Cleaning, Representing, and Comparing Texts

Korea University Methods Workshop

Taegyoon Kim | DHCSS, KAIST

Jan 22, 2026

Agenda

Things to be covered

- Unit of analysis
- Text segmentation and normalization
- BoW / vector space models
- Word embeddings
- Cosine similarity
- Guided coding

Unit of Analysis

The main element that is being analyzed in a study

- “What” or “who” that is being studied
- Depends on the research question

Unit of Analysis

Typically, information about one unit is recorded as one row

- Think of a survey data set

	A	B	C	D	E	F	G	H	I	J	K
1	ID	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10
2	10010101	Strongly Agree	Strongly Agree	Agree	Agree	Disagree	Disagree	Agree	Strongly Agree	Disagree	Disagree
3	10010102	Strongly Agree	Strongly Agree	Agree	Agree	Disagree	Disagree	Agree	Strongly Agree	Disagree	Disagree
4	10010103	Strongly Agree	Strongly Agree	Agree	Agree	Disagree	Disagree	Agree	Strongly Agree	Disagree	Disagree
5	10010104	Strongly Agree	Strongly Agree	Agree	Agree	Disagree	Disagree	Agree	Strongly Agree	Disagree	Disagree
6	10010105	Strongly Agree	Strongly Agree	Agree	Agree	Disagree	Disagree	Agree	Strongly Agree	Disagree	Disagree
7	10010106	Strongly Agree	Strongly Agree	Agree	Strongly Agree	Disagree	Disagree	Agree	Strongly Agree	Disagree	Disagree
8	10010107	Strongly Agree	Strongly Agree	Agree	Agree	Disagree	Disagree	Agree	Strongly Agree	Disagree	Disagree
9	10010108	Strongly Agree	Strongly Agree	Agree	Agree	Disagree	Disagree	Agree	Strongly Agree	Disagree	Disagree
10	10010109	Agree	Strongly Agree	Agree	Agree	Disagree	Disagree	Agree	Strongly Agree	Disagree	Disagree
11	10010110	Agree	Strongly Agree	Agree	Agree	Disagree	Disagree	Strongly Agree	Strongly Agree	Disagree	Disagree
12	10010111	Agree	Strongly Agree	Agree	Agree	Disagree	Disagree	Strongly Agree	Strongly Agree	Disagree	Disagree
13	10010112	Agree	Strongly Agree	Agree	Agree	Disagree	Disagree	Strongly Agree	Strongly Agree	Disagree	Stongly Disagree
14	10010113	Agree	Strongly Agree	Agree	Agree	Disagree	Disagree	Strongly Agree	Strongly Agree	Disagree	Stongly Disagree
15	10010114	Agree	Agree	Agree	Agree	Disagree	Disagree	Strongly Agree	Strongly Agree	Disagree	Stongly Disagree
16	10010115	Agree	Strongly Agree	Agree	Disagree	Strongly Disagre	Disagree	Strongly Agree	Strongly Agree	Disagree	Disagree
17	10010116	Agree	Strongly Agree	Agree	Agree	Strongly Disagre	Disagree	Strongly Agree	Strongly Agree	Disagree	Disagree
18	10010117	Agree	Strongly Agree	Agree	Agree	Strongly Disagre	Disagree	Strongly Agree	Strongly Agree	Disagree	Disagree
19	10010118	Agree	Strongly Agree	Agree	Agree	Strongly Disagre	Disagree	Strongly Agree	Strongly Agree	Disagree	Disagree
20	10010119	Strongly Agree	Strongly Agree	Agree	Agree	Strongly Disagre	Disagree	Strongly Agree	Strongly Agree	Disagree	Disagree
21	10010120	Strongly Agree	Strongly Agree	Agree	Agree	Strongly Disagre	Disagree	Strongly Agree	Agree	Strongly Disagree	Disagree
22	10010121	Strongly Agree	Disagree	Agree	Agree	Strongly Disagre	Disagree	Strongly Agree	Agree	Strongly Disagree	Disagree
23	10010122	Strongly Agree	Strongly Disagree	Agree	Strongly Agree	Disagree	Disagree	Strongly Agree	Agree	Strongly Disagree	Disagree
24	10010123	Strongly Agree	Strongly Disagree	Strongly Agrees	Strongly Agree	Disagree	Disagree	Strongly Agree	Agree	Strongly Disagree	Disagree
25	10010124	Disagree	Strongly Disagree	Strongly Agrees	Strongly Agree	Disagree	Disagree	Strongly Agree	Agree	Strongly Disagree	Disagree
26	10010125	Disagree	Strongly Agree	Strongly Agrees	Strongly Agree	Disagree	Disagree	Strongly Agree	Agree	Strongly Disagree	Disagree
27	10010126	Disagree	Strongly Agree	Agree	Strongly Agree	Disagree	Disagree	Strongly Agree	Agree	Strongly Disagree	Disagree
28	10010127	Disagree	Strongly Agree	Agree	Agree	Disagree	Disagree	Strongly Agree	Agree	Strongly Disagree	Agree
29	10010128	Disagree	Strongly Agree	Agree	Agree	Disagree	Disagree	Strongly Agree	Agree	Strongly Disagree	Disagree
30	10010129	Disagree	Strongly Agree	Agree	Agree	Disagree	Disagree	Strongly Agree	Agree	Strongly Disagree	Disagree
31	10010130	Disagree	Strongly Agree	Agree	Agree	Agree	Disagree	Strongly Agree	Agree	Strongly Disagree	Disagree
32	10010131	Disagree	Strongly Agree	Agree	Agree	Disagree	Disagree	Strongly Agree	Agree	Disagree	Disagree
33	10010132	Strongly Agree	Strongly Agree	Agree	Agree	Disagree	Disagree	Strongly Agree	Disagree	Disagree	Disagree
34	10010133	Strongly Agree	Strongly Agree	Agree	Agree	Disagree	Disagree	Strongly Agree	Agree	Disagree	Agree
35	10010134	Strongly Agree	Strongly Agree	Agree	Disagree	Disagree	Disagree	Strongly Agree	Agree	Disagree	Disagree

Unit of Analysis

The key consideration is our research question

Unit of Analysis

E.g., Barbera et al. (2019)

- Investigate whether politicians respond to people's policy interests, focused on Twitter (2013–2014)
- Run topic models (**L**atent **D**irichlet **A**llocation) on tweets from ordinary users and 500+ legislators in the U.S.
- Examine if the topics in the former at **t** predicts the latter at **t+1**

Unit of Analysis

E.g., Barbera et al. (2019)

- *"Our definition of "document" is the aggregated total of tweets sent by members of Congress each day"*
- *"Our conceptualization of each day's tweets as the political agenda that each party within each legislative chamber is trying to push for that specific day"*
- *"Conducting an analysis at the tweet level is complex, given its very limited length"*

Tokenization

Breaking up a text into discrete components

- Tokenization is a form of segmentation (= word segmentation)
- Token: each individual component in the document
  - Possibly including numbers, punctuation, or other symbols

Tokenization

“To be or not to be, that is the question”

⟶ “To”, “be”, “or”, “not”, “to”, “be”, “that”, “is”, “the”, “question”

Tokenization

Types

- Each token is of a particular “type”
- The set of types is the vocabulary (often denoted as $|V|$)
- “To be or not to be, that is the question”

  $\longrightarrow$ “to” “be” “or” “not” “that” “is” “the”, “question” ($|V| = 8$)

Tokenization

`“Let us explore tokenization.”`

- Word-level: [“Let”, “us”, “explore”, “tokenization.”]
- Subword-level: [“Let”, “us”, “explore”, “token”, “ization.”]
- Character-level: [“L”, “e”, “t”, “u”, “s”, “e”, “x”, “p”, “l”, “o”, “r”, “e”, “t”, “o”, “k”, “e”, “n”, “i”, “z”, “a”, “t”, “i”, “o”, “n”, “.”]

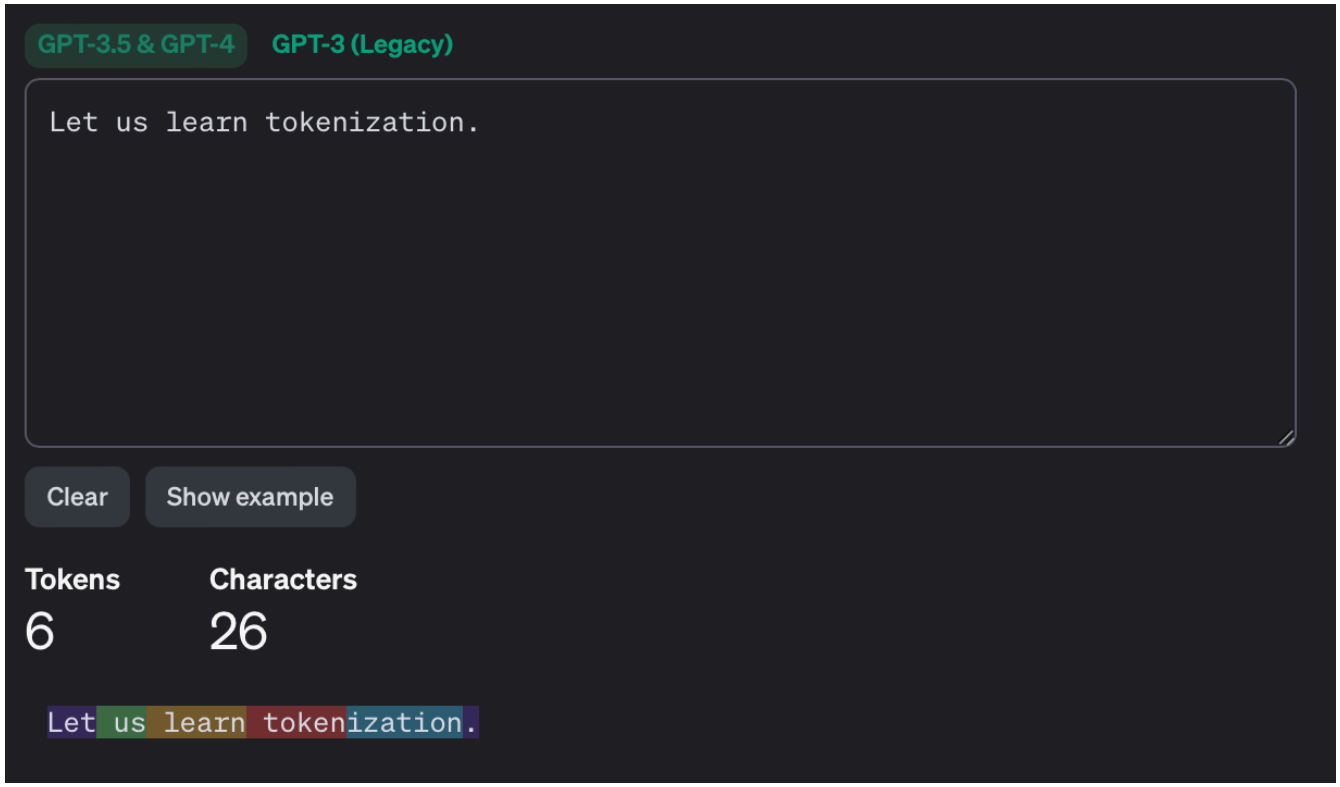
Tokenization

Levels of tokenization

- Words: most common pre-LLM
- Subwords: now prevalent in neural NLP / LLM
  - Handling of OOV (out-of-vocabulary) words
    - E.g, if the model has `vi` (as in virus) and `rologist` (as in neurologist, urologist, etc.), it can handle `virologist`
  - Reduced vocabulary / more efficiency (again, consider “tokenization”)
  - Common approaches include Byte Pair Encoding (BPE) for GPT, WordPiece for BERT
- Character: no meaning (although computationally very efficient)
  - What would be the vocabulary size?

Tokenization

Subword tokenization in GPTs



Tokenization

Tokenization at the word level

Tokenization

Tokenization at the sub-word level

- Very common in LLMs
- Tokenizers are built (trained) as a separate process before model training
- After a tokenizer is initialized and trained, it is then used in the training process of its associated (L)LMs
  → The model is “locked” to its tokenizer
  → (L)LMs are linked with their tokenizers

Tokenization

Tokenization at the sub-word level (cont'd)

- Methods: while there are various methods, they all aim to optimize an efficient set of tokens to represent a texts data set
  - E.g., Byte Pair Encoding or WordPiece
- Special tokens: used to indicate specific roles or structures
  - E.g., `[CLS]` (BERT) or `<s>` (GPT) to mark the beginning of input/output
  - E.g., `[SEP]` (BERT) or `</s>` (GPT) to separate sentences or mark the end of input/output
- Vocabulary size: it should be decided how many tokens to keep in the tokenizer's vocablary

Tokenization

n-grams

- A sequence of *n* adjacent tokens
- Unigrams, bigrams, trigrams, etc.
- Why would we need multi-grams?
  - E.g., “White House”, “look after”, “take care of”, etc.

Tokenization

n-grams

- Be aware of the computational cost
  - Consider the number of all consecutive sets of two words in the corpus
- Alternatively, we can compile a list of particular bi-grams or tri-grams

Tokenization

n-grams

- With modern (L)LMs, we no longer need to explicitly generate n-grams most of the time
- However, the idea behind n-grams remains relevant
- The purpose of *n*-grams is often achieved implicitly through subword tokenization and attention mechanisms in modern transformer-based models
  - This includes capturing local word patterns, collocations, and context,
  - E.g., subword tokenization involves common patterns (e.g., `artificial intelligence` or `New York`)

Segmenting Sentences/paragraphs

Sentence segmentation

- Useful cues: periods, question marks, or exclamation marks
- Prone to errors (the example of `"."`)
  - Abbreviations and initials: “Ph.D.”, “J.K. Rowling”, etc.
  - Decimal numbers: “3.14”
  - Websites and email addresses: “www.kaist.ac.kr”) and email addresses
  - Quotations within a sentence: “He said, ‘Stop.’ Then he left.”
- Rule-based/deterministic or ML-based approaches (part of NLTK and `spaCy`)

Segmenting Sentences/paragraphs

Paragraph segmentation

- Not as commonly addressed
- There are useful cues
  - Newline characters (`\n`) or double newline characters (`\n\n`)
  - Indentations (e.g., `\t`)
  - With HTLM documents, we could potentially use tags (e.g., `<p>`) to parse different parts of the document (not necessarily paragraphs though)
- Fewer specialized libraries or algorithms in Python

Text Normalization

A set of approaches to reducing complexity in text (a.k.a. pre-processing)

- The output from tokenization will contain too many words
- With normalization, vocabulary size can be reduced (computationally more efficient)
- It can enhance many downstream tasks
  - E.g., topic modeling (player, players, playing, etc.)
  - E.g., information retrieval: finding a pattern in a corpus (e.g., Penny, Pennies, penny, pennies, etc.)

Text Normalization

We will discuss five approaches

- Lowercasing
- Removing punctuation
- Removing stop words
- Lemmatization/stemming
- Filtering by frequency

Text Normalization

Lowercasing

- We often replace all capital letter with lowercase letters
- It is assumed that there is no (semantic) difference
- Is it?

Text Normalization

Lowercasing

- Compare “NOW” and “now” in terms of sentiment
- Capital letters also signal the start of of a sentence
- Proper nouns (May vs. may. US vs. us)

Text Normalization

Removing punctuation

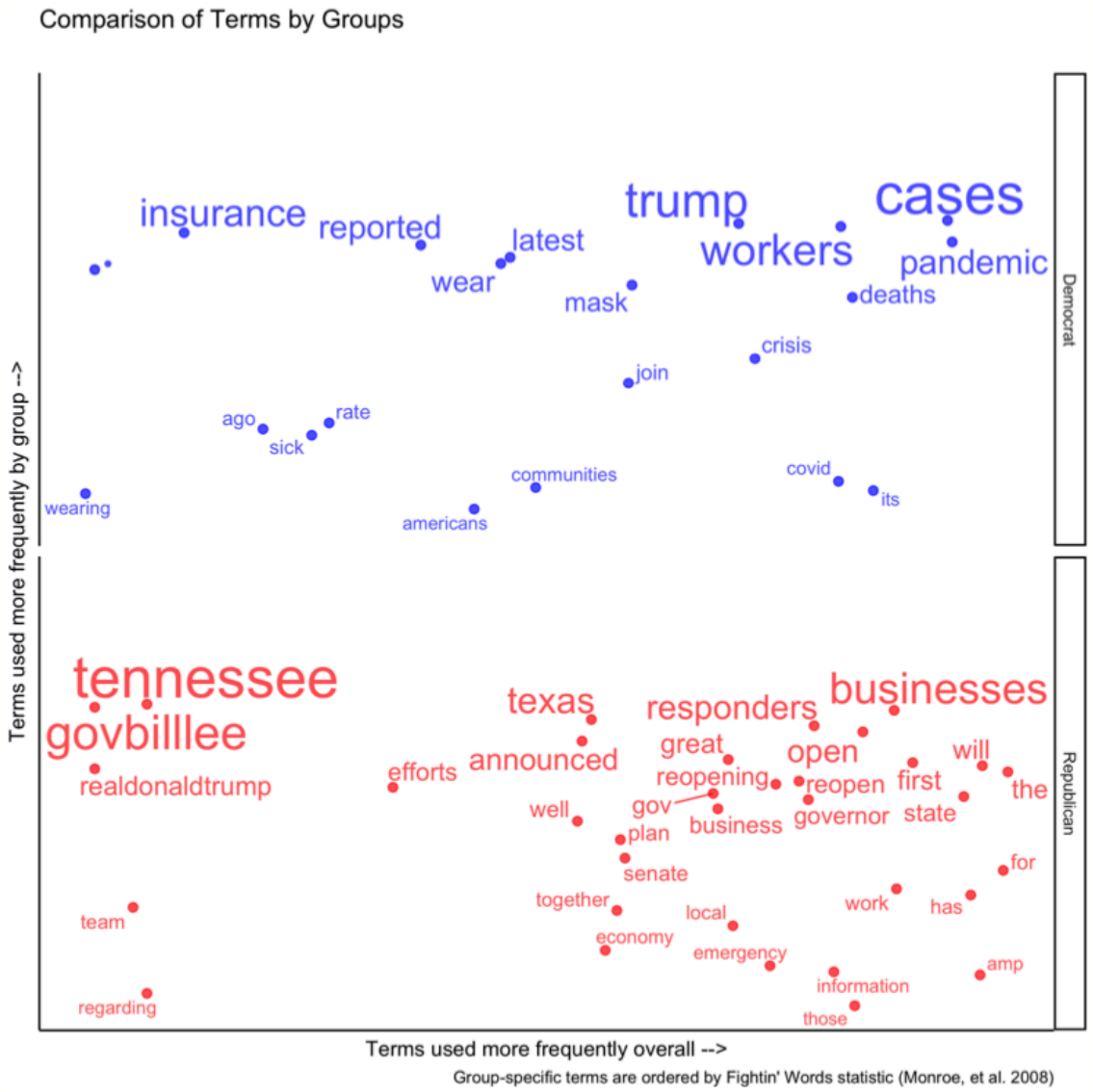
- Period (.), comma (,), apostrophe ('), quotation ("), question (?), exclamation (!), dash (–), ellipsis (...), colon (:), semicolon (;), etc.
- In many cases, these are (considered) unimportant
- Are they?

Text Normalization

Removing punctuation

- Punctuation carries important information
  - Exclamation mark (`!!!`), hashtags (`#metoo`), emojis (`:)`), etc.
- Punctuation itself can be of interest (studying writing styles)

Text Normalization



Text Normalization

Removing stop words

- Common words used across documents that do not give much information
- E.g., “and”, “the”, or “that”

Text Normalization

Removing stop words can spare much computational power

- C.f., Heaps' Law
- However, under what circumstances are these words *not* stop words? For instance, consider the case of `the`.

Text Normalization

Lemmatization

- Lemma: the base form
  - E.g., “run”
- Wordform: various forms derived the lemma
  - E.g., “runs”, “ran”, “running”
- Lemmatizatoin is the process of mapping words to their lemma

Text Normalization

Lemmatization

- Not always straightforward
  - Irregular variations E.g., “see-saw-seen”
  - Same token but different lemmas
    - E.g., he is “writing” an email vs. a nice piece of “writing”
- Necessitates a dictionary and POS (**p**art **o**f **s**peech) tagging

Text Normalization

Stemming is a popular approximation to lemmatization

- Simply discards the end of a word
  - E.g., family: famili
- Errors
  - E.g., “leav” for both “leaves” (as in “He leaves the room”) and “leaves” (as in parts of a plant)
- Various algorithms: *Porter*, *Lancaster*, etc.

Text Normalization

Filtering by frequency

- Too (in)frequent words across documents
  - E.g., stop words
- Can be filtered by the minimum/maximum document frequencies
  - Word that appear in fewer/more than *n*% of documents
- The rationale
  - Discriminatory power
  - Computational savings

Text Normalization

(How) Should we normalize?

- Difficult to know its consequences *a priori*
- Before analysis: carefully think about the pros and cons in each of the steps
- After analysis: conduct robustness check

Text Normalization

Normalization with LLMs

- With LLMs, traditional normalization steps are less relevant, especially due to the widespread adoption of subword-level tokenization (BPE, WordPiece, etc.)
- However, they are still relevant depending on the context

Text Normalization

Normalization with LLMs (cont’d)

- Lower-casing: different models deal with lower-casing differently
  - BERT-base-uncased vs. BERT-cased, GPT models, etc.
- Punctuation and stopwords: LLMs can handle them effectively by treating them as meaningful tokens
- Lemmatization/stemming: subword tokenization already handles word variations
  - See [AG] pp. 47–55 for various approaches
- White spaces: white space handling can matter where they represent structure
  - E.g., four white spaces as a single token representing an indentation should work better for code (see CodeBERT)

Text Representation

Levels of text representation

- Word representations
  - Static embeddings: Word2Vec, GloVe, FastText
    - The same embeddings for the `bank` in `river bank` and `bank account`
  - Contextual embeddings: neural embeddings from transformer models (BERT, GPT, LLama)
    - The embedding for the `bank` differs across contexts

Text Representation

Levels of text representation (cont'd)

- Representations beyond the word-level (sentences, paragraphs, or documents)
  - Lexical approaches
    - Bag of Words (BoW): basic word frequency representation
    - TF-IDF (Term-Frequency Inverse Document Frequency): weighting of words based on their importance to documents
  - Transformer-based neural embeddings can be used for representing entire sentences, paragraphs, or documents
    - There are models specifically tailored for sentence-level representations too (e.g., S-BERT)

The Bag of Words (BoW) model

The most basic text representation model

- A text is represented as a set of words that appear in it

I love this movie! It's sweet, but with satirical humor. The dialogue is great and the adventure scenes are fun... It manages to be whimsical and romantic while laughing at the conventions of the fairy tale genre. I would recommend it to just about anyone. I've seen it several times, and I'm always happy to see it again whenever I have a friend who hasn't seen it yet!

it	6
I	5
the	4
to	3
and	3
seen	2
yet	1
would	1
whimsical	1
times	1
sweet	1
satirical	1
adventure	1
genre	1
fairy	1
humor	1
have	1
great	1
…	…

The Bag of Words (BoW) model

Document-Feature Matrix (or Document-Term Matrix)

- Columns record features/terms (all types or $|V|$)
- Rows record documents
- Cells can be binary vectors or count vectors

The Bag of Words (BoW) model

An example corpus

- Doc 1: “The clever fox cleverly jumps over the lazy dog, showcasing its cleverness.”
- Doc 2: “Magic and mysteries mingle in the wizard’s daily musings, revealing mysteries unknown.”
- Doc 3: “Sunny days bring sunshine and sunsets, making sunny parks the best for sunny strolls.”

The Bag of Words (BoW) model

An example DFM

Document	clever	jumps	lazy	dog	magic	mysteries	…
Doc 1	3	1	1	1	0	0	…
Doc 2	0	0	0	0	1	2	…
Doc 3	0	0	0	0	0	0	…

The Bag of Words (BoW) model

An example DFM

Document	clever	jumps	lazy	dog	magic	mysteries	…
Doc 1	3	1	1	1	0	0	…
Doc 2	0	0	0	0	1	2	…
Doc 3	0	0	0	0	0	0	…

The Bag of Words (BoW) model

An example corpus

- Doc 1: “The `clever` fox `cleverly` jumps over the lazy dog, showcasing its `cleverness`.”
- Doc 2: “Magic and mysteries mingle in the wizard’s daily musings, revealing mysteries unknown.”
- Doc 3: “Sunny days bring sunshine and sunsets, making sunny parks the best for sunny strolls.”

The Vector Space Model

What is the vector space model?

- Each row (representing a text) in a DFM is a vector (an array of numbers) in a high-dimensional space
- The size of the dimension (the number of columns) is $|V|$
- Originates from IR (**I**nformation **R**etrieval)
  - See Turney and Pantel (2010) for details

Comparing Texts

With some form of DFM, we are ready to compare different documents

- “Similar” can mean different things
  - Sentiments, stances, themes, etc.
- There is no “correct” notion of similarity
- Yet there are metrics that are more or less effective across contexts

Cosine Similarity

We have two vectors (representing two documents), $\vec{A}$ and $\vec{B}$:

$$\vec{A} = [a_1, a_2, \ldots, a_n]$$

$$\vec{B} = [b_1, b_2, \ldots, b_n]$$

The inner product:

$$\vec{A} \cdot \vec{B} = (a_1 \times b_1) + (a_2 \times b_2) + \ldots + (a_n \times b_n)$$

Cosine Similarity

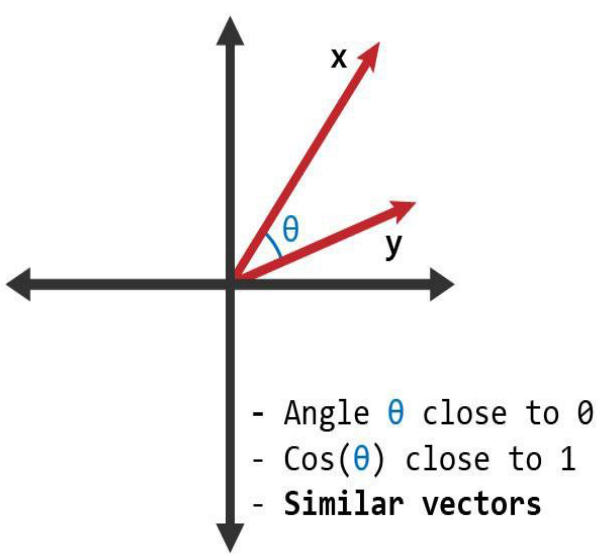
Cosine similarity between vectors $\vec{A}$ and $\vec{B}$ is given by:

$$\text{Cosine Similarity}(A, \vec{B})\vec{} = \frac{A \vec{\cdot} B\vec{}}{\|A\|\vec{\|}\|B\|\vec{}}$$

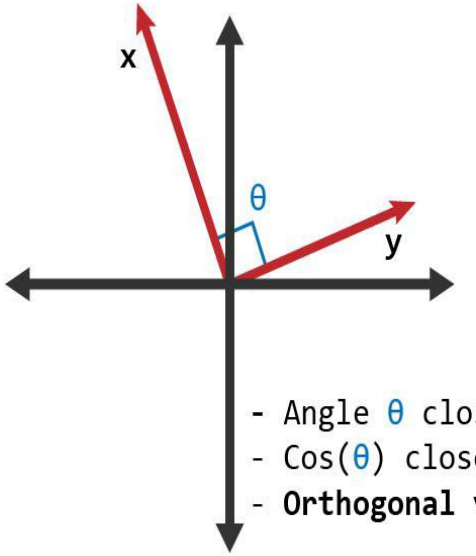
$A\vec{} \cdot B\vec{}$ is the inner product, and $\|A\vec{}\|$ and $\|B\vec{}\|$ are defined as

$$\|A\|\vec{} = \sqrt{a_1^2 + a_2^2 + \ldots + a_n^2} \|B\|\vec{} = \sqrt{b_1^2 + b_2^2 + \ldots + b_n^2}$$

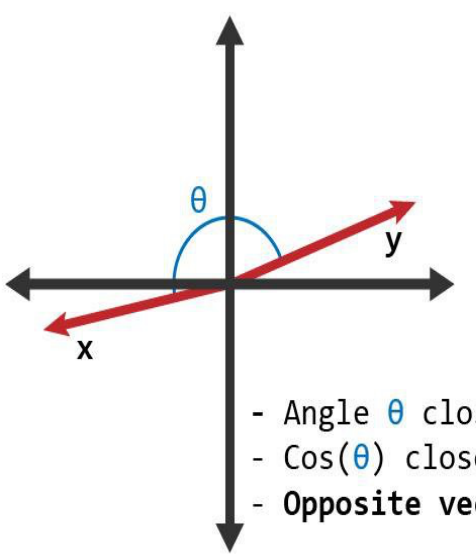
Cosine Similarity



- Angle θ close to 0
- Cos(θ) close to 1
- **Similar vectors**

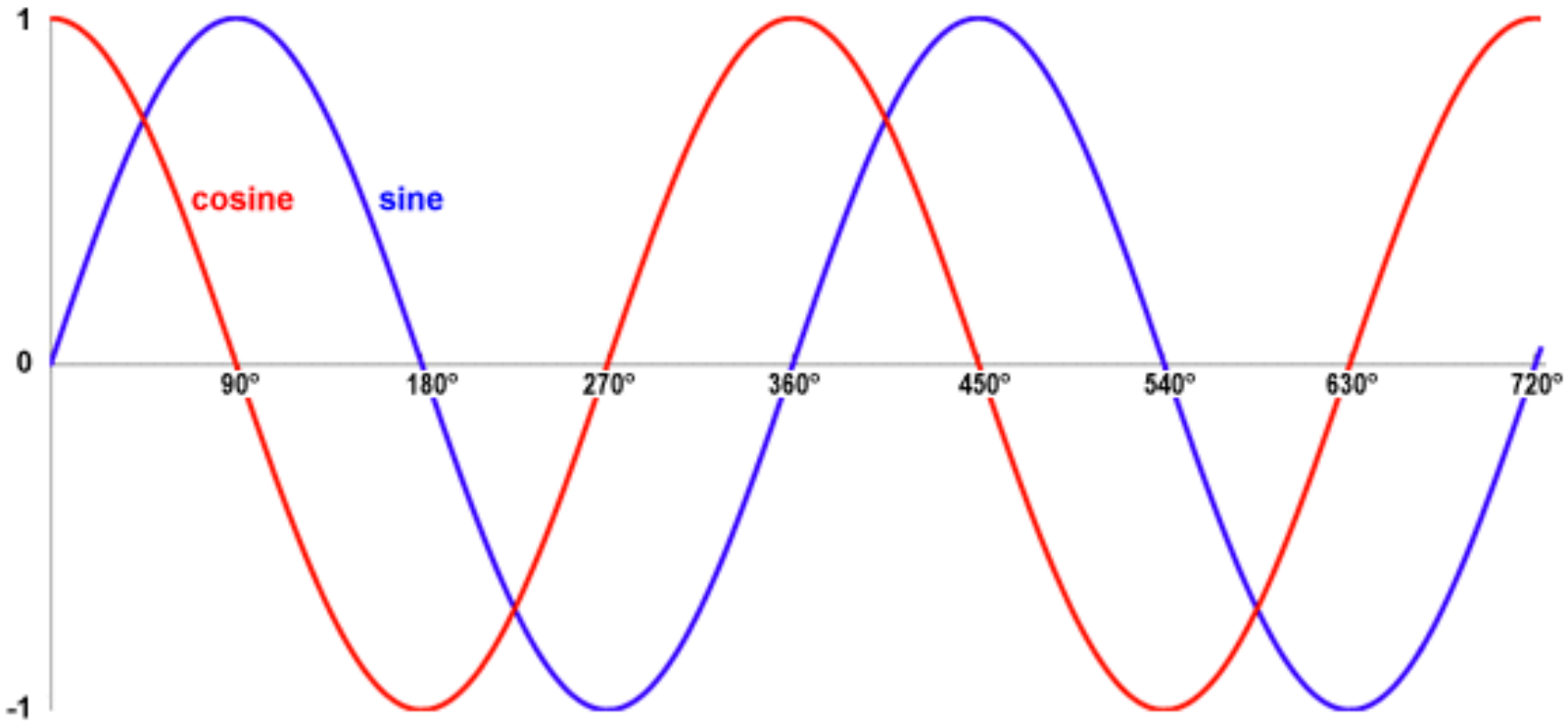


- Angle θ close to 90
- Cos(θ) close to 0
- **Orthogonal vectors**



- Angle θ close to 180
- Cos(θ) close to -1
- **Opposite vectors**

Cosine Similarity



TF-IDF Weighting

TF (Term Frequency) - IDF (Inverse Document Frequency)

- Count vectors consider the frequencies of words
- However, some words are too frequent across different documents
  - E.g., *the*, *a*, *an*, etc.
- We want to weight how unique a word to a document

TF-IDF Weighting

TF-IDF is a numerical statistic that reflects *how important a word is to a document* in a corpus.

TF-IDF Weighting

The **TF-IDF** value is obtained by multiplying **TF** (**T**erm **F**requency) and **IDF** (**I**nverse **D**ocument **F**requency) for a term in a document, highlighting the importance of rare terms

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

TF-IDF Weighting

Term Frequency

- Reflects how frequently a term occurs in a document, normalized by the document length

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

TF-IDF Weighting

Inverse Document Frequency

- Scales down terms that occur very frequently across the corpus and are less informative

$$\text{IDF}(t, D) = \log\left(\frac{\text{Total number of documents } D}{\text{Number of documents with term } t \text{ in it} + 1}\right)$$

TF-IDF Weighting

Many versions of TF-IDF: link

Count Vectors Vs. TF-IDF Vectors

Count Vectors

Term	can	you	fly	sleep
‘can you fly’	1	1	1	0
‘can you sleep’	1	1	0	1

TF-IDF Vectors

Term	can	you	fly	sleep
‘can you fly’	0.5	0.5	0.7	0
‘can you sleep’	0.5	0.5	0	0.7

Word Representation

How do we represent *words*?

- Vector semantics: a method that represents texts in a multi-dimensional space
- The simplest approach: one-hot encoding
  - A vector with one dimension per unique word in the vocabulary
  - Records 1 for that word and 0 for all the others
  - E.g., `author` = (0, 0, 0, 0, 1, …, 0, 0) (the dimension size is $|V|$)

Word Representation

Limitations of one-hot encoding

- Semantics
  - Similarity: one-hot(`author`) $\perp$ one-hot(`writer`)
- Computation
  - Sparsity (mostly 0s in huge dimensional space: $|V|$)

Word Representation

Term-document matrix (TDM)

- Rows represents words, and columns represent documents
- Similar words have similar vectors because they tend to occur in similar documents (documents are the context)
- E.g., four words in four Shakespeare plays ([JM] Chp. 6)

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

Figure 6.5 The term-document matrix for four words in four Shakespeare plays. The red boxes show that each word is represented as a row vector of length four.

Word Representation

Term-term matrix (TTM)

- Dimension: $|V| \times |V|$
- Each cell records the number of times the row word and the column word co-occur in some context
- Contexts are often a window around the word (e.g., $\pm$ 5)

	aardvark	...	computer	data	result	pie	sugar	...
cherry	0	...	2	8	9	442	25	...
strawberry	0	...	0	0	1	60	19	...
digital	0	...	1670	1683	85	5	4	...
information	0	...	3325	3982	378	5	13	...

Figure 6.6 Co-occurrence vectors for four words in the Wikipedia corpus, showing six of the dimensions (hand-picked for pedagogical purposes). The vector for *digital* is outlined in red. Note that a real vector would have vastly more dimensions and thus be much sparser.

Word Embeddings

What are word embeddings?

- *Dense* vectors representing word meanings in a multi(low)-dimensional space
  - Word embeddings $\subset$ word vectors
  - Traditional embeddings: d = 50–
  - Neural embeddings: d = 500– (e.g., GPT-3: 12,288)
- Words are “embedded” into a low-dimensional space

Word Embeddings

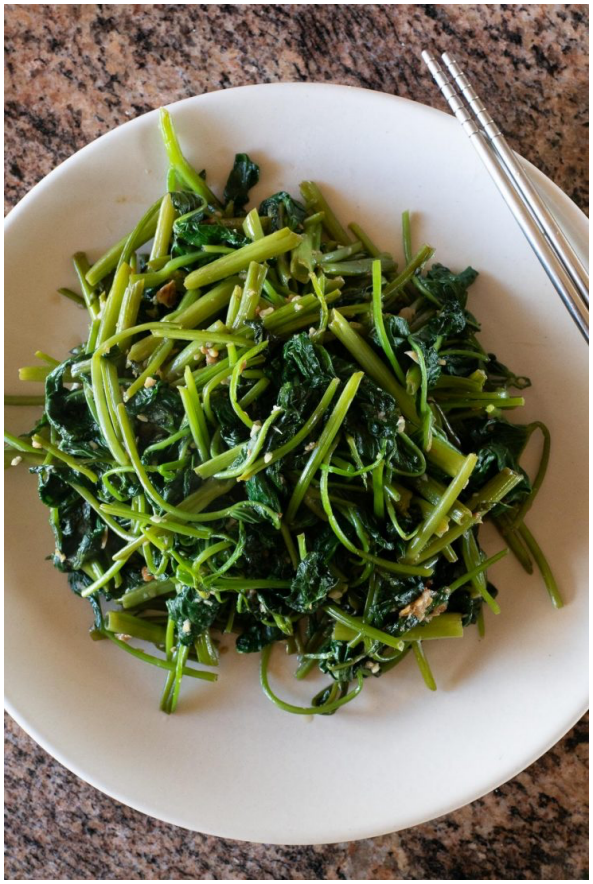
What are word embeddings? (cont'd)

- Distributional hypothesis (Joos 1950; Harris 1954)
  - Word that occur in similar contexts tend to have similar meanings
  - "You shall know a word by the company it keeps" (Firth 1957)
- E.g., oculists & eye-doctor: eyes, examine, diagnose, patient, etc.

Word Embeddings

If we have seen

- "… spinach sauteed with garlic over rice …"
- "… chard stems and leaves are delicious …"
- "… collard greens and other salty leafy greens …"



We can guess what `ongchoi` is

- `ongchoi` is delicious sauteed with garlic
- `ongchoi` is superb over rice
- `ongchoi` leaves with salty sauces

Word Embeddings

Why useful?

- Direct object of interest (to study word usage and meaning)
- Downstream tasks: feature representations
  - Text classification
  - Part of speech tagging
  - Named entity recognition
  - Etc.

Word Embeddings

Why useful?

- A measure of word meaning

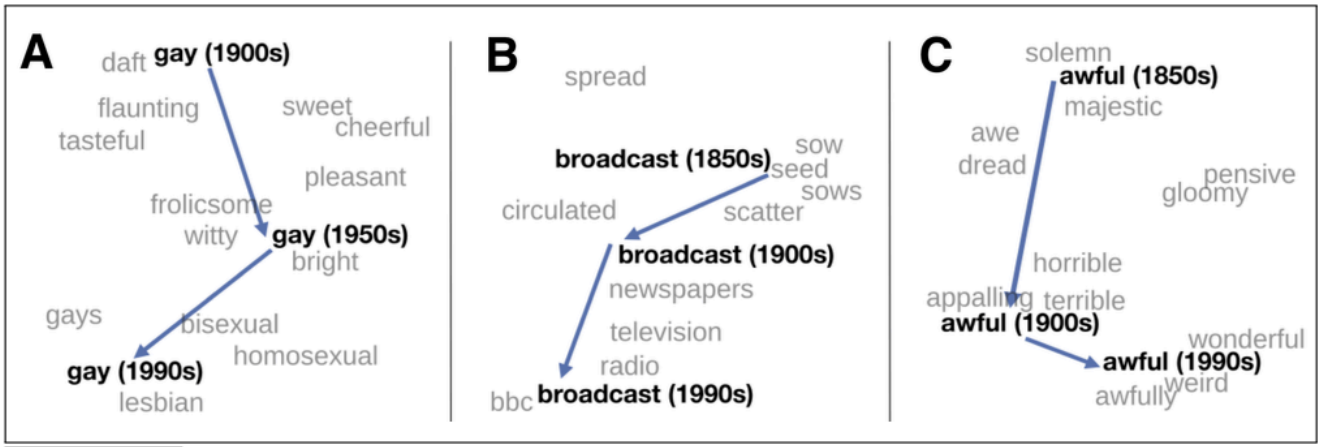
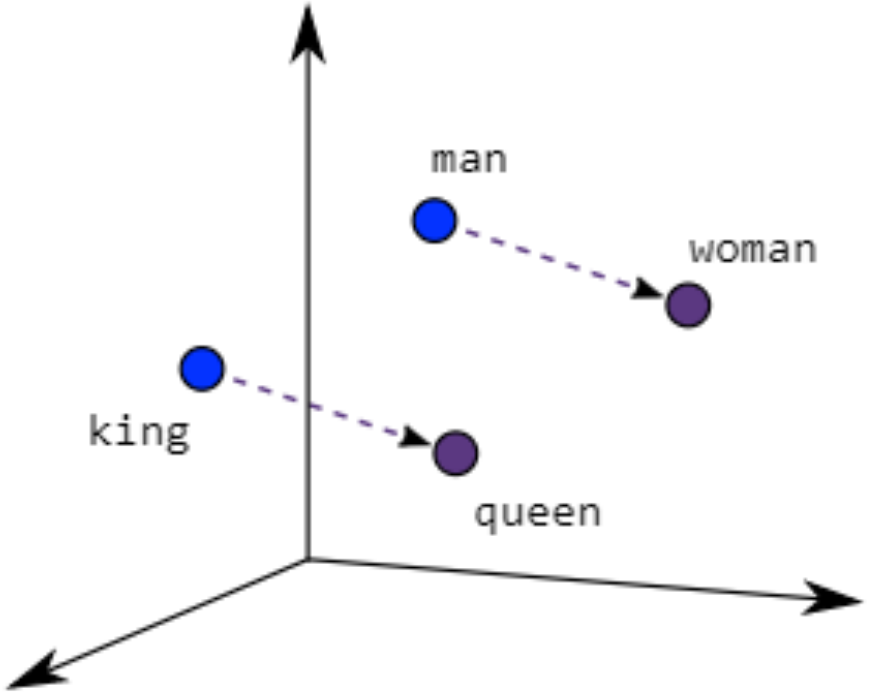


Figure 6.17 A t-SNE visualization of the semantic change of 3 words in English using word2vec vectors. The modern sense of each word, and the grey context words, are computed from the most recent (modern) time-point embedding space. Earlier points are computed from earlier historical embedding spaces. The visualizations show the changes in the word *gay* from meanings related to "cheerful" or "frolicsome" to referring to homosexuality, the development of the modern "transmission" sense of *broadcast* from its original sense of sowing seeds, and the pejoration of the word *awful* as it shifted from meaning "full of awe" to meaning "terrible or appalling" (Hamilton et al., 2016).

Word Embeddings

Why useful?

- Encoding similarity
  - For similar words, their embeddings point in similar directions ($\iff$ one-ho encodings)
    - E.g., $\overrightarrow{e_{author}} \propto \overrightarrow{e_{writer}}$
  - Similarity in relations ("vector arithmetic")
    - E.g., `king` - `man` + `woman` $\approx$ `queen` (Mikolov et al. 2013)



Word Embeddings

Why useful?

- Automatic generalization
  - Information retrieval
    - E.g., identifying academic papers about literacy in the digital age
      - Seed keywords: `digital literacy`, `information literacy`, etc.
      - Identifying similar words using word embeddings: `e-literacy`, `technology proficiency`, etc.

Word Embeddings

Why useful?

- Automatic generalization (cont'd)
  - Dictionaries combined with word embeddings (Garten et al. 2018)
  - Osnabrugge et al. (2021): measuring emotive rhetoric from political speech

$$\text{AvgSim}_{\text{emotive}}(w) = \frac{1}{N_{emotive}} \sum_{i=1}^{N_{emotive}} \cos(w, \overrightarrow{e_i})$$

$$\text{AvgSim}_{\text{neural}}(w) = \frac{1}{N_{neutral}} \sum_{i=1}^{N_{neutral}} \cos(w, \overrightarrow{e_i})$$

$$\text{Emotiveness}(w) = \text{AvgSim}_{\text{emotive}}(w) - \text{AvgSim}_{\text{neutral}}(w)$$

Estimating Word Embeddings

Word2Vec (Mikolov et al. 2013a; Mikolov et al. 2013b)

- Skipgram and CBOW (Continuous Bag Of Words)
  - Skipgram: given a target word, predict the context words (e.g., $\pm$ 5)
  - CBOW: given the context words, predicts the target word
- SGNS (skip-gram with negative sampling)
  - Given a pair of a target word and another word c, what is the probability of c being the actual context word (c_{pos})?

Estimating Word Embeddings

Word2Vec SGNS

- Self-supervision: “+” if in context, otherwise “-”
  - L: the size of the context window
  - K: the proportion of positive (or context) to (randomly selected) negative examples (recommended K: 2–5 for big, 5–20 for small data)

```
... lemon,  a [tablespoon of apricot jam,     a] pinch ...
               c1         c2    w     c3       c4
```

positive examples +

w	c_{pos}
apricot	tablespoon
apricot	of
apricot	jam
apricot	a

negative examples -

w	c_{neg}	w	c_{neg}
apricot	aardvark	apricot	seven
apricot	my	apricot	forever
apricot	where	apricot	dear
apricot	coaxial	apricot	if

Estimating Word Embeddings

Word2Vec SGNS

- Task
  - Train a binary classifier that computes Pr(+|w, c)
  - Pr(+|w, c) = $\sigma(\overrightarrow{e_w} \cdot \overrightarrow{e_c})$
- Goal
  - Maximize the similarity of the target-context pairs (w, c_{pos})
  - Minimize the similarity of the target-non-context pairs (w, c_{neg})

Estimating Word Embeddings

Word2Vec SGNS

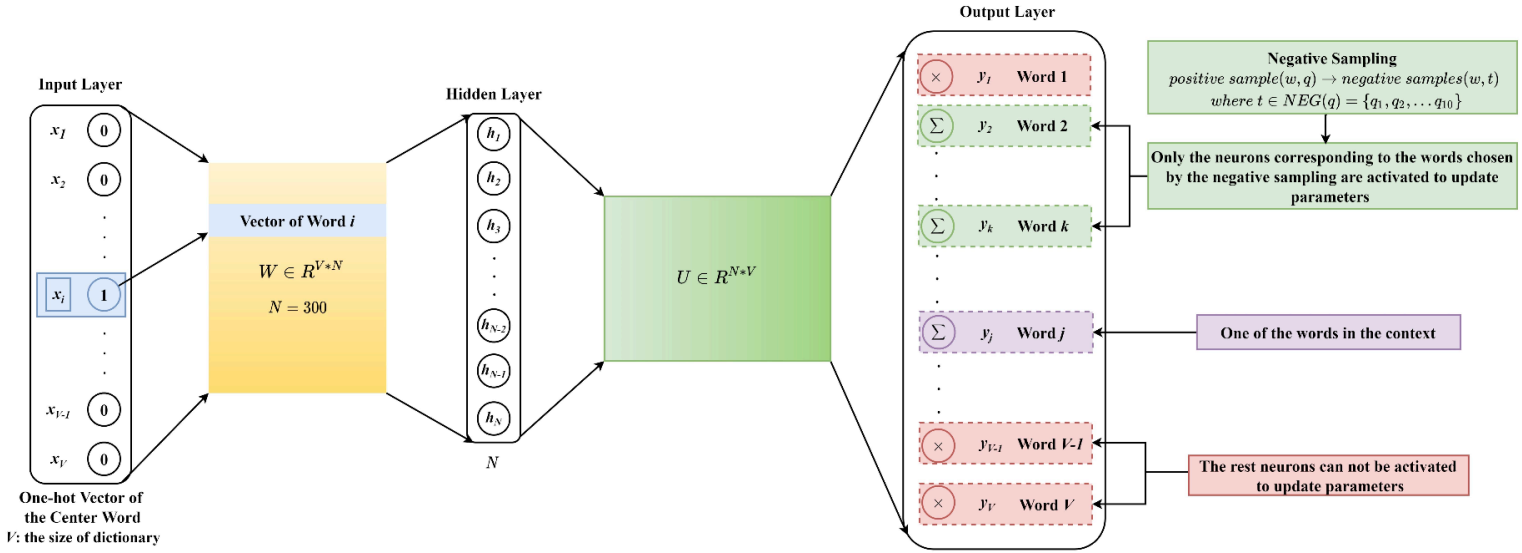
- Optimization: minimize the cross-entropy loss function using (stochastic) gradient descent

$$L_{CE} = -log[P(+|w, c_{pos}) \prod_{i=1}^{k} P(-|w, c_{neg_i})]$$

Estimating Word Embeddings

Word2Vec SGNS

- The neural network for SG(NS) (source: link)



Estimating Word Embeddings

Word2Vec SGNS

- Detailed treatments of SG and SGNS
  - SG: link
  - SGNS: link

Various Approaches

Different approaches to obtaining word embeddings

- GloVe (Global Vectors for Word Representation) (Pennington et al. 2014)
- FastText (Bojanowski et al. 2017)
  - Subword-level model
    - Each word is represented as itself along with a bag of constituent n-grams, with boundary symbols < and >
    - E.g., $\overrightarrow{e_{apple}} = \overrightarrow{e_{<ap}} + \overrightarrow{e_{app}} + \overrightarrow{e_{ppl}} + \overrightarrow{e_{ple}} + \overrightarrow{e_{le>}} + \overrightarrow{e_{<apple>}}$
  - Deals with OOV (out of vocabulary), rare words, and typos (e.g., `appple`) efficiently

Pre-trained Embeddings

General embeddings

- Word2Vec: link ("GoogleNews-vectors-negative300.bin.gz")
- GloVe: link
- FastText: link

(A few examples from many) domain-specific embeddings

- Trained on 19th-century British newspapers: link
- Trained on tweets: link ("glove.twitter.27B.zip")

Static vs. Contextual Embeddings

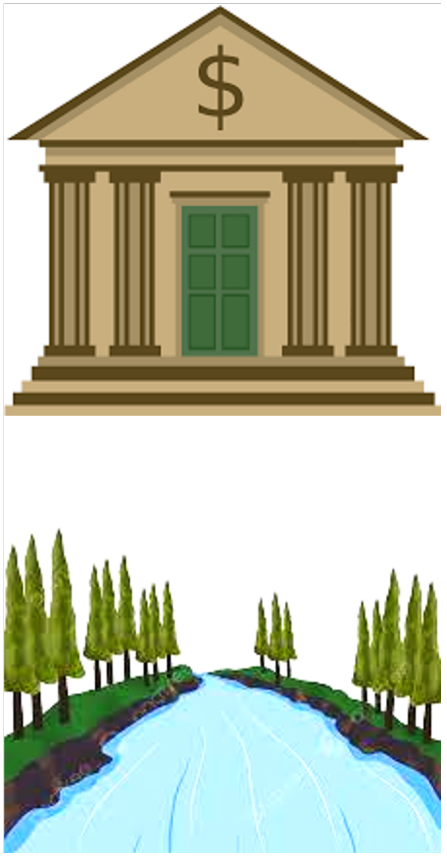
From contexts but not contextual

- Word2Vec, GloVe, and Fast2Text are *static* embeddings
- A word's embedding *is* derived from its context
- Different contexts do *not* lead to different embeddings
- However, a word's meaning differs depending on the context even if a word has the same form

Static vs. Contextual Embeddings

Example

- Embeddings for the same word differ by context
- Sentence 1: *Open a bank account*
- Sentence 2: *On the river bank*



Static vs. Contextual Embeddings

Modern contextual embeddings are based on neural network models with **self-attention**

- The primary goal of self-attention is to compute contextualized representations of each token in the sequence
- Self-attention allows each token in the sequence to 'attend' to (or reference) all other parts of the sequence (including self)
- This allows for capturing contextual meanings of tokens
  - E.g., "The `dog` in the yard started to bark because `it` was hungry"

Static vs. Contextual Embeddings

Previous example

- *Open a bank account* $\rightarrow e_{bank_{s1}}$: [0.3, 0.9, …]
- *On the river bank* $\rightarrow e_{bank_{s2}}$: [0.8, 0.1, …]
- $e_{bank_{s1}}$ should be similar to $e_{account_{s1}}$, and $e_{bank_{s2}}$ should be similar to $e_{river_{s2}}$

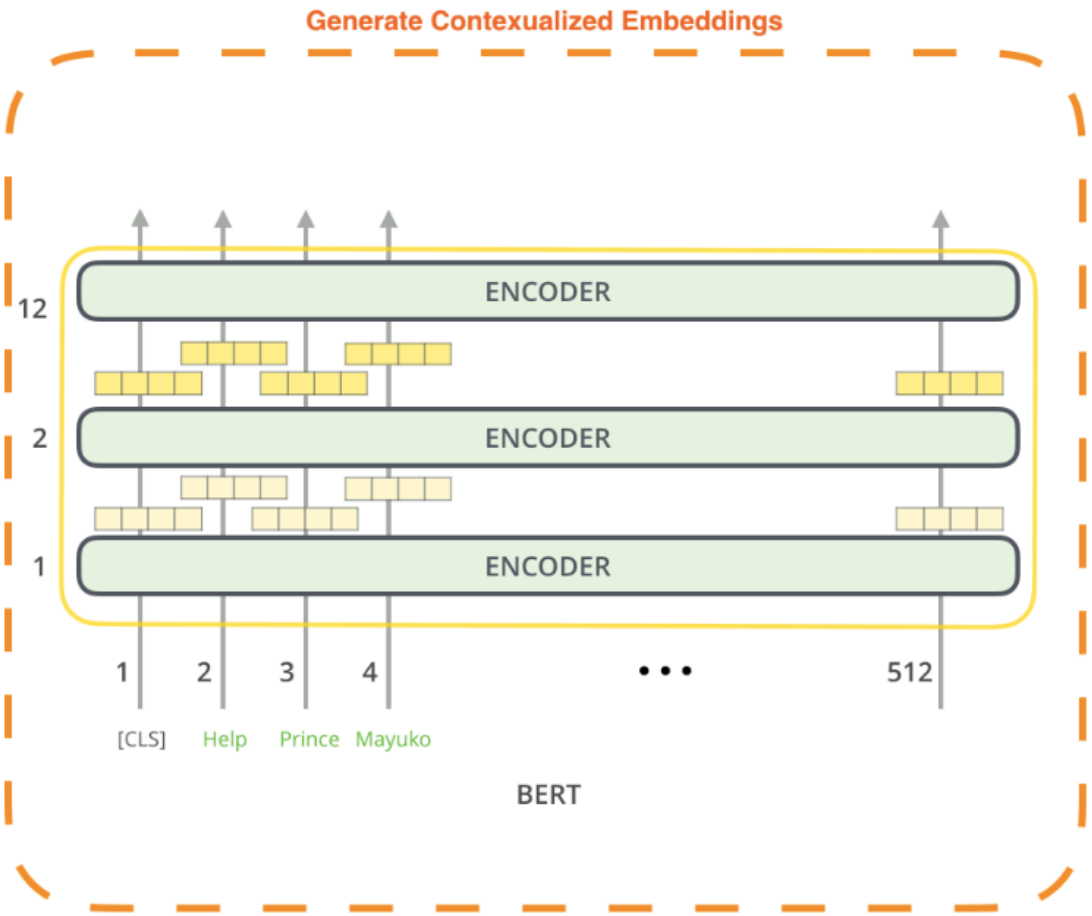
Static vs. Contextual Embeddings

Bi-directional **E**ncoder **R**epresentations from **T**ransformers

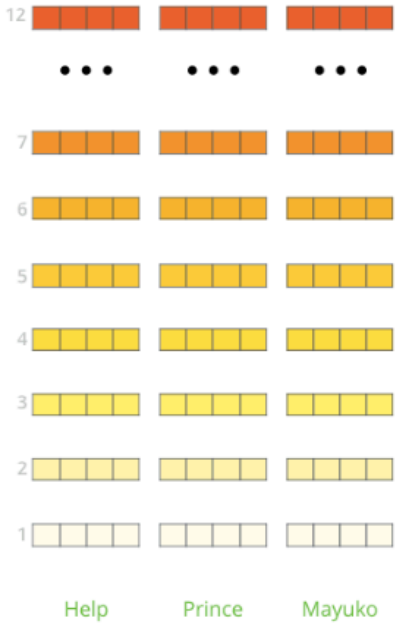
- A form of transformer (a type of neural network architecture with self-attention mechanisms)
- An encoder model (processing an input sequence and transforms it into an embedding)
- (Jointly) trained on huge data sets (BookCorpus and Wikipedia) for two tasks: masked language modeling & next sentence prediction
- Two versions of the model introduced in the original paper
  - BERT BASE (12 encoder stacks)
  - BERT LARGE (24 encoder stacks)

Static vs. Contextual Embeddings

Bi-directional Encoder Representations from Transformers



The output of each encoder layer along each token's path can be used as a feature representing that token.



But which one should we use?

Static vs. Contextual Embeddings

What to use?

- If your analysis is about words—especially general or out-of-context meanings—static embeddings are often sufficient and computationally efficient
- If your task involves meaning in context or sentence-level analysis (e.g., classification, topic modeling, semantic similarity), contextual embeddings are typically more appropriate.

Pre-trained vs. Self-trained

A few circumstances that require self-training

- Temporal changes in language
  - Known as diachronic/dynamic embeddings (e.g., Kim and Jeon 2023)
- Group-specific language (e.g., Democrats vs. Republicans)
- Domain-specific language (e.g., Case2Vec)
- Low resource languages

Pre-trained vs. Self-trained

Which one captures word similarity better?

- Experiment by Rodriguez and Spirling (2022)
  - Provide crowd workers (human annotators) on MTurk 10 political words and ask them to produce a set of ten nearest neighbors (“human”)
  - They then use these same words to generate machine nearest neighbors by finding the most cosine similar vector using word embeddings (“local” or pre-trained “GloVe”)
  - They then have a separate set of humans (human judge) look at a prompt word and two possible nearest neighbors (“human” vs. “local” vs. pre-trained “GloVe”)

Pre-trained vs. Self-trained

Findings from Rodriguez and Spirling (2022)

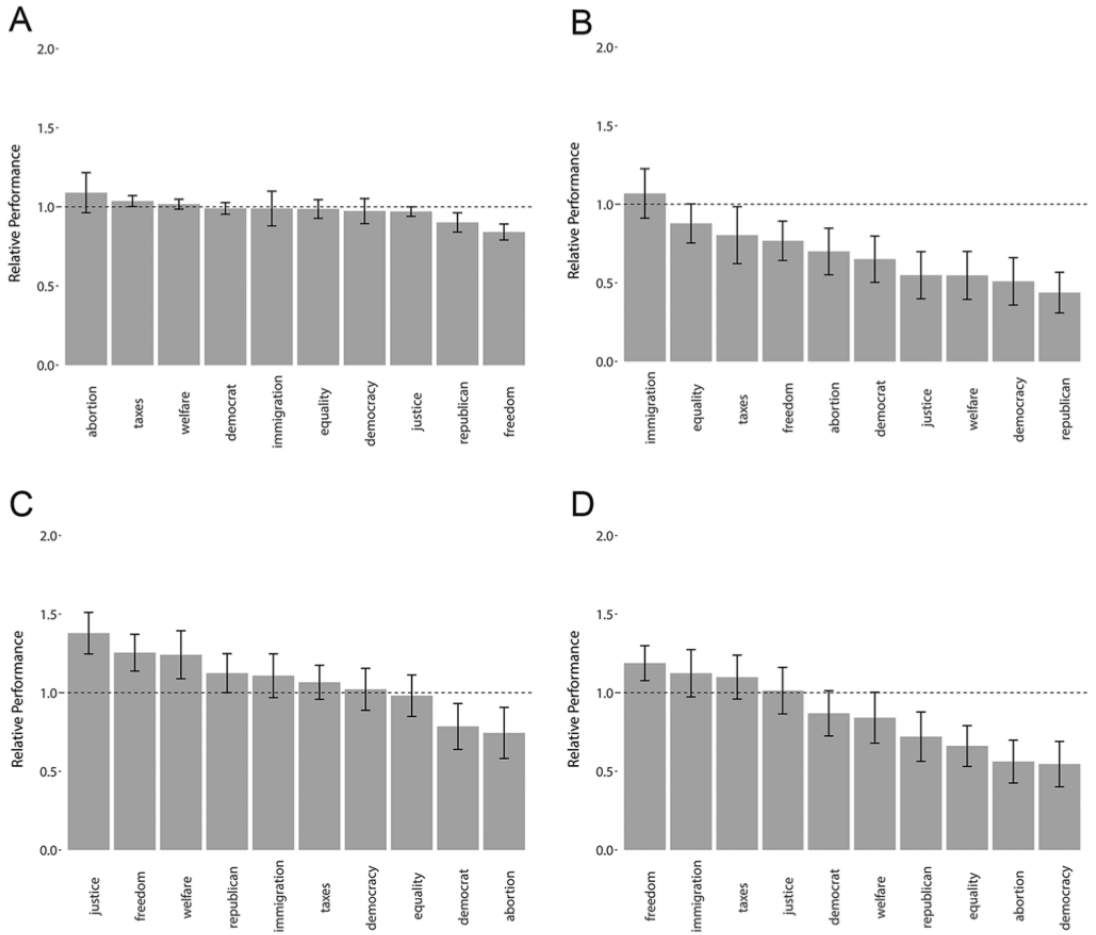


Figure 2. Human preferences: Turing assessment. *A*, Candidate: local 48–300; baseline: local 6–300. *B*, Candidate: local 6–300; baseline: human. *C*, Candidate: GloVe; baseline: local 6–300. *D*, Candidate: GloVe; baseline: human.

Pre-trained vs. Self-trained

Findings from Rodriguez and Spirling (2022)

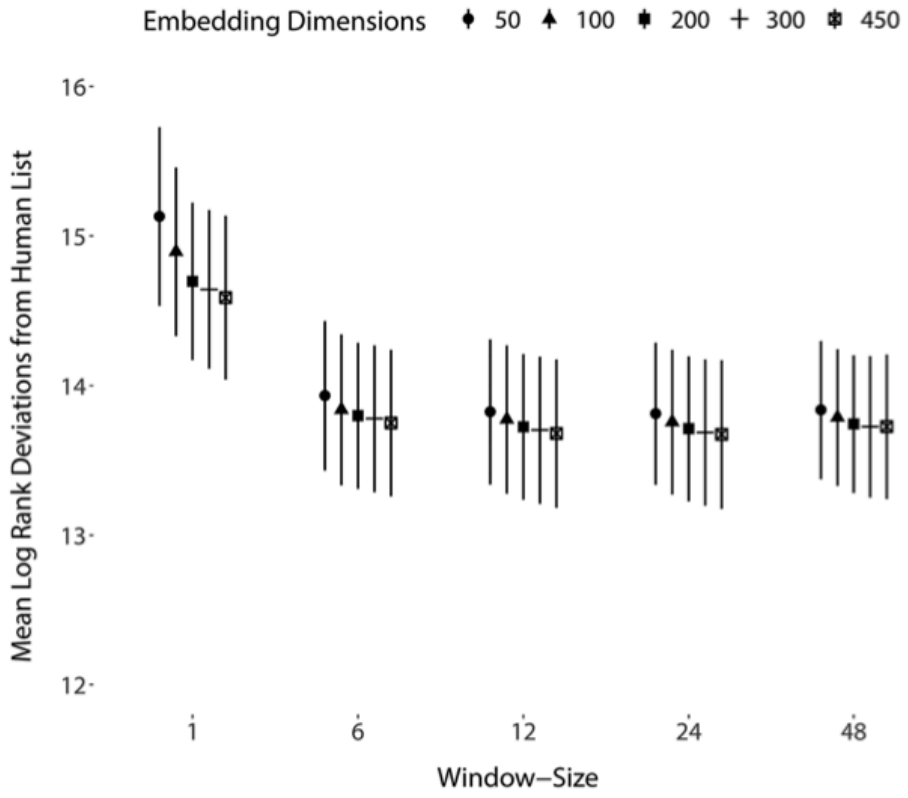


Figure 3. Human preferences: log rank deviations: complex models come closer to "human" assessments, but medium-size models are almost as good as very large ones.

Pre-trained vs. Self-trained

Findings from Rodriguez and Spirling (2022)

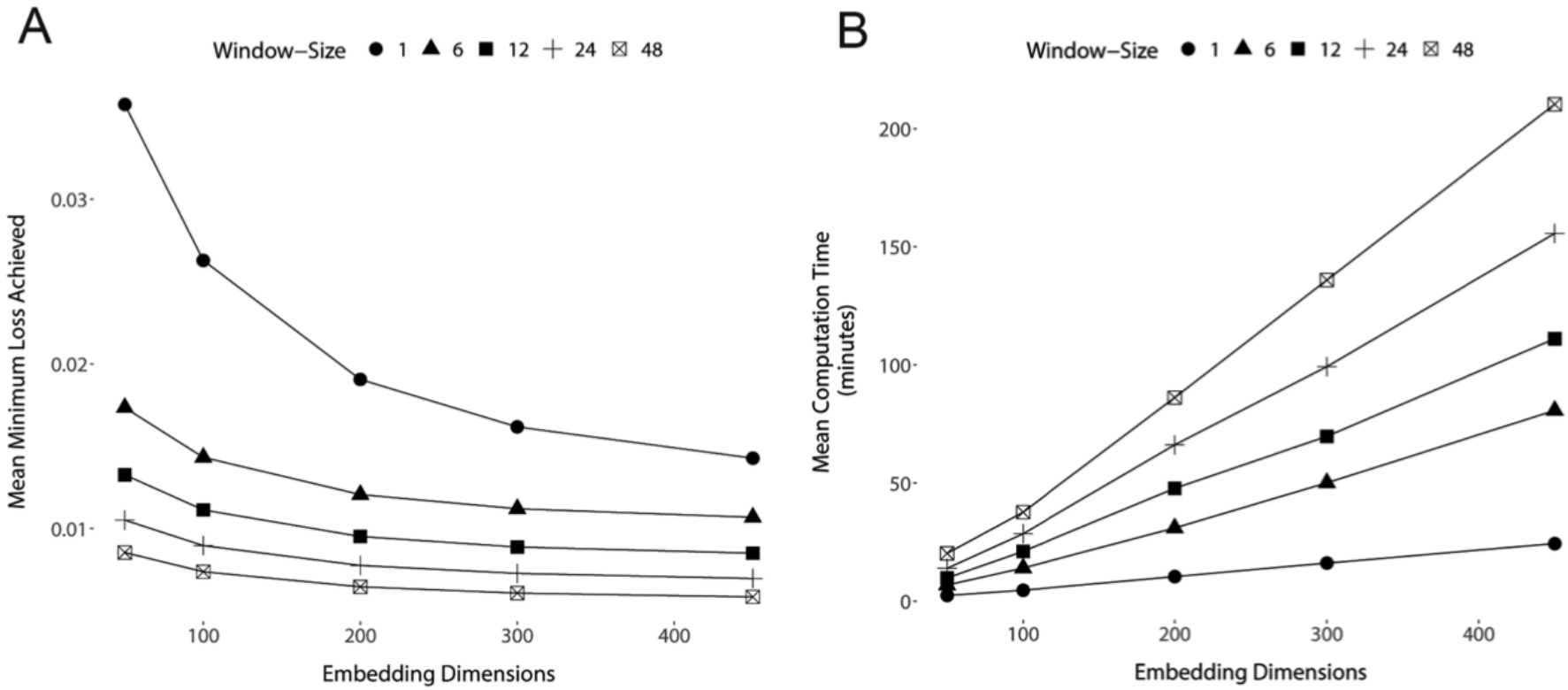


Figure 4. Technical criteria: larger models fit better but take longer to compute. *A*, Mean minimum loss achieved. *B*, Computation time (minutes)

Pre-trained vs. Self-trained

Lessons

- Popular pre-trained word embeddings “perform” at a level close to–or even surpassing–both human annotators and locally fit models with various configurations

Caveat

- A specific meaning of “perform”
- The results are based on a limited set of corpora (political in nature)

Guided Coding

- Normalization, representation, and comparison
- Word2Vec and FastText
- Exploring BERT
- Measuring moral rhetoric with word embeddings