Lecture 10: Topic Modeling, Word2Vec

LING-351 Language Technology and LLMs

Instructor: Hakyung Sung

September 25, 2025

Table of contents

Review

Python tutorial so far

- Tokenization
- Lemmatization
- Frequency calculation
- Concordance
- (Collocation)

Word distributions

Word distributions

Examining word distributions is the first and most important step in corpus/text analysis.

Zipf's power law (1932)

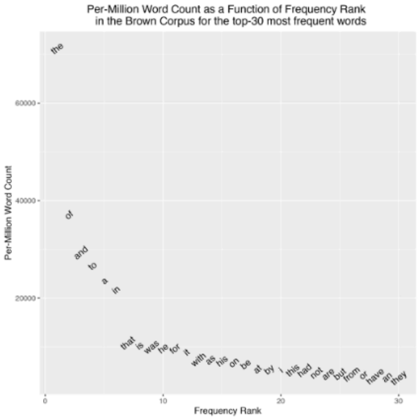


Figure 4.3: Per-million-word frequency of words in the Brown Corpus as a function of their frequency rank (ordered from left to right as the first most frequent word, the second most frequent, and so on).

Zipf's power law (1932)

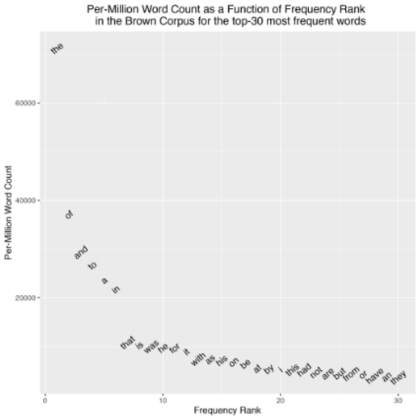


Figure 4.3: Per-million-word frequency of words in the Brown Corpus as a function of their frequency rank (ordered from left to right as the first most frequent word, the second most frequent, and so on).

Implication: Few words are very frequent; many are rare ⇒ long tail.

Zipf's power law (1932)

- type vs. token

Zipf's power law (1932)

- type vs. token
- frequency $\propto$ 1/rank.

Zipf's power law (1932)

- type vs. token
- frequency $\propto 1/\text{rank}$.
- e.g., Brown corpus: *the* $\approx$ 6% tokens; *of* $\approx$ 3%; *and* $\approx$ 2.6%.

Zipf's brevity law

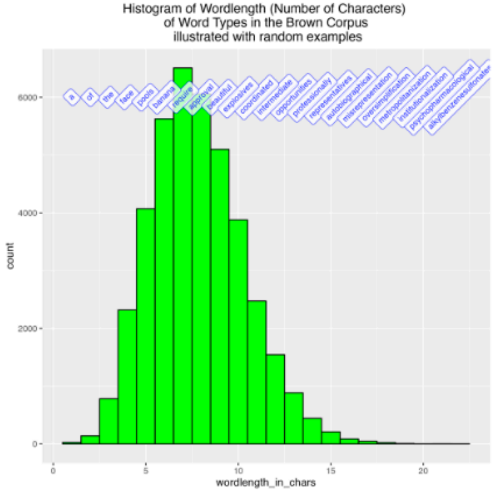


Figure 4.4: Histogram of the length (number of characters) of all word types in the Brown Corpus.

Zipf's brevity law

- More frequent words tend to be **shorter** (characters/syllables).

Zipf's brevity law

- More frequent words tend to be **shorter** (characters/syllables).
- Efficiency pressure: frequent items economize articulatory/processing effort.

Zipf's brevity law

- More frequent words tend to be **shorter** (characters/syllables).
- Efficiency pressure: frequent items economize articulatory/processing effort.
- Most frequent Brown words: monosyllabic, $\leq$3 letters (*the, of, and, a, in, to, is, was, I, for*).

Heaps' law

As you read more tokens in a corpus, you keep seeing new word **types**, but the **rate** of new words **slows down**.

Heaps' law

As you read more tokens in a corpus, you keep seeing new word **types**, but the **rate** of new words **slows down**.

Example

- After 1,000 tokens: $\sim$700 unique words

Why it matters?

Heaps' law

As you read more tokens in a corpus, you keep seeing new word **types**, but the **rate** of new words **slows down**.

Example

- After 1,000 tokens: $\sim$700 unique words
- After 10,000 tokens: not 7,000, but maybe $\sim$2,500–3,500

Why it matters?

Heaps' law

As you read more tokens in a corpus, you keep seeing new word **types**, but the **rate** of new words **slows down**.

Example

- After 1,000 tokens: $\sim$700 unique words
- After 10,000 tokens: not 7,000, but maybe $\sim$2,500–3,500

Why it matters?

- Estimate how much data you need before vocabulary “stabilizes”

Heaps' law

As you read more tokens in a corpus, you keep seeing new word **types**, but the **rate** of new words **slows down**.

Example

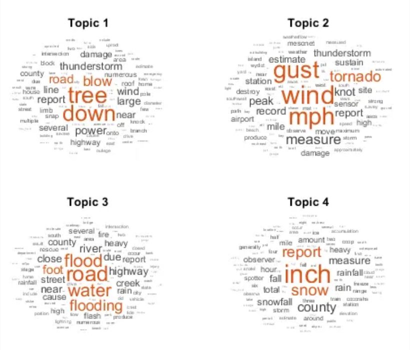
- After 1,000 tokens: $\sim$700 unique words
- After 10,000 tokens: not 7,000, but maybe $\sim$2,500–3,500

Why it matters?

- Estimate how much data you need before vocabulary "stabilizes"
- Reminds us that growth is **sublinear**

Topic modeling

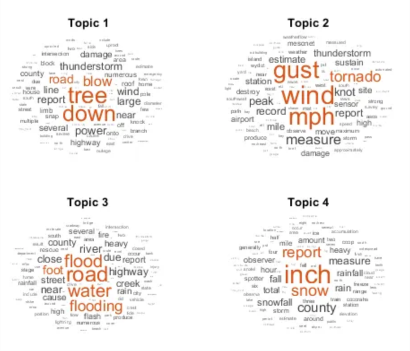
- A topic modeling is a type of **statistical modeling** for discovering the **abstract** topic that occur in a collection of documents.



Sourced from *https://medium.com/analytics-vidhya/*

Topic modeling

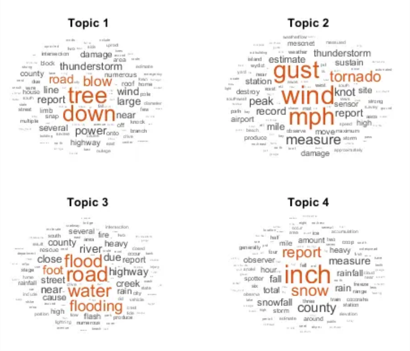
- A topic modeling is a type of **statistical modeling** for discovering the **abstract** topic that occur in a collection of documents.
- **Exploratory**: No annotated labels; discover latent structure using word frequencies/distributions



Sourced from *https://medium.com/analytics-vidhya/*

Topic modeling

- A topic modeling is a type of **statistical modeling** for discovering the **abstract** topic that occur in a collection of documents.
- **Exploratory**: No annotated labels; discover latent structure using word frequencies/distributions
- Classic model: LDA (Blei et al., 2003).



Sourced from `https://medium.com/analytics-vidhya/`

LDA: minimal workflow

Key idea: (1) Each document is a *mixture of topics*. (2) Each topic is a *distribution over words*. (3) Given only the words, LDA uses Bayesian inference to approximate the hidden topic structure.

1. Collect documents; tokenize, lemmatize; remove stop words.

LDA: minimal workflow

Key idea: (1) Each document is a *mixture of topics*. (2) Each topic is a *distribution over words*. (3) Given only the words, LDA uses Bayesian inference to approximate the hidden topic structure.

1. Collect documents; tokenize, lemmatize; remove stop words.
2. Choose K topics; start with random assignments.

LDA: minimal workflow

Key idea: (1) Each document is a *mixture of topics*. (2) Each topic is a *distribution over words*. (3) Given only the words, LDA uses Bayesian inference to approximate the hidden topic structure.

1. Collect documents; tokenize, lemmatize; remove stop words.
2. Choose K topics; start with random assignments.
3. Iterate: update topic assignments using document–topic and topic–word counts.

LDA: minimal workflow

Key idea: (1) Each document is a *mixture of topics*. (2) Each topic is a *distribution over words*. (3) Given only the words, LDA uses Bayesian inference to approximate the hidden topic structure.

1. Collect documents; tokenize, lemmatize; remove stop words.
2. Choose K topics; start with random assignments.
3. Iterate: update topic assignments using document–topic and topic–word counts.
4. Inspect top words per topic; give each topic a human-readable label.

LDA: minimal workflow

Key idea: (1) Each document is a *mixture of topics*. (2) Each topic is a *distribution over words*. (3) Given only the words, LDA uses Bayesian inference to approximate the hidden topic structure.

1. Collect documents; tokenize, lemmatize; remove stop words.
2. Choose K topics; start with random assignments.
3. Iterate: update topic assignments using document–topic and topic–word counts.
4. Inspect top words per topic; give each topic a human-readable label.
5. We'll do some hands-on practice with topic modeling.

Word vectors

Encoding

- Words themselves cannot be given as inputs to computers

Encoding

- Words themselves cannot be given as inputs to computers
- BUT, numbers can be given as inputs to computers

Encoding

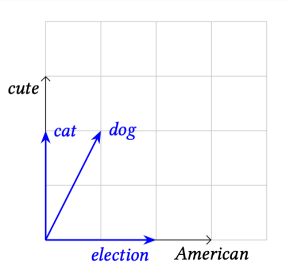
- Words themselves cannot be given as inputs to computers
- BUT, numbers can be given as inputs to computers
- Encoding = converting words to vectors

Encoding

- Words themselves cannot be given as inputs to computers
- BUT, numbers can be given as inputs to computers
- Encoding = converting words to vectors
    - *vector*: an ordered list of numbers (e.g., [0.1, 0.3, -0.5])

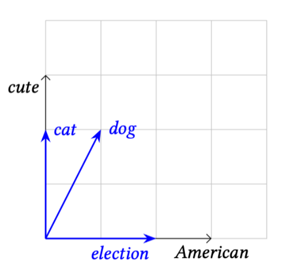
Word vectors

- If two words often occur near the same neighbors (e.g., *dog* and *cat* near *cute*), their vectors will be similar.



Word vectors

- If two words often occur near the same neighbors (e.g., *dog* and *cat* near *cute*), their vectors will be similar.
- *How*? Algorithms can automatically learn these vectors from corpus data



Word vectors

Core idea:

- Start with a large corpus

Word vectors

Core idea:

- Start with a large corpus
- Every word in a fixed vocabulary is represented by a vector

Word vectors

Core idea:

- Start with a large corpus
- Every word in a fixed vocabulary is represented by a vector
- Go through each position t in the text, which has a center word and a context word

Word vectors

Core idea:

- Start with a large corpus
- Every word in a fixed vocabulary is represented by a vector
- Go through each position t in the text, which has a center word and a context word
- Calculate the probability of a center word given a context word (or vice versa)

Word vectors

Core idea:

- Start with a large corpus
- Every word in a fixed vocabulary is represented by a vector
- Go through each position t in the text, which has a center word and a context word
- Calculate the probability of a center word given a context word (or vice versa)
- Keep adjusting the word vectors to **maximize** the probability

Word vectors

Core idea:

- Start with a large corpus
- Every word in a fixed vocabulary is represented by a vector
- Go through each position t in the text, which has a center word and a context word
- Calculate the probability of a center word given a context word (or vice versa)
- Keep adjusting the word vectors to **maximize** the probability
- (*more on this in the NLP class*!)

Vector arithmetic

- Once the vectors are learned, word vectors can be used for **mathematical operations**.

Vector arithmetic

- Once the vectors are learned, word vectors can be used for **mathematical operations**.
- *Word2Vec* (Mikolov et al. 2013):

$$\text{king} - \text{man} + \text{woman} \approx \text{queen}$$

Vector arithmetic

- Once the vectors are learned, word vectors can be used for **mathematical operations**.
- *Word2Vec* (Mikolov et al. 2013):

$$\text{king} - \text{man} + \text{woman} \approx \text{queen}$$

- We'll also explore *Word2Vec*.

Submission/Grading Guidelines

- **Text/Docx file** (either *`.txt`* or *`.docx`*): Submit your output from the corpus exploration on Tuesday. (If you missed class, complete it individually and submit.) (10 points)
- **Notebook file** (*`.ipynb`*): Submit your work from today's session.
  - Topic Modeling (5 points)
    - + 5 extra points, if you experiment this code on another corpus
  - Word2Vec (5 points)
- **Optional:** Please complete the *Collocation* Tutorial for extra credit (+3 points above max)
  - Guidelines/Code are on the last week's section (course website)
- PLEASE run all the codes, so the grader can seamlessly check your outputs!