CS 333: Natural Language Processing

Fall 2025

Prof. Carolyn Anderson
Wellesley College

Reminders

- Quiz 2 on Tuesday: J&M Chapter 3
- HW 2 will be released today
- My help hours: Monday 4-5:30

Working with Text Collections

Working With Text Collections

A **corpus** is a collection of texts. The NLTK Python library comes with many **corpora**:

Corpus	Contents
Brown Corpus	15 genres, 1.15M words, tagged, categorized
CMU Pronouncing Dictionary	127k entries
Gutenberg (selections)	18 texts, 2M words
Inaugural Address Corpus	US Presidential Inaugural Addresses (1789-present)
Indian POS-Tagged Corpus	60k words, tagged (Bangla, Hindi, Marathi, Telugu)
Movie Reviews	2k movie reviews with sentiment classification
Reuters Corpus	1.3M words, 10k news documents, categorized
Roget's Thesaurus	200k words, formatted text
Shakespeare texts (selections)	8 books in XML format
State of the Union Corpus	485k words, formatted text
Switchboard Corpus (selections)	36 phonecalls, transcribed, parsed
Univ Decl of Human Rights	480k words, 300+ languages

Working With Text Collections

NLTK provides easy methods for accessing a corpus in different formats:

```
>>> from nltk.corpus import brown
>>> brown.categories()
['adventure', 'belles_lettres', 'editorial', 'fiction',
'government', 'hobbies', 'humor', 'learned', 'lore', 'mystery',
'news', 'religion', 'reviews', 'romance', 'science_fiction']

>>> brown.words(categories='news')
['The', 'Fulton', 'County', 'Grand', 'Jury', 'said', …]

>>> brown.words(fileids=['cg22'])
['Does', 'our', 'society', 'have', 'a', 'runaway', ',', …]

>>> brown.sents(categories=['news', 'editorial', 'reviews'])
[['The', 'Fulton', 'County'...], ['The', 'jury',
'further' ...], ...]
```

Corpora

Corpora vary along many dimensions:

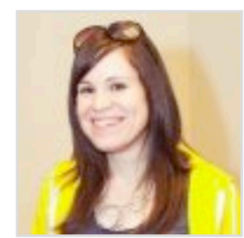
- **Language**: 7000+ languages in the world
- **Variety**, like African American Language varieties.
  AAE Twitter posts might include forms like "iont" (I don't)
- **Code switching**, e.g., Spanish/English, Hindi/English:
  S/E: Por primera vez veo a @username actually being hateful! It was beautiful:)
  [For the first time I get to see @username actually being hateful! it was beautiful:)]
- **Genre**: newswire, fiction, scientific articles, Wikipedia
- **Author Demographics**: writer's age, gender, ethnicity, SES

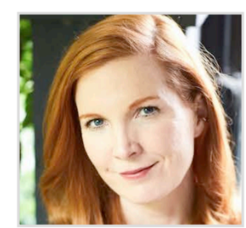
Corpus Datasheets

- Motivation:
  - *Why was the corpus collected?*
  - *By whom?*
  - *Who funded it?*
- Situation: In what situation was the text written?
- Collection process: Was there consent? Pre-processing?
- Annotation process, language variety, demographics, etc.









Datasheets for Datasets

TIMNIT GEBRU, Black in AI
JAMIE MORGENSTERN, University of Washington
BRIANA VECCHIONE, Cornell University
JENNIFER WORTMAN VAUGHAN, Microsoft Research
HANNA WALLACH, Microsoft Research
HAL DAUMÉ III, Microsoft Research; University of Maryland
KATE CRAWFORD, Microsoft Research

1 Introduction

Data plays a critical role in machine learning. Every machine learning model is trained and evaluated using data, quite often in the form of static datasets. The

Text Distributions

Text Distributions and Frequency

Last class:

Zipf's hypothesis:
Shorter words are more frequent because languages maximize efficiency: they assign common meanings to words that take less effort to produce.

This class:

Zipf's law: The frequency of a word is inversely proportional to its frequency ranking.

Zipf’s Law

The frequency of a word is proportional to the inverse of its rank.

$$f(r) \propto \frac{1}{r^a}$$

where r is the frequency rank

Zipf's Law Demo

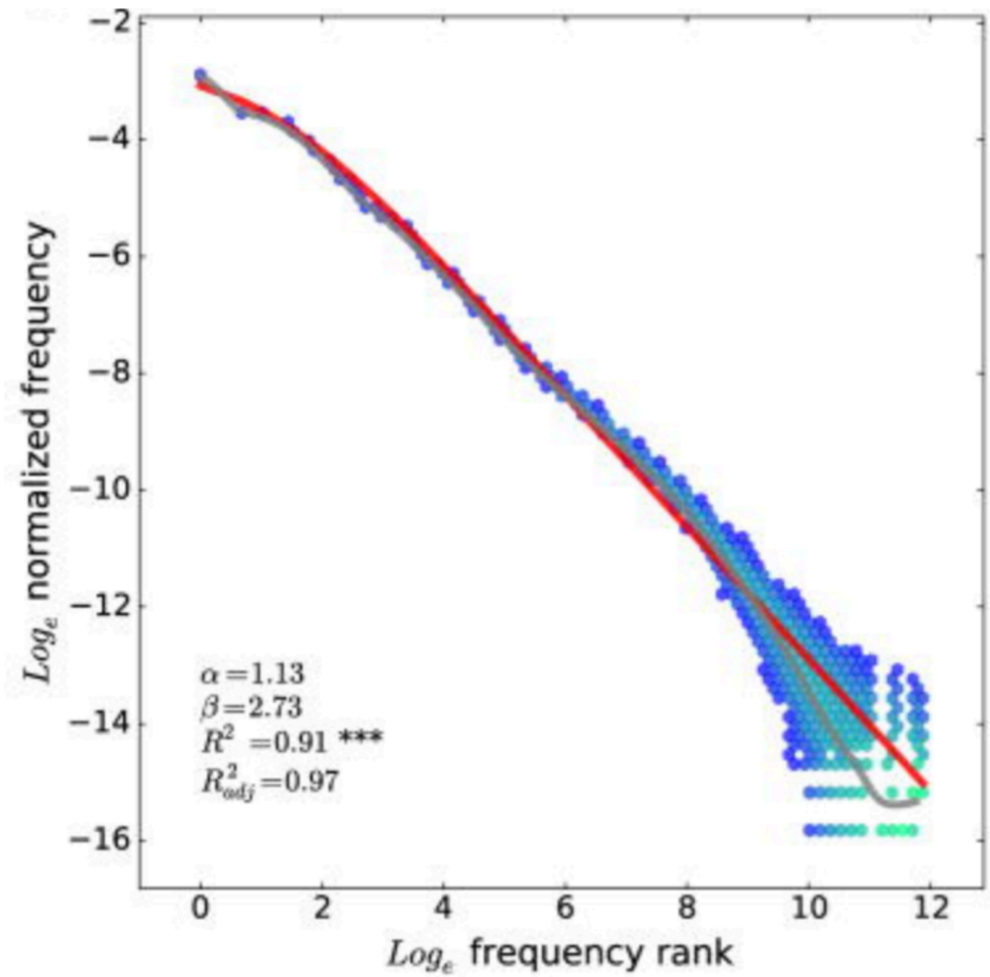
See associated code.

Text Distributions and Frequency

Zipf's Law

Here, frequency and frequency rank are calculated on two separate halves of the American National Corpus.

Piantadosi (2014)



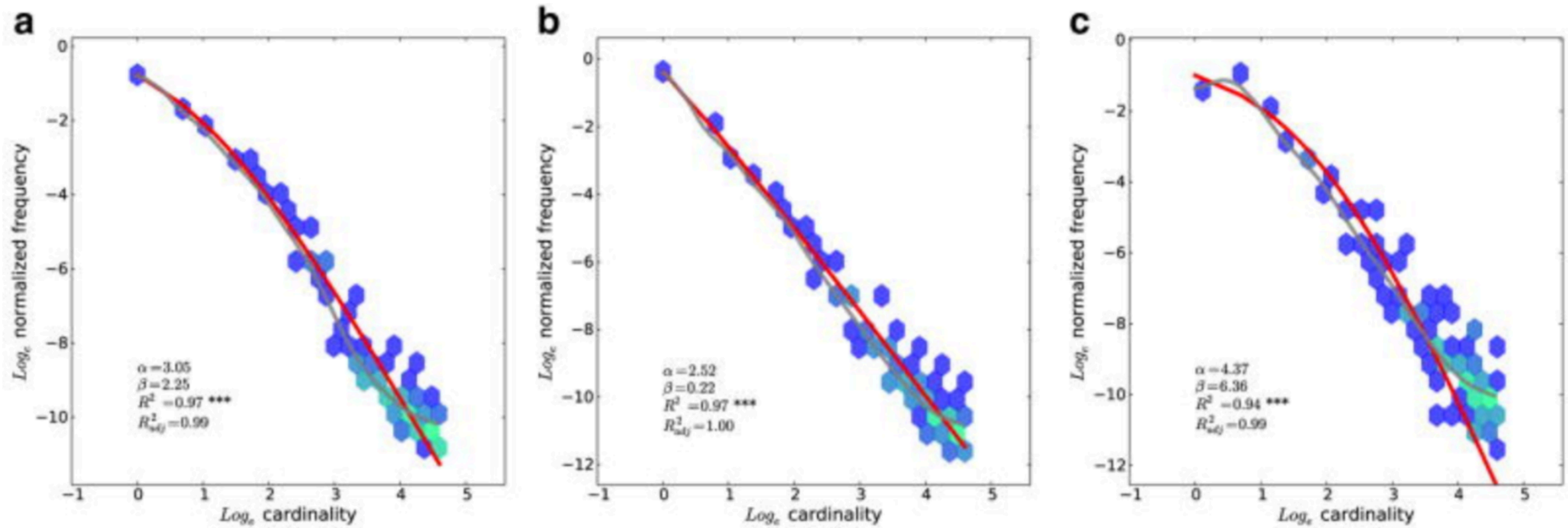
Text Distributions and Frequency

Proposed explanations for Zipf's Law

- Semantics: there are cross-linguistically stable relationships between frequency and meaning.

Text Distributions and Frequency

Counter: Zipf's Law holds within domains



Power law frequencies for number words (“one,” “two,” “three,” etc.) in English (a), Russian (b), and Italian (c), taken from Piantadosi (2014)

Plus: Zipf's Law holds in artificial languages

Text Distributions and Frequency

Proposed explanations for Zipf's Law

- Memory: since Zipf's Law holds in artificial language learning experiments, maybe it is due to constraints on human memory

Text Distributions and Frequency

Zipf's Law holds for other human systems

- Distribution of instructions in computer architecture
- Token sequences in programming languages
- Music

Encoding Text

Talking About Text

How do we start to talk about language?

In our first class, I talked about language as linguists think about it: through the lens of levels of linguistic abstraction.

When we handle raw text, though, those layers aren't always easy to pull apart.

How can we describe text data?

Encoding Text

What do we find in a **collection of texts**?

At the most basic level, a corpus is a collection of text data, stored in some **encoding**.

Encoding Text

Python 3 strings are stored internally as **Unicode**

- each string is a sequence of Unicode code points (characters)
- string functions, regex apply natively to code points.
  - len() returns string length in code points, not bytes

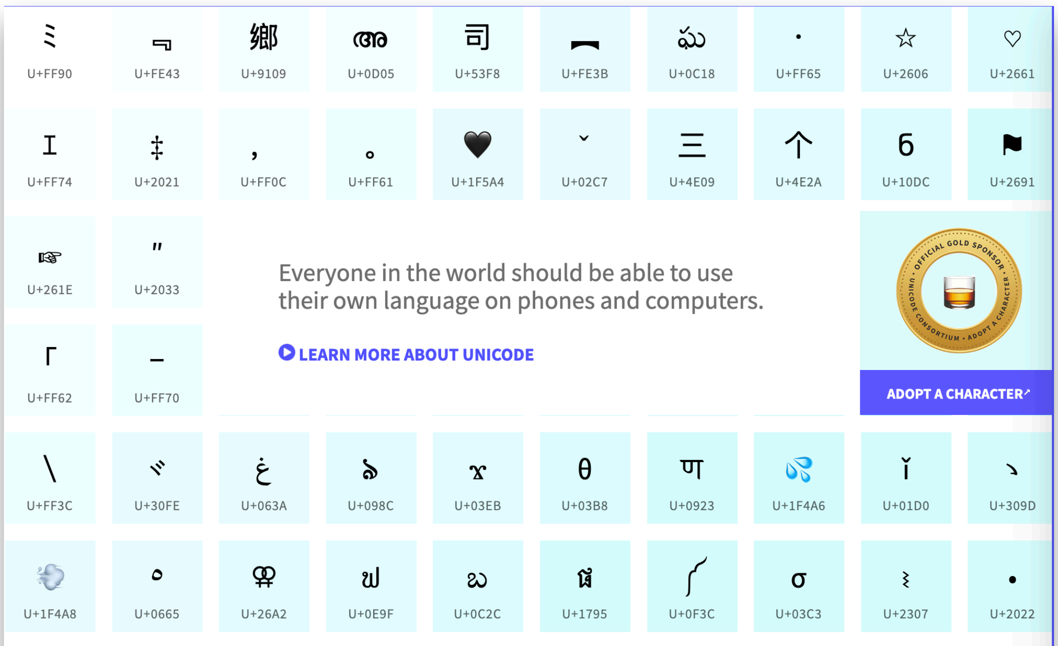
Unicode

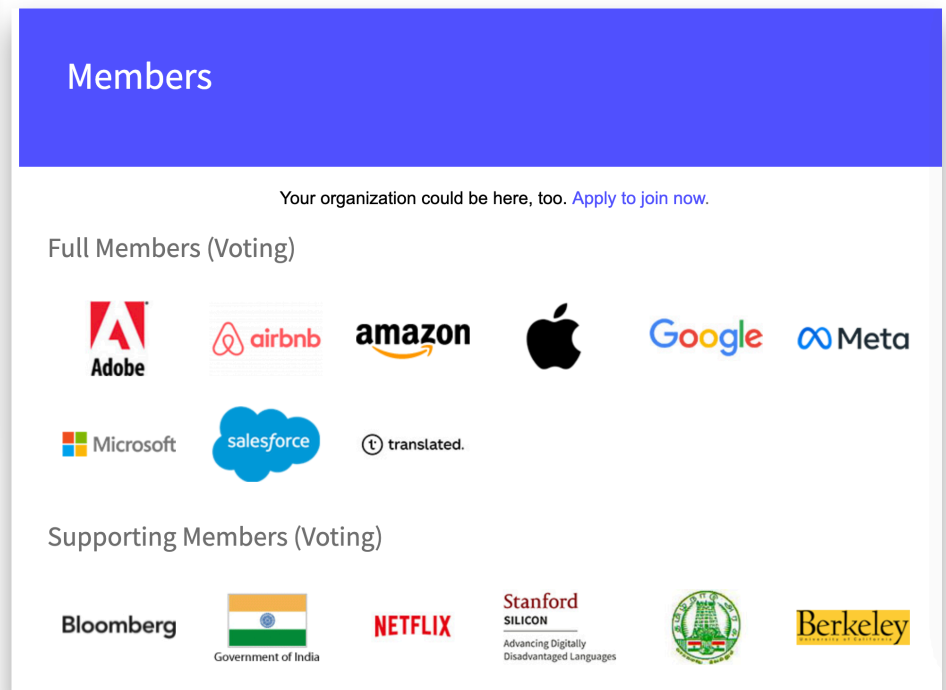
Unicode is a method for representing text using:

- any character (more than 150,000!)
- in almost all scripts (168 to date!)
- of the languages of the world:
  - Chinese, Arabic, Hindi, Cherokee, Ethiopic, Khmer, N'Ko,…
  - dead ones like Sumerian cuneiform
  - invented ones like Klingon
- plus emojis, currency symbols, etc.

Unicode

Unicode is a standard encoding format determined by the Unicode Consortium:





Encoding Text

Files need to be **encoded/decoded** when written or read

- Every file is stored in some encoding
- No such thing as a text file without an encoding
  - If it's not UTF-8 (Unicode), then it's something older like ASCII or iso_8859_1

Encoding Text

That's how programming languages encode text data.

However, humans often talk instead about **words**.

Talking About Text

Talking About Words

"Seuss's cat in the hat is different from other cats!"

Lemma: same stem / rough meaning

cat, cats

Word form: full inflected surface form

cat $\neq$ cats

Talking About Words

"Seuss's cat in the hat is different from other cats!"

- **Lemma**: same stem, part of speech, rough word sense

- **Wordform**: the full inflected surface form

Talking About Words

they lay back on the San Francisco grass and looked at the stars

- **Type**: an element of the vocabulary

12

- **Token**: an instance of that type in running text

13

Heap's Law

Vocabulary size grows at a rate equal to or greater than the square root of the number of word tokens

	Tokens = N	Types = |V|
Switchboard phone conversations	2.4 million	20 thousand
Shakespeare	884,000	31 thousand
COCA	440 million	2 million
Google N-grams	1 trillion	13+ million

Text Processing

Tokenization

Tokenizers map text bits to numeric token ids.

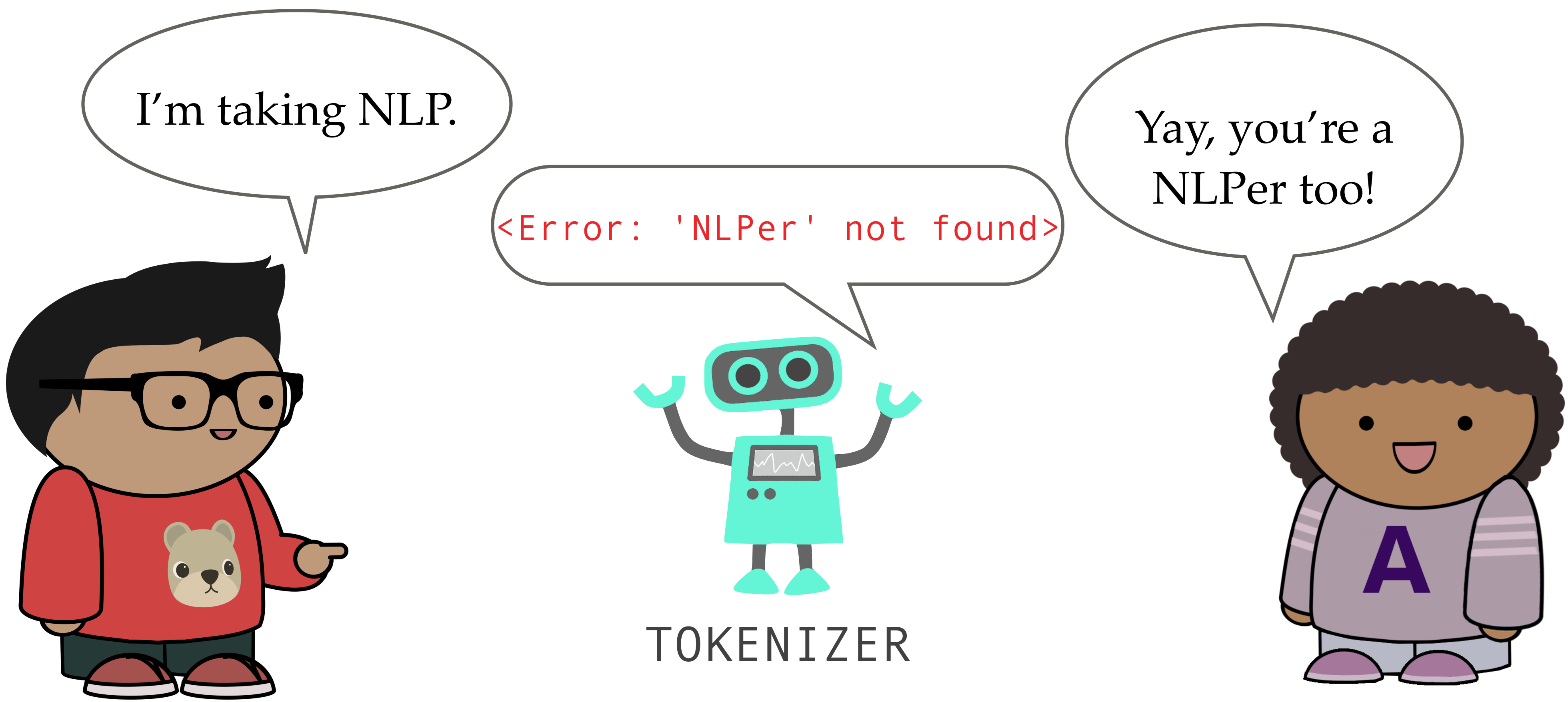
But wait: why can't we just use words?

Tokenization

Heap's Law is one reason. There are too many words in large corpora for words to be an efficient information representation.

Tokenization

Also, if we use words, we’ll run into problems if we later see a new word (which is always possible, because language is productive!).



Tokenization

Finally, finding word boundaries is not trivial for many languages.

For instance, Chinese is written using characters, each of which represents a **morpheme: a minimal meaning-bearing unit in a language.**

Tokenization

Dividing strings of characters into words is complex, and there are different systems:

姚明进入总决赛 “Yao Ming reaches the finals”
◦yáo míng jìn rù zǒng jué sài

3 words?							Chinese Treebank
姚明	进入	总决赛					
YaoMing	reaches	finals					
5 words?							Peking University
姚	明	进入	总	决赛			
Yao	Ming	reaches	overall	finals			
7 words?							Use characters.
姚	明	进	入	总	决	赛	
Yao	Ming	enter	enter	overall	decision	game	

Tokenization

Options:

- White-space / orthographic words
  - Lots of languages don't have them
  - The number of words grows without bound
- Unicode characters
  - Too small as tokens for many purposes
- Morphemes
  - Hard to define/find in many languages

Tokenization

Options:

- White-space / orthographic words
  - Lots of languages don't have them
  - The number of words grows without bound
- Unicode characters
  - Too small as tokens for many purposes
- Morphemes
  - Hard to define/find in many languages

Alternative: We use the data to tell us how to tokenize.

Byte Pair Encoding

Byte Pair Encoding

Byte Pair Encoding (BPE) (Sennrich et al., 2016) is an algorithm for finding an efficient, fixed-length vocabulary of tokens.

We will **learn** a mapping from a training corpus.

Byte Pair Encoding Learner

Repeat:

- Choose most frequent neighboring pair ('A', 'B')
- Add a new merged symbol ('AB') to the vocabulary
- Replace every 'A' 'B' in the corpus with 'AB'.

Until *k* merges

Vocabulary

[A, B, C, D, E]

[A, B, C, D, E, AB]

[A, B, C, D, E, AB, CAB]

Corpus

A B D C A B E C A B

AB D C AB E C AB

AB D CAB E CAB

Byte Pair Encoding

Most subword algorithms are run inside space-separated tokens.

So we commonly first add a special end-of-word symbol '__' before space in training corpus

Next, separate into letters.

Byte Pair Encoding

Original corpus:

that sweet i guess so that’s that me espresso

Vocabulary (includes space tokens)

__, i, s, t, h, a, w, e, g, u, o, ', m, p, r

Byte Pair Encoding

Vocabulary

__, i, s, t, h, a, w, e, g, u, o, ', m, p, r

Corpus

3: t h a t _
1: s w e e t _
1: i _
1: g u e s s _
1: s o _
1: ’ s _
1: m e _
1: e s p r e s s o _

Merge

t _ → t_

Byte Pair Encoding

Vocabulary

__, i, s, t, h, a, w, e, g, u, o, ', m, p, r, t_

Corpus

3: t h a t_
1: s w e e t_
1: i _
1: g u e s s _
1: s o _
1: ’ s _
1: m e _
1: e s p r e s s o _

Merge:

t h to th

Byte Pair Encoding

Vocabulary

__, i, s, t, h, a, w, e, g, u, o, ', m, p, r, t_, th

Corpus

3: th a t_
1: s w e e t_
1: i _
1: g u e s s _
1: s o _
1: ’ s _
1: m e _
1: e s p r e s s o _

Merge

e s → es

Byte Pair Encoding

Vocabulary

__, i, s, t, h, a, w, e, g, u, o, ', m, p, r, t_, th, es

Corpus

3: th a t_
1: s w e e t_
1: i _
1: g u es s _
1: s o _
1: ’ s _
1: m e _
1: es p r es s o _

Next Merges:

(th, a) → (tha)

(tha, t_) → (that_)

Byte Pair Encoding

Vocabulary

__, i, s, t, h, a, w, e, g, u, o, ', m, p, r, t_, th, es

Corpus

3: th a t_
1: s w e e t_
1: i _
1: g u es s _
1: s o _
1: ’ s _
1: m e _
1: es p r es s o _

Next merges:

Byte Pair Encoding

Once we're done learning, we run each merge learned from the training data:

- Greedily
- In the order we learned them
- (test frequencies don't play a role)

So: merge every t h to th, then merge th a to tha, etc.

Result:

"r e s t _" —>   r es t_

BPE In the Wild

BPE is one of the most widely used tokenizer learning algorithms for LLMs.

In general, we take Unicode text and encode it using UTF-8 into bytes. We run the algorithm over the **bytes**, rather than characters.

BPE In the Wild

Tiktokenizer

meta-llama/Meta-Llama-3-8B

I would like some cake.

Token count
6

I·would·like·some·cake.

40, 1053, 1093, 1063, 19692, 13

https://tiktokenizer.vercel.app/?model=meta-llama%2FMeta-Llama-3-8B

Text Normalization

Other Normalization Tasks

Working with corpora usually requires text normalization:

1. Tokenizing (segmenting) words
2. Normalizing word formats
3. Segmenting sentences

Sentence Segmentation

!, ? mostly unambiguous but **period** "." is very ambiguous

- Sentence boundary
- Abbreviations like Inc. or Dr.
- Numbers like .02% or 4.3

Common algorithm:

- Tokenize first
- Use rules or ML to classify a period as either (a) part of the word or (b) a sentence-boundary.
- An abbreviation dictionary can help

Sentence segmentation can then often be done by rules based on this tokenization.

Normalizing Words

- Depending on the application, we might want to:
  - Standardize spelling
  - Use a consistent case (all lowercase)
  - Separate some morphemes:
    didn't —> did n't

Or we might not want to! It depends on the task.

Homework 2

- Part 1: tokenization
  - Goal: explore different choices to be made in tokenizing English text
- Part 2: sentence segmentation
  - Goal: write a robust English sentence segmenter