are centered (i.e., have means of zero), then their cosine similarity will be the same as the correlation coefficient

# Angles vs. Cosine of the Angles

Cosine similarity translates the angle between two vectors into a value between  $-1$  and  $+1$ , where the cosine of the angle determines how similar the vectors are in terms of direction:



Since we are working with word counts, which are always non-negative, the cosine similarity for document vectors will always range from  $0$  (completely dissimilar) to  $1$  (identical direction), rather than from  $-1$  to  $+1$ .



# Comparing the Similarity of Documents

- Using our TDM, let's compute similarity between documents:
  - ▶ Doc1 and Doc2 share only "cat"  $\rightarrow$  cosine similarity  $\approx 0.29$
  - ▶ Doc1 and Doc3 share only "mat"  $\rightarrow$  cosine similarity  $\approx 0.35$
  - ▶ Doc2 and Doc3 share no terms  $\rightarrow$  cosine similarity  $= 0$
- Cosine similarity is **insensitive to document length**, making it ideal for comparing documents of different sizes
- Using TF-IDF in general produces more informative similarity scores by:
  - ▶ Giving more weight to distinctive shared terms
  - ▶ Reducing the impact of commonly shared terms

# Application: Information Retrieval

- Information retrieval systems use text representations to **find relevant documents**
- Basic process:
  - ▶ Convert user query to a vector (BOW or TF-IDF)
  - ▶ Compare query vector to document vectors using similarity measures
  - ▶ Rank documents by similarity scores
  - ▶ Return most similar documents
- Example query: "cat mat"
  - ▶ Query vector (BOW): [1, 0, 0, 1, 0, 0, 0, 0]
  - ▶ Doc1 similarity:  $\approx 0.71$  (highest - contains both terms)
  - ▶ Doc2 similarity:  $\approx 0.41$  (contains only "cat")
  - ▶ Doc3 similarity:  $\approx 0.41$  (contains only "mat")
- TF-IDF typically improves retrieval performance by emphasizing distinctive terms

# Application: Text Analysis for Sociological Research

- TDM and TF-IDF enable several forms of sociological analysis:
  - ▶ **Content analysis:** Systematic analysis of communication content
  - ▶ **Discourse analysis:** Examining language patterns across social groups
  - ▶ **Comparative textual analysis:** Cross-group or cross-time comparisons
- Example applications:
  - ▶ **Political text analysis:** Compare language use across political parties
    - ★ TF-IDF reveals distinctive terminology of each political group
    - ★ Cosine similarity quantifies ideological distance between platforms
  - ▶ **Media framing analysis:** How different outlets present social issues
    - ★ TDM shows term frequency differences across news sources
    - ★ TF-IDF highlights distinctive framing terminology
  - ▶ **Historical language evolution:** Changes in terminology over time
    - ★ Track shifting vocabulary around social concepts
    - ★ Measure similarity between texts from different time periods

# Things to Know

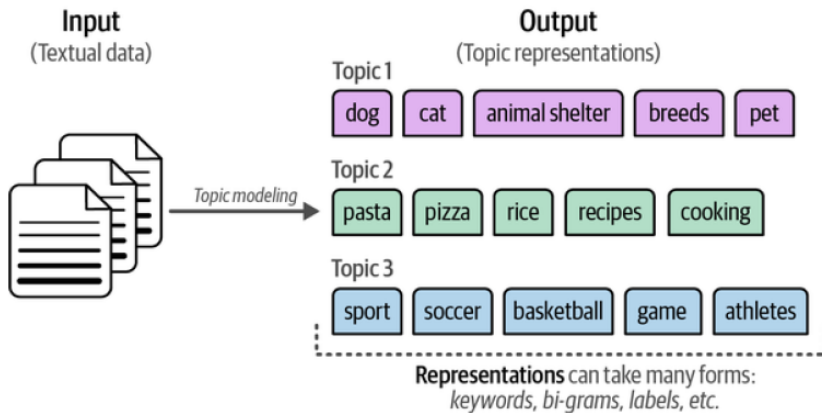
- **Conventional NLP** (pre-2010) relies on **rule-based** and **statistical** methods, along with **feature engineering**
- The **NLP pipeline** typically involves **sentence segmentation**, **tokenization**, **POS tagging**, **lemmatization/stemming**, and **stop word removal**
- **Bag of Words (BOW)** and the **Term-Document Matrix (TDM)** are common ways to convert text into numerical features, but they ignore word order and semantics
- **TF-IDF** addresses “all words are equal” by highlighting **distinctive** terms, improving upon basic frequency counts
- Despite their utility, these approaches often produce **sparse**, high-dimensional vectors that capture only limited context or meaning

# Basics of Topic Modeling

# Topic Modeling: Introduction

- Before the popularity of neural networks (that is, pre-2010 or so), **topic modeling** was arguably one of the leading methods for text analysis
- **Topic modeling** is an unsupervised machine learning approach that uncovers abstract “topics” (or themes) in a set of documents
- Key ideas:
  - ▶ A **topic** is a pattern of words that frequently appear together
  - ▶ Documents are viewed as **mixtures of topics** in different proportions
  - ▶ Words in a document are produced by the topics that make up that document
- Goals of topic modeling:
  - ▶ Reveal hidden thematic structures in collections of text
  - ▶ Organize, search, and summarize large text archives
  - ▶ Lower dimensionality while keeping semantic relations
- Moves beyond TF-IDF by capturing deeper, latent semantic structures

# Basic Idea of Topic Modeling



Both documents are treated in the same way with respect to the two highlighted words!

# Topic Modeling: Why Do We Need It?

- BOW and TF-IDF have notable constraints:
  - ▶ They do not fully capture deeper relationships between words (even though TF-IDF with cosine similarity can measure document similarity, it still lacks a full probabilistic model of how words form topics)
  - ▶ They do not explicitly describe how documents are generated by underlying themes (generative process)<sup>2</sup>
  - ▶ They rely on explicit word counts rather than conceptual groupings
- Topic modeling offers solutions:
  - ▶ Groups thematically related words (even if they rarely co-occur)
  - ▶ Lets documents belong to multiple topics at once
  - ▶ Provides more flexible, “fuzzy” categorization
- Example: A document about “Apple computers” and “Apple fruit” can reflect both technology and food topics

---

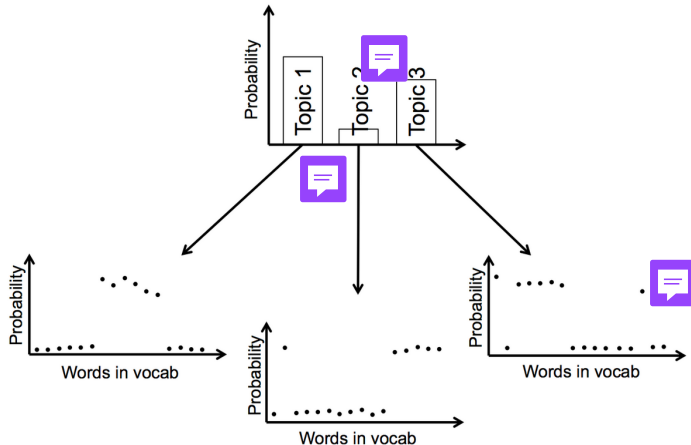
<sup>2</sup>A **generative process** is a statistical model that assumes that the documents arise from latent (i.e., unobserved) topics, each characterized by a probability distribution across different words. These topics “generate” the words in a document through probabilistic sampling.



# Key Concepts in Topic Modeling

- **Document-Term Matrix:** Topic modeling is based on a document-term matrix (DTM) that represents word counts (or frequencies) across documents.
- **Topics:** Groups of words that tend to appear together in documents
  - ▶ Each topic assigns a probability to different words, indicating how strongly they are associated with the topic.
  - ▶ Example: A technology-related topic might assign high probabilities to “computer,” “software,” and “hardware.”
- **Document-Topic Distribution:** The proportion of each topic in a given document
  - ▶ Example: Document A might be 70% about technology and 30% about economics.
- Two key questions in topic modeling:
  - ▶ Which words are most strongly associated with each topic?
  - ▶ Which topics appear in each document, and in what proportions?

# Visualizing a Topic Model



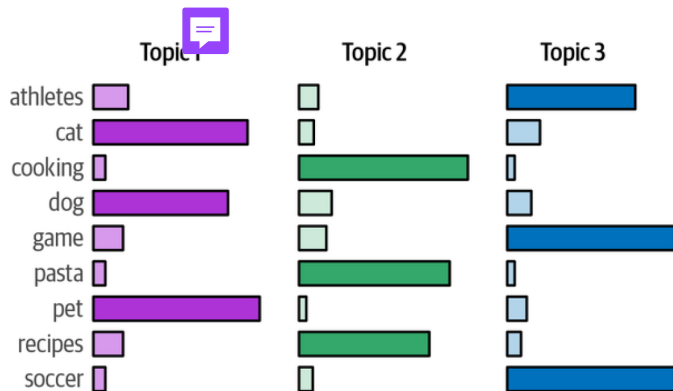
This diagram shows how this might work for three topics and 16 words in the vocabulary. The results for a particular document are the first graph at the top, showing the mix of topics within it. Each topic is itself a probability distribution of words in the vocabulary.

# Latent Dirichlet Allocation (LDA)



- **Latent Dirichlet (Deer-uh-clay) Allocation (LDA)** is one of the most popular topic modeling methods
- It assumes:
  - ▶ Each **topic** is a probability distribution of words (i.e., which words are more likely to occur in that topic)
  - ▶ Each **document** is a probability distribution of topics (i.e., which topics are more likely to occur in that document)
- **Generative perspective:** Documents are formed by
  - 1 Picking a set of topics for a document (weighted by some distribution)
  - 2 For each word, choosing which topic will “generate” that word
  - 3 Drawing the chosen word from that topic’s word distribution
- Minimizes assumptions about the exact meanings of topics; it just looks for consistent patterns of word usage

# Interpreting Topics Using Word Distributions



Each topic is a probability distribution of words. We can see which words most likely appear in each topic to interpret that topic!

# LDA: Intuition and Assumptions

- **Bag of words assumption:** The model treats each document as an unordered collection of words
- **Exchangeability:** Reordering the documents does not affect the outcome, reflecting that each document is equally likely to appear in any position within the corpus
- **Fixed vocabulary:** Before modeling, we define a list of possible words (the vocabulary), and LDA only considers these words during analysis
- **Key Idea:** We observe only the words in each document (the data), while topics and each word's topic assignment are hidden (latent) variables
- **Goal:** Infer the most likely:
  - ▶ Topic assignment for each word
  - ▶ Topic distribution for each document
  - ▶ Word distribution for each topic
- **Methods to infer these distributions:**
  - ▶ **Gibbs sampling:** A Markov chain Monte Carlo method that iteratively samples each latent variable from its conditional distribution
  - ▶ **Variational inference:** An optimization-based approach that approximates the true posterior by a simpler distribution, often with faster convergence

# Topic Modeling: Example Setup

Let's consider a very simple toy example with 4 short documents and try to discover 2 topics:

Document	Content
Doc1	"The cat chased the mouse across the keyboard. The computer crashed afterward."
Doc2	"My laptop battery died during the presentation. I had to plug in the power cord."
Doc3	"The dog barked at the mailman. The cat hid under the bed."
Doc4	"Software updates improve computer security and performance."

For simplicity, let's assume:

- We've already removed stopwords (the, and, at, etc.)
- We're looking for 2 topics ( $K = 2$ )
- We want to find the probability distribution of words for each topic and the probability distribution of topics for each document

# Topic Modeling: Example Results

After running LDA on our toy example, we might get these results:

**Word Distributions for Each Topic**

Word	Topic 1 (Technology)	Topic 2 (Animals)
computer	0.23	0.01
keyboard	0.14	0.03
laptop	0.18	0.00
battery	0.12	0.00
software	0.16	0.01
cat	0.02	0.31
mouse	0.05	0.21
dog	0.00	0.25
barked	0.00	0.15

**Topic Distributions for Each Document**

Document	Topic 1 (Technology)	Topic 2 (Animals)
Doc1	0.40	0.60
Doc2	0.95	0.05
Doc3	0.10	0.90
Doc4	0.98	0.02

Note how Doc1 is a mixture of both topics, while Doc2 and Doc4 are primarily about technology.

# Applications of Topic Modeling

- **Document organization and browsing**

- ▶ **Digital libraries and archives** Classifies documents by topic, enabling efficient search and retrieval in collections like JSTOR or arXiv
- ▶ **News article categorization** Groups articles based on shared themes (e.g., politics, sports, finance) to facilitate personalized news feeds and summarization

- **Recommendation systems**

- ▶ **Suggesting similar documents based on topic similarity** Computes topic distributions for each document and recommends others with high cosine similarity in topic space
- ▶ **Content-based filtering** Matches users with articles, books, or papers by analyzing their past reading history and finding documents with similar topic proportions

- **Content analysis**

- ▶ **Understanding customer feedback** Extracts recurring themes from reviews, social media comments, or surveys to reveal key issues covered
- ▶ **Analyzing political discourse** Identifies dominant topics in political speeches, debates, or legislative records, helping to map ideological positions and policy priorities



# Limitations and Considerations

- **Choosing the Number of Topics ( $K$ ):** No single "correct" value for  $K$ <sup>3</sup>
- **Interpreting Topics**
  - ▶ Topics are probability distributions over words, not predefined categories
  - ▶ Human judgment is required to assign meaningful labels
- **Preprocessing Decisions Matter**
  - ▶ Choices like stemming, lemmatization, and stopword removal influence results
  - ▶ Adequate document length and corpus size improve model stability and coherence
- **Beyond Basic LDA: Advanced Models**
  - ▶ **Dynamic Topic Models:** Track topic shifts over time
  - ▶ **Hierarchical Topic Models:** Capture topic relationships (parent-child structure)
  - ▶ **Correlated Topic Models:** Allow topics to share associations

---

<sup>3</sup> Metrics are available, however. **Perplexity** is a measure indicating how well the model predicts unseen data; lower values suggest a better fit. However, perplexity does not necessarily capture how interpretable the topics are to human readers. **Coherence scores** quantify how semantically consistent the top words in a topic are by examining word co-occurrence patterns, often aligning more closely with human judgment of topic quality.

# Things to Know

- **Topic modeling** (e.g., **LDA**) extends classic text analysis by uncovering **latent themes** from documents
- **BOW-based foundation**: Despite revealing deeper structure, LDA still relies on the **bag-of-words** representation
- **Number of topics is not fixed**: You can get different results depending on the **K** number of topics you think underlies the data!
- **Interpretation requires human judgment**: Topics are discovered automatically, but **human input** is needed to label them meaningfully
- **Preprocessing decisions matter**: Steps like **stopword removal**, **lemmatization**, and **tokenization** affect the final topics
- **Extensions abound**: Variants like **dynamic**, **hierarchical**, or **correlated** topic models address more complex scenarios

# Introduction to Embeddings

# What Are Embeddings?



- **Embeddings** are numerical representations of tokens, words, sentences, or documents as vectors in a space
- They capture **semantic meaning** based on how tokens, words, sentences, or documents are used in context
- Text with similar meanings are placed **close together** in this vector space
- Think of it as creating a "**map of language**" where related concepts are nearby

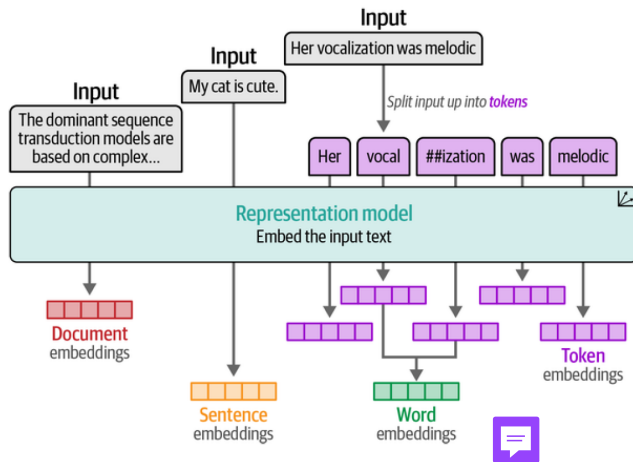
For example, in studies of social stratification, terms like "**inequality**," "**disparity**," and "**stratification**" would be positioned near each other in the embedding space.

Embeddings<sup>4</sup> represent words as mathematical objects that capture meaning.

---

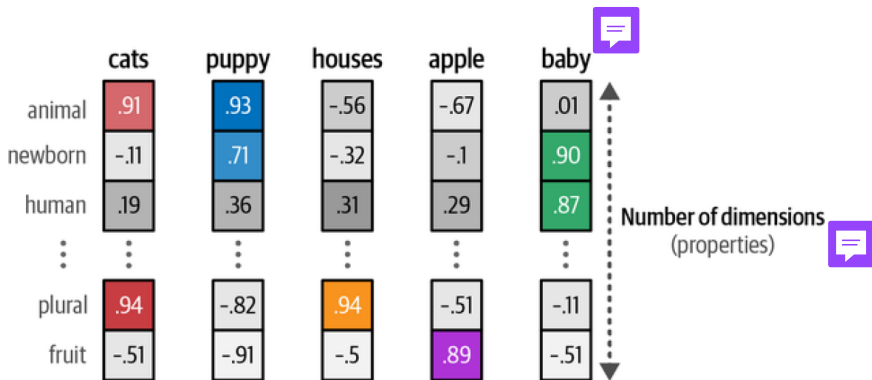
<sup>4</sup>Embeddings represent words as dense vectors typically between 100-300 dimensions. The similarity between vectors measures semantic similarity.

# There are Many Kinds of **Embeddings**



We will focus primarily on **word embeddings**!

# What Word Embeddings Look Like



It is somewhat fictitious, but you can imagine each element of a word embedding as representing numerically some feature/aspect of a word!

# Why Should Social Scientists Care?

- Analyze **large text datasets**<sup>5</sup> (social media, historical documents, policy papers)
- Discover **hidden patterns** and relationships between social concepts
- Measure **cultural and linguistic change** over time
- Quantify **semantic associations** that may reveal societal biases

---

<sup>5</sup>These techniques enable computational text analysis at scale, allowing sociologists to work with corpora that would be impossible to analyze manually.

# From Word Counts to Word Embeddings

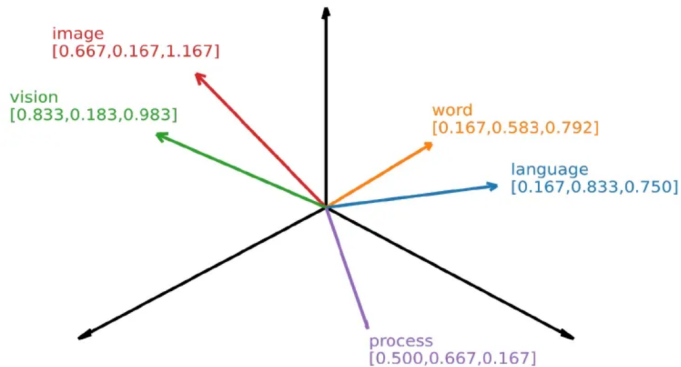
- Both word counts and TF-IDF scores are **fundamentally limited** in how much information they convey, as they are both based on how often words occur in a set of documents
- In recent years word embedding vectors, or **word embeddings** have exploded in popularity
- **Definition:** Word embeddings are **dense vector representations of words** in a lower-dimensional space that capture semantic and syntactic information<sup>6</sup>
  - ▶ Word vectors based on, say, counts alone tend to lie in a very high-dimensional space, and the vectors are “sparse” (i.e., contain many zeros)
  - ▶ By “dense” the vector is filled with informative, non-zero entries, and that there are relatively few of these entries (that is, the vector is “low-dimensional”)
- **Key Idea:** Words that appear in similar contexts have similar meanings and thus similar vector representations

---

<sup>6</sup>Semantic information captures the meanings and conceptual relationships of words, whereas syntactic information describes their grammatical roles and structural arrangement within sentences.



# Visualization of Word Embeddings



# BOW Vectors vs. Word Embeddings

- **BOW Vectors:**

- ▶ Numerical representations derived directly from text based on word counts
- ▶ Considered high-dimensional (with as many entries as there are documents) and sparse (many zeros)

- **Word Embeddings:**

- ▶ A specialized type of word vector that is learned from large corpora
- ▶ Maps words to a **dense**, low-dimensional space (commonly 100–300 dimensions)
- ▶ Captures semantic and syntactic relationships so that words used in similar contexts have similar representations
- ▶ **Example:** The word cat might be represented as  $[0.15, 0.68, -0.32]$ , revealing nuanced meanings and closeness to words like dog

- **Key Differences:** high- vs. low-dimensional, sparsity vs. density, and low vs. high information content

# How Word Embeddings Are Obtained

- **Starting Point:** A large text corpus is used to extract words and their surrounding contexts
- **Defining Context:** For each target word, a **context window** (e.g., a few words before and after) is established
- **Approaches:**
  - ▶ **Predictive Methods:** Train a neural network model, a complex predictive algorithm, to predict a word from its neighboring words (or vice versa),<sup>7</sup> learning vectors that maximize this prediction accuracy.
    - ★ E.g., Word2Vec developed by Tomas Mikolov and colleagues at Google in 2013
  - ▶ **Count-based Methods:** Construct a word–context co-occurrence matrix from the corpus and then reduce its dimensionality (using methods like singular value decomposition or SVD) to obtain dense vectors.
    - ★ E.g., GloVe or “Global Vectors” developed by Jeffrey Pennington et al. at Stanford University in 2014
- **Result:** Each word is represented by a low-dimensional, dense vector that captures both semantic and syntactic properties.

---

<sup>7</sup>In the Skip-Gram model, the network is trained to predict context words given a target word, whereas in the Continuous Bag-of-Words (CBOW) model, the target word is predicted from its surrounding context.

# The Distributional Hypothesis

- **Core Principle:** Words that occur in similar contexts tend to have similar meanings.
- **Famous Maxim:** “You shall know a word by the company it keeps” by the British linguist John Rupert Firth in 1957
- **Implications:**
  - ▶ The meaning of a word is inferred from the words it co-occurs with.
  - ▶ Words that share contexts are likely to belong to similar semantic or syntactic groups.
- **Relevance:** This idea is foundational for methods that convert text into numerical representations.

# Linking Word Embeddings with the Distributional Hypothesis

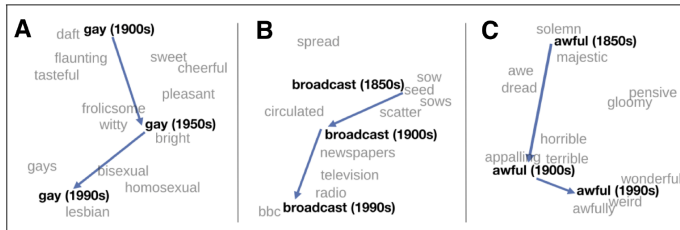
- **Practical Realization:** Word embedding algorithms (e.g., Word2Vec, GloVe) rely on co-occurrence information, such that they are directly applying the distributional hypothesis.
- **How It Works:**
  - ▶ The training goal (or objective) is designed so that **words appearing in similar contexts acquire similar vectors**
  - ▶ Whether using predictive or count-based methods, the underlying data (word–context co-occurrences) reflects the distributional hypothesis
- **Outcome:** The resulting vectors encode nuanced semantic relationships, allowing for effective similarity computations, clustering, and various NLP tasks
- **Summary:** Word embeddings are a numerical embodiment of the distributional hypothesis, transforming contextual similarity into geometric proximity

# Some Properties and Uses of Word Embeddings

Word embeddings have some remarkable properties:

- **Semantic Similarity:** Vectors for semantically similar words are close together (e.g., the vectors for “king” and “queen” are near each other)
- **Analogical Reasoning:** Vector arithmetic can capture analogies between words
  - ▶ **Example:**  $king - man + woman \approx queen$
  - ▶ **Mathematically:**  $\mathbf{v}_{king} - \mathbf{v}_{man} + \mathbf{v}_{woman} \approx \mathbf{v}_{queen}$
- **Clustering:** Words related to similar concepts form distinct clusters in the vector space (e.g., words like “apple,” “banana,” and “orange” cluster together as fruits)
- **Syntactic Relationships:** Embeddings capture grammatical relationships (e.g., the difference between “walk” and “walking” is similar to that between “run” and “running.”)
- **Dimensionality Reduction:** High-dimensional word data is represented in lower dimensions (typically 100–300), making computations more efficient
- **Visualization:** Dimensionality reduction techniques like principal component analysis (PCA) can project embeddings into 2D or 3D space for visualization, revealing patterns and groupings

# Visualizing Social Change Using Word Embeddings



This visualization illustrates the semantic changes of three English words based on word embeddings generated by the Word2Vec algorithm. The first panel traces the semantic shift of the word “gay,” the second panel shows the evolution of the word “broadcast,” and the third panel highlights the transformation of the word “awful” (from “full of awe” to “terrible or appalling”).

# Using Word Embeddings for Analyzing Documents

- **Document Representation:** Instead of using only word count or TF-IDF vectors, we employ word embeddings.
  - ▶ Each word is mapped to a dense vector of fixed length  $d$  (the **embedding dimension**), typically about 100–300 features
  - ▶ A common approach is to **average the word embeddings** for all words in a document to form a single **document vector**; alternatively, one may compute a weighted average (e.g., using TF-IDF weights) to incorporate word frequency
  - ▶ These document vectors are assembled into a matrix of size  $n \times d$ , where  $n$  is the number of documents (each document is a row) and  $d$  is the embedding dimension (each column is a feature)
- **Measuring Similarity:** Once represented as vectors, documents are compared using **cosine similarity** to assess their content similarity
- **Clustering:** Clustering methods (e.g., K-means) are applied to the document vectors (rows of the matrix) to group similar documents
- **Visualization:** Dimensionality reduction techniques such as principal component analysis (PCA) are applied to reduce their  $d$ -dimensional feature space to 2 or 3 dimensions for visualization



# Practical Example: Analyzing Policy Documents

**Research question:** "How has the framing of poverty changed over time in policy documents?"

- ① Collect U.S. policy documents from different decades
- ② Create word embeddings for each time period
- ③ Track how "**poverty**" relates to other concepts:
  - ▶ 1960s: close to "**welfare**", "**assistance**", "**programs**"
  - ▶ 1980s: closer to "**dependency**", "**responsibility**", "**work**"
  - ▶ 2010s: closer to "**opportunity**", "**education**", "**inequality**"

This quantifies the shift from welfare-based to personal responsibility to opportunity-based framing of poverty in American policy discourse

This approach enables **quantitative analysis** of conceptual change over time

# Limitations of Static Embeddings

## 1. Polysemy (“PAH-luh-see-mee”) Problem<sup>8</sup>

- Can't handle words with multiple meanings
- “Class” can mean social class or classroom
- Static embeddings blend these meanings together

## 2. Context Insensitivity

- Same vector regardless of context
- “Capital” means different things in Marx vs. Bourdieu

## 3. Bias Reflection

- Embeddings trained on social text reflect social biases
- Can perpetuate stereotypes if not carefully addressed

---

<sup>8</sup>When using word embeddings in sociological research, these limitations are particularly important because social language is highly contextual and often the subject of study itself is social bias.

# Limitations (Continued)

## 4. Training Data Dependency

- Quality depends entirely on training corpus
- Sociological concepts need sociological texts


## 5. Static Nature

- Cannot adapt to evolving language use
- The meaning of "queer" has changed dramatically over time



These limitations are actually **opportunities for sociological insight** they reveal how meaning is contextual, historically situated, and socially constructed.

# Advancements: Contextualized Word Embeddings

- **Addressing Polysemy (“PAH-luh-see-mee”)**: Assign different vectors to a word based on its context
- Consider the word **“bank”** in different sentences:
  - ▶ **Sentence 1**: “I need to withdraw money from the **bank**” 
  - ▶ **Sentence 2**: “The river **bank** was eroded after the flood.”
- **Traditional (or static) embeddings** would assign the same vector to **“bank”** in both sentences
- **Contextualized (or dynamic) embeddings** generate different vectors for **“bank”** based on surrounding words, capturing the different meanings
- **Models**:
  - ▶ **BERT (Bidirectional Encoder Representations from Transformers)**: Uses transformer architecture to capture context from both left and right directions
  - ▶ **GPT (Generative Pre-trained Transformer)**: Focuses on generating text based on context
- **Benefits**:
  - ▶ Capture word meanings more accurately in different contexts
  - ▶ Improve performance on a variety of NLP tasks


# How Contextualized Embeddings Work

- Use **deep neural networks** with multiple layers
- Process **entire sentences or paragraphs** at once
- Consider **bidirectional context** (words before and after)
- Apply **attention mechanisms** to focus on relevant context
- Pre-trained on **massive text corpora** (billions of words)

Unlike static embeddings, where:

"class"  $\rightarrow$  [0.2, -0.5, 0.1, 0.7...]

With contextualized embeddings:

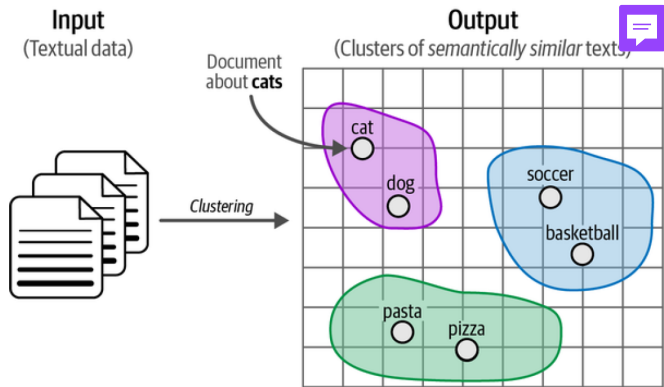
"social  structure"  $\rightarrow$  [0.3, -0.2, 0.4, 0.1...]

"canceled **class** today"  $\rightarrow$  [0.7, -0.8, -0.1, 0.3...]

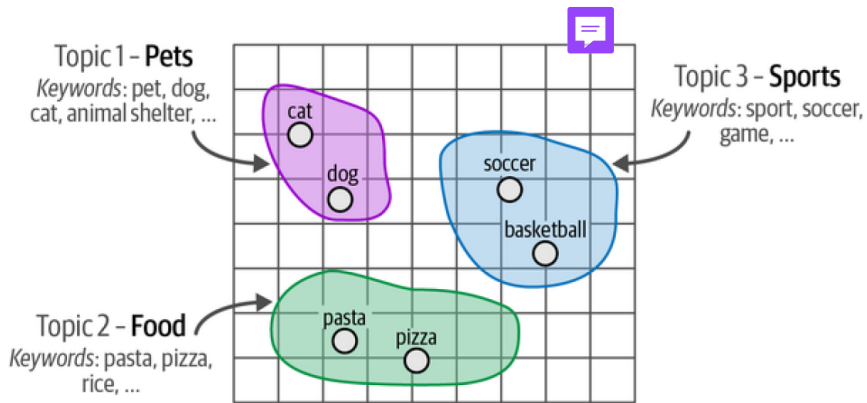
# Pipeline of Text Clustering Using Word Embeddings

- **Goal:** Group large collections of documents based on semantic similarity
- **Steps:**
  - ▶ 1. Convert text to embeddings
  - ▶ 2. Reduce dimensionality
  - ▶ 3. Cluster embeddings
  - ▶ 4. Inspect and interpret the results
- Useful for **exploratory analysis** and discovering **hidden structures** in data

# The General Idea (1)




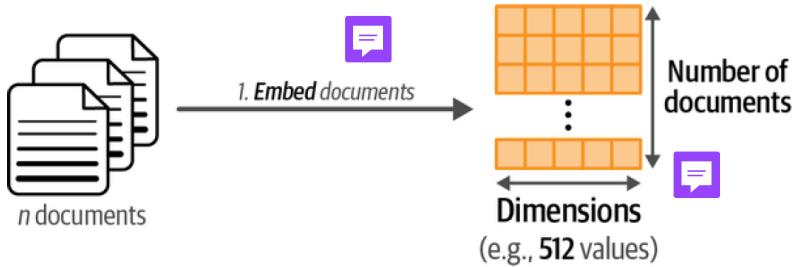
## The General Idea (2)





# Step 1: Converting Text to Embeddings

- **Word embedding**  capture the semantic meaning of words or entire documents
- Embedding models map text into numerical vectors that reflect **similarity**
- High-quality embeddings are **crucial for better clustering** performance
- Semantic embeddings go beyond traditional bag-of-words approaches



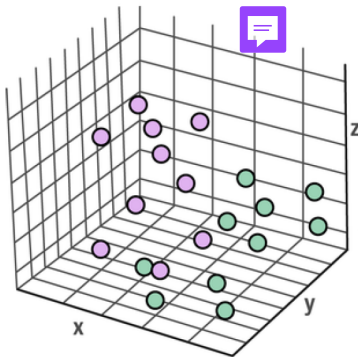
## Step 2: Reducing Dimensionality

- Embeddings can contain hundreds of dimensions, making clustering challenging
- **Uniform Manifold Approximation and Projection (UMAP)** and **Principal Component Analysis (PCA)** are effective dimension reduction methods
- **PCA** is a linear transformation that preserves global variance, whereas **UMAP** is non-linear and prioritizes local neighborhood structure
- Both approaches reduce noise and help clustering algorithms form coherent groups with minimal information loss



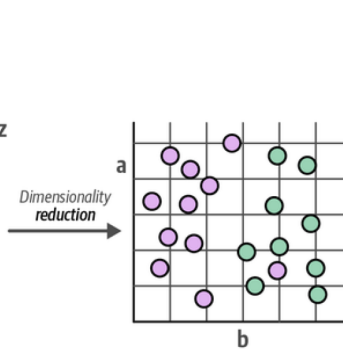
### 3-dimensional space

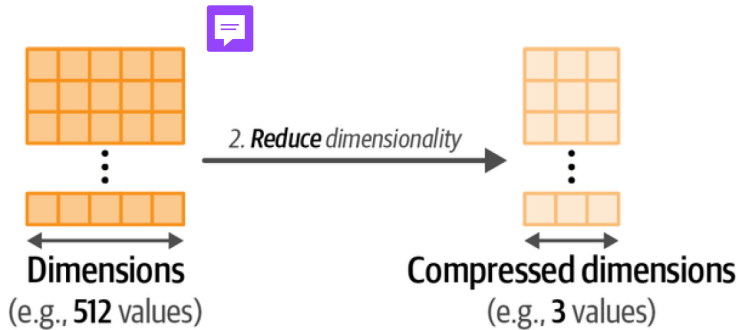
(x, y, and z)



### 2-dimensional space

(a and b)

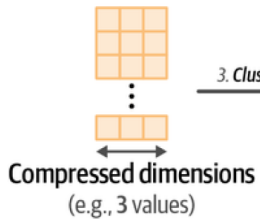




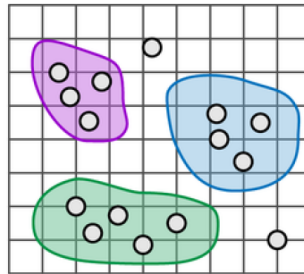
## Step 3: Clustering

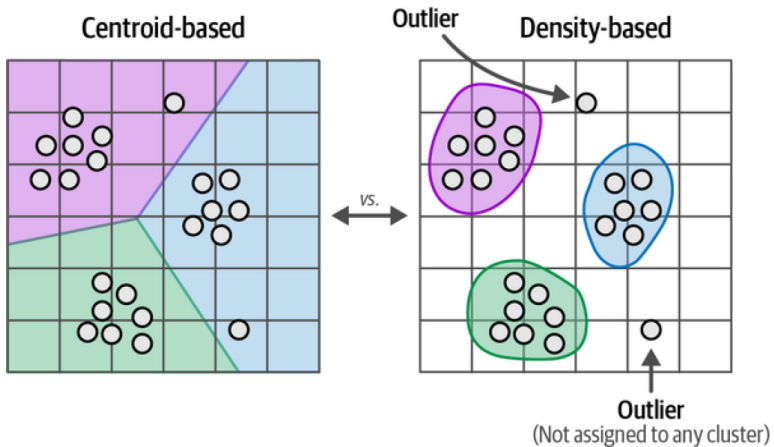


- **Density-based** methods (e.g., Density-Based Spatial Clustering of Applications with Noise or DBSCAN) adaptively find the number of clusters
- **Centroid-based** methods (e.g., K-means) need a predefined cluster count
- **Outliers** or atypical documents remain unclustered, aiding anomaly detection
- The chosen clustering technique impacts how groups are found and interpreted



3. *Cluster* reduced embeddings







## Step 4: Inspecting and Interpreting Clusters

- Manually review documents in each cluster to confirm thematic coherence
- Label each cluster with a **descriptive name** or topic
- **Qualitative analysis** is essential: look for keywords or recurring concepts
- This step reveals hidden **topics**, trends, or anomalies in the dataset

# Things to Know

- **Embeddings** map tokens/words/sentences into **dense numerical vectors** that preserve semantic relationships
- **Similar meaning** words lie **close together** in the embedding space, enabling richer analyses than bag-of-words
- **Static embeddings** (Word2Vec, GloVe) assign each word a single vector, while **contextual embeddings** (BERT, GPT) generate different vectors depending on surrounding words
- **Distributional Hypothesis**: “You shall know a word by the company it keeps” underpins all embedding methods
- **Sociological insights**: Embeddings facilitate large-scale text research on **cultural change** and **semantic shifts** over time
- **Limitations**: Trained on existing text, they may **reflect biases** or fail to capture multiple word senses in static forms