NLP Lecture Notes #1.2: Corpora, Text, and Tokenization

Mehmet Can Yavuz, PhD

September 22, 2025

In Natural Language Processing (NLP), we build computational models that learn from and process human language. But where does this language data come from? We rely on **corpora**.

Definition

A **corpus** (plural: **corpora**) is a collection of natural language data, such as text or speech, assembled for the purpose of linguistic analysis. Essentially, it's a dataset for language.

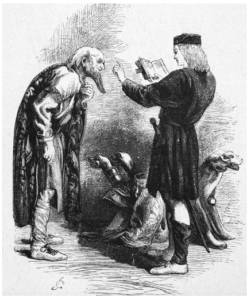
Corpora can be **raw**, meaning they have undergone minimal processing, or **annotated**, where they are enriched with linguistic information like Part-of-Speech (POS) tags, sentiment labels, or syntax trees.
We can think of text data as being organized in a hierarchy, from large collections down to the smallest units of meaning.

Definition

- **Corpora**: A collection of multiple corpora or datasets. (e.g., All plays by Shakespeare).
- **Corpus**: A single, defined dataset. (e.g., The text of *Hamlet*).
- **Document**: A single piece of text within a corpus. (e.g., Act 2, Scene 2 of *Hamlet*).
- **Token**: The basic unit of text, often a word or punctuation mark. (e.g., "Words").

When working with a corpus, it's crucial to understand its composition. Key properties include:

- **Language and Variety**: Is it English, French, or Turkish? Is it a specific dialect like British English or African American English (AAE)?.
- **Genre**: Is the text from newswire, social media, fiction, or scientific articles?.
- **Time Period**: When was the text produced? Language changes over time.



Polonius: What do you read, my lord?
Hamlet: Words, words, words.
Act 2, scene 2, Hamlet

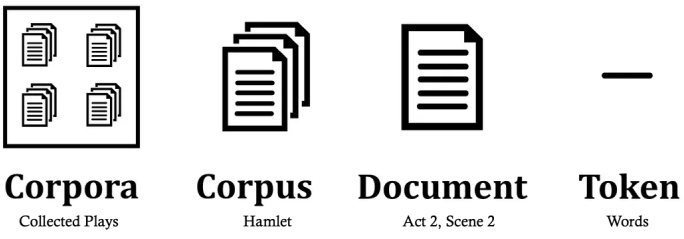


Figure 1: Understanding Text Levels: From Corpora (Collected Plays) to Tokens (Words).

- **Demographics**: Information about the speakers or authors, such as age, gender, and region, can be vital for building fair and unbiased models.

To document these properties, corpus creators often provide a **datasheet** or **data statement**, which details the motivation, collection process, and demographic makeup of the data.
When we count "words" in a corpus, we must distinguish between two concepts: types and tokens.

Definition

- A **word token** is an individual instance or occurrence of a word in the text. It's the total running count of words.
- A **word type** is a unique word form that appears in the text. The set of all types is the **vocabulary** of the corpus.

Example

Consider the famous line from *Hamlet*: **"To be or not to be"**.

- **Tokens**: There are 6 tokens in this sentence: 'To', 'be', 'or', 'not', 'to', 'be'.
- **Types**: There are 4 unique types: 'To', 'be', 'or', 'not'. The full vocabulary is {To, be, or, not}. (Note: We might choose to treat "To" and "to" as the same type through a process called normalization, discussed later).

In any natural language corpus, we observe a consistent statistical pattern known as **Zipf's Law**. It states that the frequency of any word is inversely proportional to its rank in the frequency table. This means that a small number of words are extremely frequent, while a vast number of words are extremely rare.
This pattern is observed across various languages and corpora. A few words dominate the frequency counts, while most words form a long tail of rare occurrences...

- **Frequent Words**: A few words (like "the", "a", "is") make up a huge portion of the tokens in a corpus.
- **Rare Words (The Long Tail)**: Most words in the vocabulary appear very infrequently. Even after reading millions of words, you will continuously encounter new, unseen words.

This has important implications for NLP:

- **The Good**: Frequent words give us reliable statistics and help us model the structure of a text.
- **The Bad**: Any text will contain many rare words for which we have sparse data, making them hard to model reliably.
- **The Ugly**: Any text is guaranteed to contain words that are completely unknown (Out-of-Vocabulary or OOV), which our systems must be able to handle gracefully.

To effectively handle unseen events, NLP systems must generalize from observed data. Two complementary strategies facilitate this generalization: (1) Linguistic Insights (2) Machine Learning and Statistics.

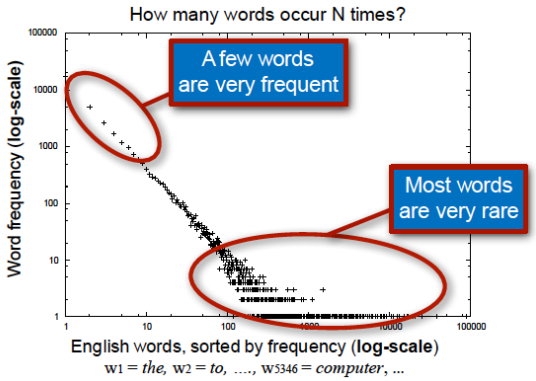


Figure 2: The chart illustrates the distribution of English word frequencies on a log-scale.

Tokenization is the fundamental first step in most NLP pipelines. It is the process of breaking a stream of text into its constituent units, called tokens. These tokens are often words and punctuation marks.
Simply splitting text by whitespace is insufficient for most languages, including English.

- **Punctuation**: Should "earth." be one token or two ('earth' and '.')? Usually, we want to split punctuation from words. For example, 'they're' should be split into 'they' and ''re' (or 'are').
- **Abbreviations**: A period in "Mr. Smith" or "U.S.A." does not mark the end of a sentence and should be handled carefully.
- **Hyphenation**: Words like "state-of-the-art" might be treated as a single token.
- **Special Constructs**: URLs (`http://google.com`), hashtags (#NLP), and emoticons (:-) are meaningful units that simple splitting would destroy.
- **Languages without Spaces**: Many languages, such as Chinese, Japanese, and Thai, do not use spaces to separate words, making tokenization (often called word segmentation) a much more complex task.

For many languages, tokenization is implemented using carefully crafted rules. The most powerful tool for specifying these rules is the **regular expression** (or regex), a formal language for specifying text patterns.

Pattern	Description	Example
`/cat/`	Matches the literal string "cat".	"The **cat** sat."
`/[cC]at/`	Matches 'c' or 'C' (disjunction).	"The **cat**" or "**C**at"
`/beg.n/`	'.' matches any single character.	"**begin**", "**began**", "**begun**"
`/colou?r/`	'¿ makes the preceding 'u' optional (0 or 1).	"**color**" or "**colour**"
`/ba+/`	'+' matches one or more 'a's.	"**ba**", "**baa**", "**baaa**"
`/ba*/`	'*' matches zero or more 'a's.	"**b**", "**ba**", "**baa**"
`/\bthe\b/`	'\b' matches a word boundary.	"**the**" but not "other"
`/^The/`	'^' matches the start of a line.	"**The** dog..." at line start.
`/end$/`	'$' matches the end of a line.	"...the very **end**"
`/[0-9]+/` or `/\d+/`	Matches one or more digits.	"User **123**"

Parentheses '()' are used to group parts of a regex. This is crucial for controlling the scope of operators like '—' (OR), '*', and '+'.

Example

Suppose we want to match "guppy" or "guppies".

- `/guppy|ies/` is **incorrect**. It matches "guppy" OR "ies" because sequence has higher precedence than disjunction '—'.
- `/gupp(y|ies)/` is **correct**. It matches "gupp" followed by either "y" or "ies".

After tokenizing, we often want to normalize words to a standard format. This helps reduce the vocabulary size and treat different forms of the same word as equivalent.

- **Case Folding**: Converting all text to a single case, usually lowercase. For example, 'The' and 'the' become the same token. This is a simple but effective normalization step.
- **Stemming**: A crude, heuristic process of chopping off the ends of words to get a common "stem". The result is not guaranteed to be a real word.
  - The **Porter Stemmer** is a classic example. It uses a series of rules like 'ATIONAL' → 'ATE' (e.g., 'relational' → 'relate') and 'ING' → ' ' (e.g., 'motoring' → 'motor').
  - **Example**: 'organizes', 'organized', 'organizing' might all become 'organ'.

- **Lemmatization**: A more principled process that uses morphological analysis to find the dictionary headword, or **lemma**, of a word.
  - **Example**: ‘organizes‘, ‘organized‘, ‘organizing‘ would all become ‘organize‘. The words ‘am‘, ‘are‘, and ‘is‘ all share the lemma ‘be‘.

Lemmatization is more accurate than stemming but is more computationally expensive as it often requires knowledge of a word's part-of-speech.

Example

Regex for Phone Numbers Write a regular expression that can match North American phone numbers in the following formats: ‘(123) 456-7890‘, ‘123-456-7890‘, ‘123.456.7890‘, and ‘123 456 7890‘. The area code in parentheses should be optional.

A regular expression to capture all formats is:
`/\(?\d{3}\)?[-. ]?\d{3}[-. ]?\d{4}/`
Explanation:

- `\(?`: Matches an optional opening parenthesis. ‘(‘ is a special character, so it's escaped with ‘\. The ‘? makes it optional.
- `\d{3}`: Matches exactly three digits (the area code). ‘\d is shorthand for ‘[0-9]‘.
- `\)?`: Matches the optional closing parenthesis.
- `[-. ]?`: Matches an optional separator, which can be a hyphen, a period, or a space. ‘[]‘ creates a character class.
- `\d{3}`: Matches the next three digits.
- `[-. ]?`: Matches the optional second separator.
- `\d{4}`: Matches the final four digits.

Example

Types and Tokens Read the following sentence:
"The quick brown fox jumps over the lazy dog, and the lazy dog sleeps."

1. How many word tokens are in this sentence (ignoring punctuation)?
2. How many word types are in this sentence if you perform case-folding (i.e., treat "The" and "the" as the same type)?

The sentence is: *"The quick brown fox jumps over the lazy dog, and the lazy dog sleeps."*

1. **Tokens**: Counting the words, we get **15** tokens. (‘The‘, ‘quick‘, ‘brown‘, ‘fox‘, ‘jumps‘, ‘over‘, ‘the‘, ‘lazy‘, ‘dog‘, ‘and‘, ‘the‘, ‘lazy‘, ‘dog‘, ‘sleeps‘)
2. **Types**: After case-folding "The" to "the", the set of unique words (types) is:
   {‘the‘, ‘quick‘, ‘brown‘, ‘fox‘, ‘jumps‘, ‘over‘, ‘lazy‘, ‘dog‘, ‘and‘, ‘sleeps‘}
   There are **10** types.

Example

Stemming vs. Lemmatization What would be the likely output if you applied a Porter stemmer and a lemmatizer to the following words?

- policies
- studying
- geese
- ponies

Word	**Porter Stemmer (Likely)**	**Lemmatizer (Correct)**
policies	polic (over-stemming error)	policy
studying	studi	study
geese	gees (error, irregular plural)	goose
ponies	poni	pony

Explanation: This example highlights the difference between the two methods. The Porter stemmer is a blunt instrument that just applies suffix-stripping rules, which can lead to non-words (‘polic‘, ‘studi‘, ‘poni‘) and fails on irregular plurals like ‘geese‘. A lemmatizer, using dictionary lookups and morphological analysis, correctly returns the base dictionary form for each word.