DSE6212

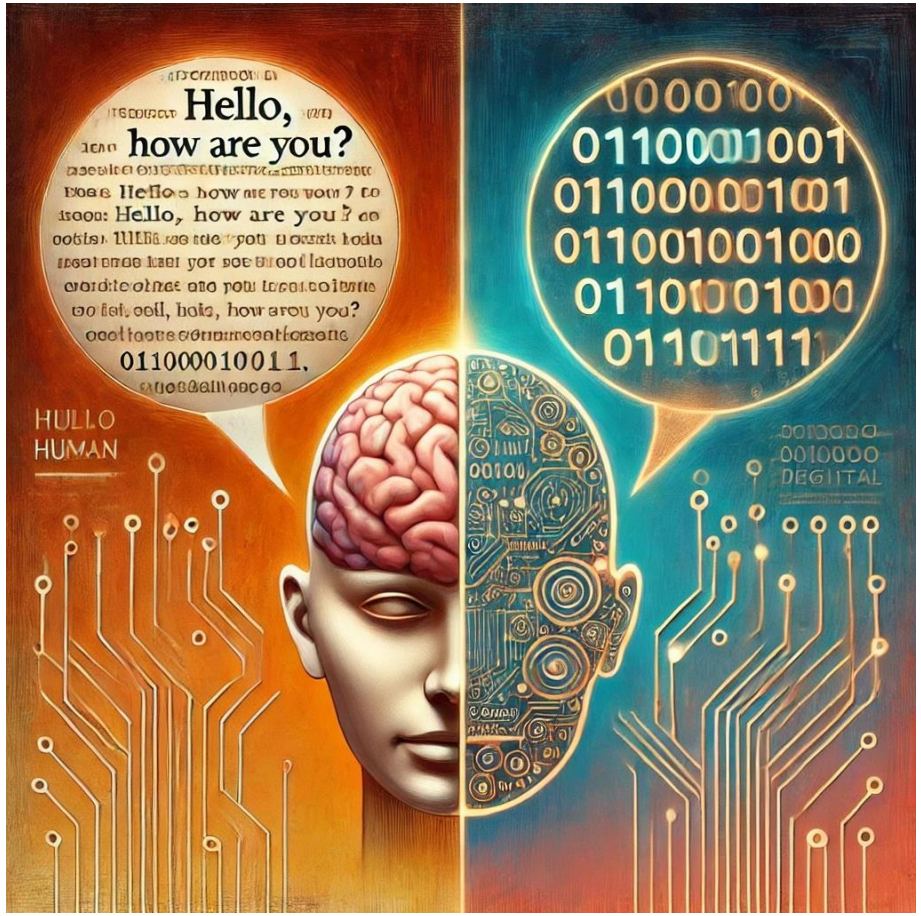
Text and Image Mining
Session # 1

Agarwal

MERRIMACK COLLEGE

Introduction to Natural Language Processing (NLP)

NLP is a field of Artificial intelligence focused on enabling machines to understand, interpret, and generate human language in a way that's both meaningful and useful..



Applications:

- Sentiment Analysis (e.g., Analyzing tweets or reviews)
  - "I absolutely loved the move! The ending was brilliant" (Positive Sentiment)
- Machine translation (e.g., Google Translate)
  - "Como estas hoy?" ("How are you today?")
- Chatbots and Virtual Assistants (e.g., Siri, Alexa)
  - "What's the weather tomorrow" ("It will be sunny with a high of 250 F")
- Information Retrieval (e.g., Search Engine)
- Text Summarization (e.g., TL;DR tools)

Why is NLP difficult

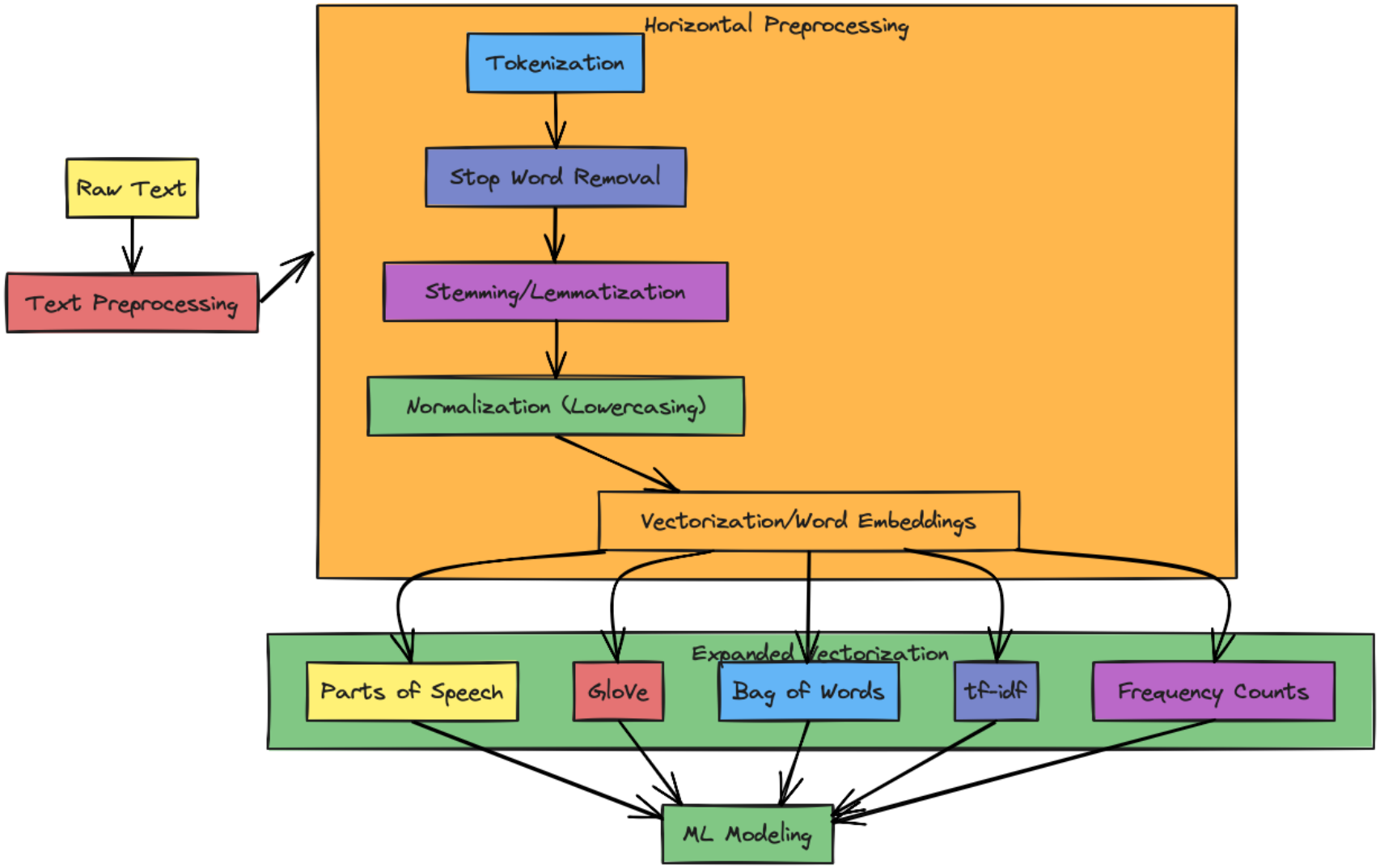
- Computers operate on strict rules, binary logic, and precise instructions. Their “thinking” is structured, predictable, and methodical—making them ideal for tasks requiring clarity and unambiguous communication.
- Human language, in contrast, is rich, nuanced, and often riddled with contradictions, idioms, emotions, and context. Its rules can be bent, broken, or interpreted differently based on culture, tone, or even body language.
  - Ambiguity in the language: where a word or phrase has multiple interpretations
    - E.g., “The man saw the women with the bike”
    - Possible interpretations: The man used a bike to see the woman OR The woman had a bike, and the man saw her
  - Contextual Understanding: It’s the ability to interpret words or phrases based on their surroundings text i.e. the meaning becomes clear when we consider broader context
    - E.g.: “The bank is on the left”
    - Possible interpretations: A financial institution is located on the left side OR the riverbank is on the left side
  - Idioms, Metaphors and Slang: Informal language expressions are not literal, making then hard for machines to understand
    - E.g.
      - Metaphor: “It’s raining cats and dogs”
      - Slang: “That’s fire!”
      - Idiom: “Break the ice” (break frozen water or start a conversation)

Why is NLP difficult

- Diverse Syntax and Grammar across Languages: Different languages follow different rules for word order, punctuation, and grammar
  - E.g.,
    - English: I am going to the store.”
    - Japanese: “Store to I go.”
- Cultural and Regional Nuances: where language uses, idioms and meaning vary between regions and cultures
  - E.g., ‘Rubber”
    - In American English, refers to Tire and in British English to an Eraser
- Polymorphic Nature of words (Morphology): Words change forms depending on tense, number and grammatical usage
  - E.g., “run” (base form), “ran” (past tense), “ running” (gerund), “runs” (third-person singular present)
- High Volume and variability of Data: Human communication involves massive amounts of data to communicate message
  - E.g., A single tweet has different linguistic characteristics compared to a research paper
- Noise and Error in Data: Human –generated text is often filled with typos, emojis, abbreviations and irrelevant information
  - Noise: “Amazing deal!!! Get 50% off!!! 🌯🌯” (Irrelevant symbols and excessive punctuation).
  - Errors: “I reely lik NLP becuz its amzing.” (Misspelled words and grammar errors).

These challenges highlight the need for preprocessing to clean, standardize, and interpret text before it’s fed into NLP models. Let’s dive into the workflow where preprocessing plays a key role

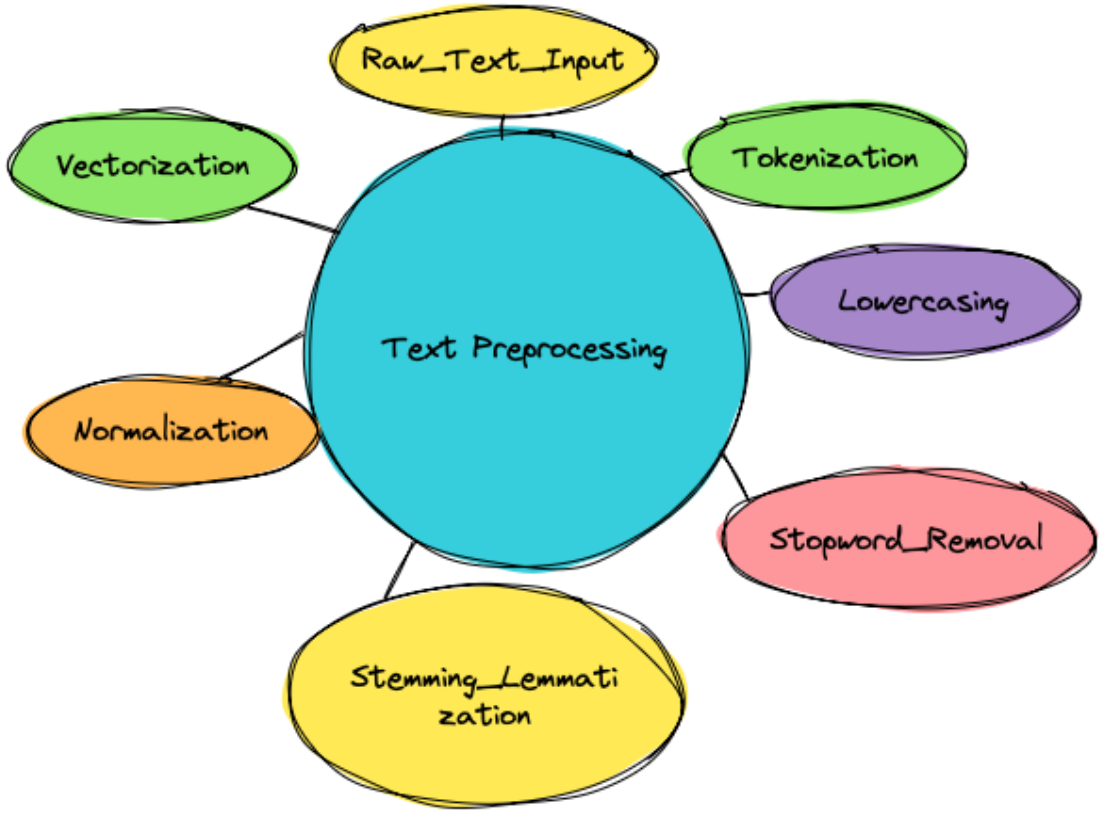
High level Processes in NLP



1. Raw Text
    - Starting point: unstructured text (e.g., tweets, articles, emails).
2. Text Preprocessing
    - Tokenization: Splitting text into words or sentences.
    - Stop-Word Removal: Removing common, non-informative words.
    - Stemming/Lemmatization: Reducing words to their root forms.
    - Normalization: Ensuring text consistency (e.g., lowercasing).
3. Vectorization: Converting preprocessed text into numbers
    - Bag of Words: Word frequency.
    - TF-IDF: Highlighting unique words.
    - GloVe/Word2Vec: Advanced embeddings capturing semantic relationships between words
4. ML Modeling
    - Processed data feeds into models for NLP tasks:
    - Sentiment analysis
    - Text classification
    - Translation

Text Preprocessing (1 of 3)

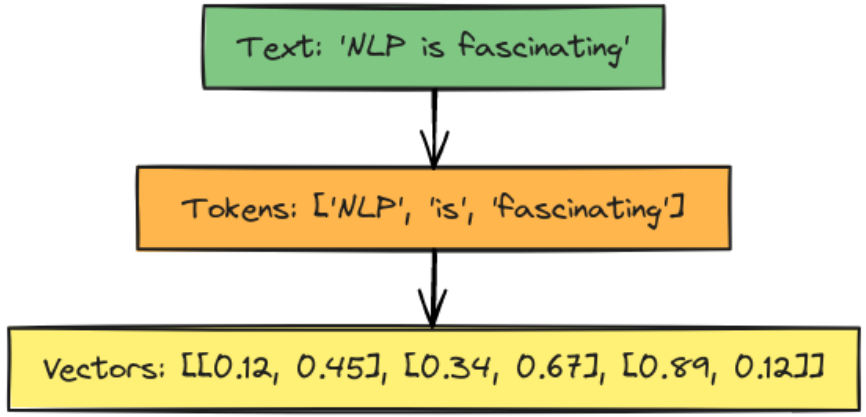
Raw text data is often noisy, inconsistent, and unstructured, containing irrelevant characters, stop-words, and variations in capitalization or spelling. Preprocessing ensures that text data is standardized, cleansed and its dimensionality reduced, facilitating effective model training and reducing computational overhead.



- **Tokenization:** Splits the text into smaller units, such as words or sentences, called tokens.
  - Example: [“Natural Language Processing is exciting”] ➔ [“Natural”, “Language”, “Processing”, “is”, “exciting”]
- **Lowercasing:** Each word is transformed into lowercase equivalent to treating same words with different cases as separate tokens, thereby simplifying the vocabulary
- **Stop-word Removal:** Stop-words eliminates commonly used words (e.g., “is,” “the,” “and”) which, while they add syntactic support to overall context of a sentence, and offer no specific insights into the meaning of the text. Removing them simplifies the text and reduces its dimensionality, making analysis more efficient.
  - Example: [“Natural”, “Language”, “Processing”, “is”, “exciting”] ➔ [“Natural”, “Language”, “Processing”, “exciting”]
- **Stemming / Lemmatization:** Reduces words to their base or root form to standardize variations of a word.
  - Example (Stemming): “running” ➔ “run”; “jumps” ➔ “jump”
  - Example (Lemmatization): “better” ➔ “good”; “children” ➔ “child”
- **Text Normalization:** Standardizes the text by ensuring uniformity in the data such as removal of punctuation, symbols and other irrelevant elements.
  - Example: “I’ve been LOVING NLP!! It’s AMAZING 😍!!!” → “i love nlp it amazing”
  - Example: “color” vs. “colour” → “color”

Text Preprocessing (2 of 3)

- Vectorization / Embedding: Converts text into numerical representations that machine learning models can process. Text, being unstructured, must be transformed into a structured numerical format for computational analysis.



- Vectorization techniques like Bag-of-Words and TF-IDF create word frequency vectors, while Embeddings (e.g., Word2Vec, GloVe) capture semantic meanings in a dense, continuous space.
- Example:
  - Document 1: "NLP is fun and exciting."
  - Document 2: "NLP is challenging but rewarding."

  Vocabulary: [“NLP”, “is”, “fun”, “and”, “exciting”, “challenging”, “but”, “rewarding”]

For BOW:

Document	NLP	is	fun	and	exciting	challenging	but	rewarding
Doc 1	1	1	1	1	1	0	0	0
Doc 2	1	1	0	0	0	1	1	1

For tf-idf:

Document	NLP	is	fun	and	exciting	challenging	but	rewarding
Doc 1	0	0	0.5	0.5	0.5	0	0	0
Doc 2	0	0	0	0	0	0.5	0.5	0.5

Using Word2Vec or GloVe, the words “king,” “queen,” “man,” and “woman” might be represented as:

king = [0.8, 0.6, 0.1], queen = [0.8, 0.6, -0.1], man = [0.7, 0.5, 0.2], woman = [0.7, 0.5, -0.2] which allows the vectors to determine mathematical relationship such that

king - man + woman ≈ queen

Text Preprocessing (3 of 3)

- **Count based N-Grams:** Represents text as sequences of n words (e.g.: bigrams, trigrams) to capture local context and word order
- Example:
  - **Text: “NLP is exciting and fun.”**
  - Bigrams: [“NLP is”, “is exciting”, “exciting and”, “and fun”]
  - Trigrams: [“NLP is exciting”, “is exciting and”, “exciting and fun”]

- One-Hot Encoding: Represents each word as a binary vector where only one element (corresponding to the word) is 1, and all others are 0.
- Example:
  - **Vocabulary: [“NLP”, “is”, “fun”, “challenging”]**

  One-Hot Encodings:
  - NLP: [1, 0, 0, 0]
  - is: [0, 1, 0, 0]
  - fun: [0, 0, 1, 0]
  - challenging: [0, 0, 0, 1]

After these preprocessing steps, the text data is structured, simplified, and numerically encoded, ready for input into NLP models.

This preparation ensures that models receive consistent, relevant data, enabling them to learn patterns more effectively.

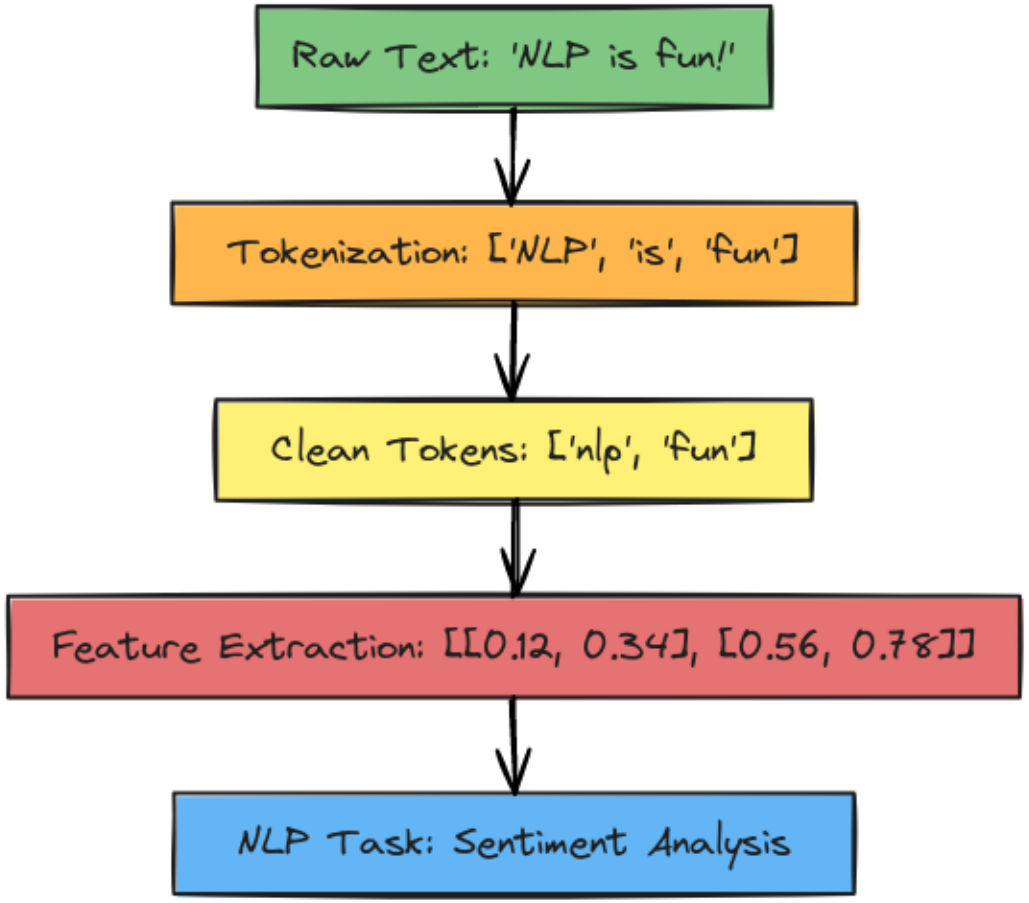
Each step in this pipeline is crucial to avoid noise, increase interpretability, and improve model performance.

Tokenization

Tokenization is one of the most foundational steps in NLP tasks - it transforms raw text into smaller, more manageable units called tokens.

Tokenization is also crucial because it transforms unstructured, complex language data into organized pieces. Without tokenization, an NLP model wouldn't know where one word ends, and another begins, making it nearly impossible to analyze or model text data.

Consider this: if we wanted to teach a computer to "read" and understand language, we wouldn't start by having it read an entire book or sentence all at once. Instead, we'd break down the text into manageable parts—words or sentences—so it can understand language step-by-step. Tokenization allows us to do exactly that.



Raw Text: "The quick brown fox jumps over the lazy dog."
Tokenized Text: ["The", "quick", "brown", "fox", "jumps", "over", "the", "lazy", "dog"]

Tokenization

Types of Tokenization:

Manual tokenization involves using custom rules or regular expressions (regex) to define how text should be split into tokens. It provides high control and flexibility, especially for domain-specific tasks or noisy data. Tokens are extracted based on patterns defined explicitly in the code.

Automated tokenization relies on pre-built libraries, such as NLTK, SpaCy (can handle multiple languages), which use optimized algorithms to split text into tokens. These tools are fast, reliable, and suited for general-purpose NLP tasks.

text = "Natural Language Processing (NLP) is fascinating! Let's dive into tokenization."

Automated Tokenization:

- **Sentence Tokenization:** Splits text into individual sentences, useful for document and paragraph-level analysis such as summarization where understanding full sentences is essential
  - **Example:**

```
from nltk.tokenize import sent_tokenize
sentences = sent_tokenize(text)
print("Sentences:", sentences)
```

Output:

Sentences: ['Natural Language Processing (NLP) is fascinating!', "Let's dive into tokenization."]

Here, the *sent_tokenize* function from NLTK breaks down the text into two sentences. This step is essential for tasks requiring an understanding of sentence structure, such as summarization or translation.

Auto Tokenization

- Special Cases: Punctuation and Contractions

Tokenization becomes complex when dealing with special cases like punctuation, contractions, or noisy text. In typical NLP preprocessing, punctuation may need to be removed, and contractions expanded to improve model understanding. For e.g.: Clean up punctuation, and handle contractions (like “Let’s” ➔ “Let us”) using custom tokenization rules or additional libraries. On other hand contractions such as ”don’t” that can split into [“don”,”t”] can distort meaning and create ambiguities

Punctuation can either be retained (e.g., sentiment analysis) or removed (e.g., for topic modeling)

- **Example**

```
from nltk.tokenize import RegexpTokenizer
# Include punctuation
tokens_with_punct = RegexpTokenizer(r'\w+|[!?.,]').tokenize(text)
# Exclude punctuation
tokens_no_punct = RegexpTokenizer(r'\w+').tokenize(text)
print("With Punctuation:", tokens_with_punct)
print("Without Punctuation:", tokens_no_punct)
```

Output:

With Punctuation: ['Natural', 'Language', 'Processing', '(', ')', 'is', 'fascinating', '!', "Let's", 'dive', 'into', 'tokenization', '.']

Without Punctuation: ['Natural', 'Language', 'Processing', 'NLP', 'is', 'fascinating', 'Let', 's', 'dive', 'into', 'tokenization’]

Auto Tokenization

- Special Cases: Punctuation and Contractions

Contractions like “isn’t” or “Let’s” can be split for finer granularity or retained as single tokens.

- **Example**

```
from nltk.tokenize import word_tokenize
text = "Let's dive into NLP. It's amazing!"
tokens = word_tokenize(text)
print(tokens)
```

Output:

```
['Let', "'s", 'dive', 'into', 'NLP', '.', 'It', "'s", 'amazing', '!']
```

Noisy Text: Regex can help extract meaningful tokens from noisy text such as social media posts

```
import re
text = "NLP #rocks! 🚀 Follow @OpenAI for updates."
tokens = re.findall(r"#\w+|@\w+|[A-Za-z]+", text)
print(tokens)
```

Output:

```
['NLP', '#rocks', 'Follow', '@OpenAI', 'for', 'updates']
```

Manual Tokenization

Regex (Regular Expression) Tokenization: Allows for precise and flexible tokenization, making it a powerful tool for handling domain-specific and complex text scenarios. While automated tokenizers are robust for general NLP tasks, regex shines in extracting specific patterns from text, handling noisy or unstructured data, and offering fine-grained control.

- Extracting Domain-Specific pattern: Customized to extracts tokens such that matches specific formats such as email addresses, phone numbers etc.
  - **Example**

```
import re

text = "Contact us at support@example.com or call +1-800-555-1234. Follow #NLPUpdates!"

# Regex pattern for domain-specific tokens

pattern = r"[A-Za-z]+(?:'[a-z]+)?|@\w+|#\w+|\+?\d{1,2}-\d{3}-\d{3}-\d{4}|\b[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Z|a-z]{2,}\b”

tokens = re.findall(pattern, text)

print("Tokens:", tokens)
```

Output:

```
Tokens: ['Contact', 'us', 'at', 'support@example.com', 'or', 'call', '+1-800-555-1234', 'Follow', '#NLPUpdates']
```

Pattern Explanation:

- @\w+: Captures mentions like @username.
- #\w+: Captures hashtags like #NLPUpdates.
- \+?\d{1,2}-\d{3}-\d{3}-\d{4}: Captures phone numbers in formats like +1-800-555-1234.
- [A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Z|a-z]{2,}: Captures email addresses like support@example.com.

Manual Tokenization

- Handling Noisy Text (Social Media or Logs): Help tokenize text while preserving meaningful tokens like emojis, URLs or special characters
  - **Example**

```
text = "NLP is amazing! 🚀 Visit https://example.com for more info. #AI #MachineLearning"

# Regex pattern for noisy text

pattern = r"#\w+|https?://[^\s]+|[\w']+|[^\w\s]"

tokens = re.findall(pattern, text)

print("Tokens:", tokens)
```

Output:

```
Tokens: ['NLP', 'is', 'amazing', '!', '🚀', 'https://example.com', 'for', 'more', 'info', '.', '#AI', '#MachineLearning']
```

Pattern Explanation:

- #\w+: Captures hashtags.
- https?://[^\s]+: Captures URLs starting with http or https.
- [\w']+: Captures words and contractions (e.g., “isn’t”).
- [^\w\s]: Captures special characters like punctuation and emojis.

Tokenization is not a one-size-fits-all approach; it requires careful consideration based on the task at hand , language, type of text and context we want to retain

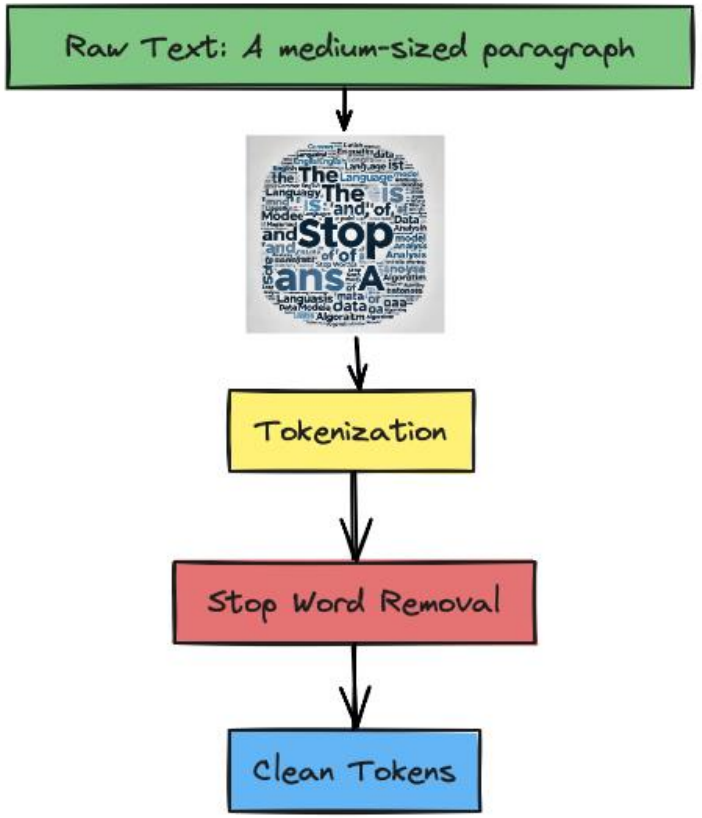
Stop word Removal

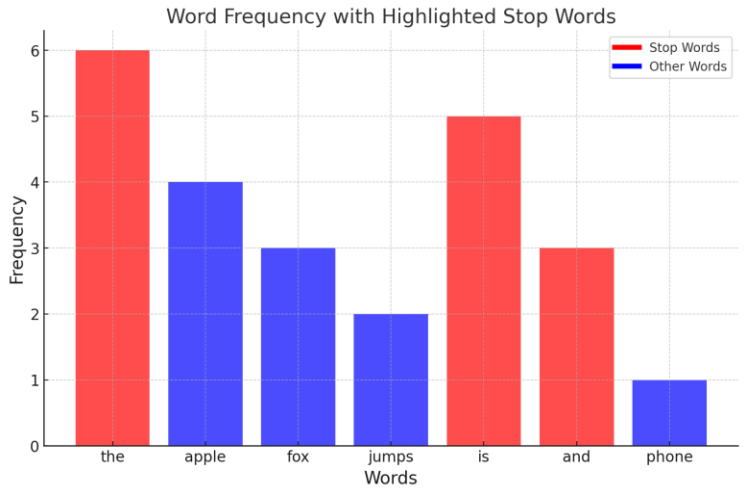
Stop words are high-frequency common words (e.g., “the,” “is,” “and”) that occur frequently in text but often contribute little to the semantic meaning in most NLP tasks

Stop words often constitute ~50% of a text corpus and removing them can significantly reduce noise

Removing stop words reduces the dimensionality of the text data, making NLP models more efficient and accurate

Stop word removal follows tokenization as it splits the text into smaller units (tokens), making it possible to identify stop words





Stop word Removal: Using Python Libraries

Both NLTK & SpaCy provide automated approach to implement Stop words removal

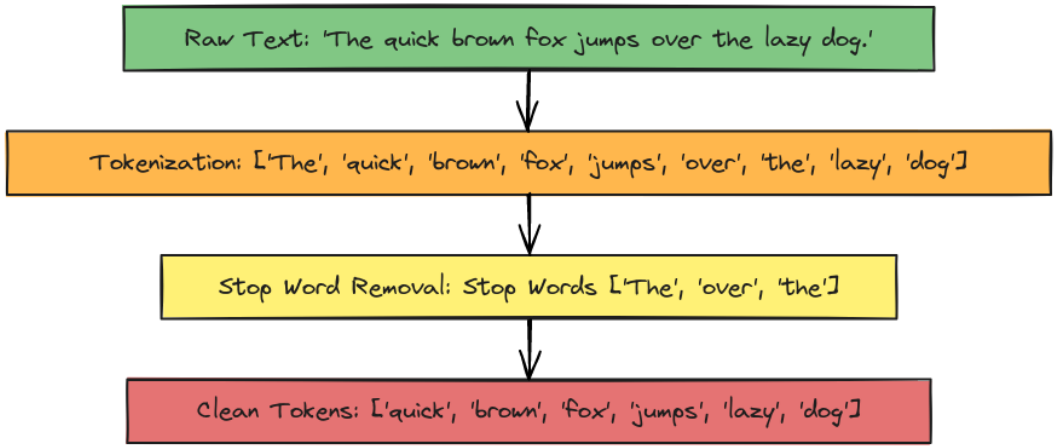
```
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize

text = "The quick brown fox jumps over the lazy dog."
tokens = word_tokenize(text)
stop_words = set(stopwords.words('english'))
filtered_tokens = [word for word in tokens if word.lower() not in stop_words]

print("Filtered Tokens:", filtered_tokens)
```

Output:

["quick", "brown", "fox", "jumps", "lazy", "dog"]



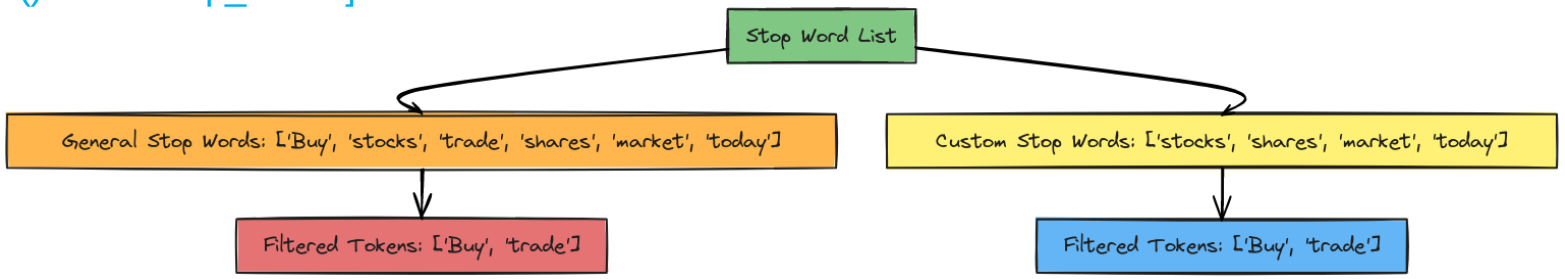
Customizing Stop word Lists

- Sometimes the predefined or general stop-word lists may not suit all domains (e.g., finance, medical)
- Domain-specific tasks require retaining or removing contextually important words.

```
tokens = word_tokenize("Buy stocks and trade shares in the market today.")
stop_words = set(stopwords.words('english'))
filtered_tokens = [word for word in tokens if word.lower() not in stop_words]
print(filtered_tokens)
```

Output:

["Buy", "stocks", "trade", "shares", "market", "today"]



Applying custom stop words:

```
custom_stop_words = stop_words.union({"buy", "trade"})
filtered_tokens = [word for word in tokens if word.lower() not in custom_stop_words]
print(filtered_tokens)
```

Output:

["stocks", "shares", "market", "today"]

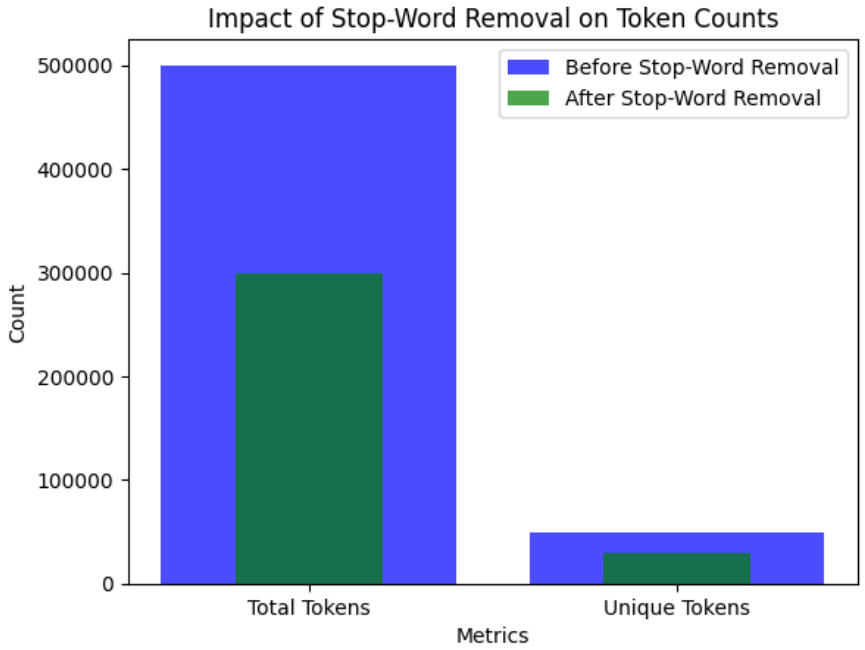
Challenges in Stop word Removal

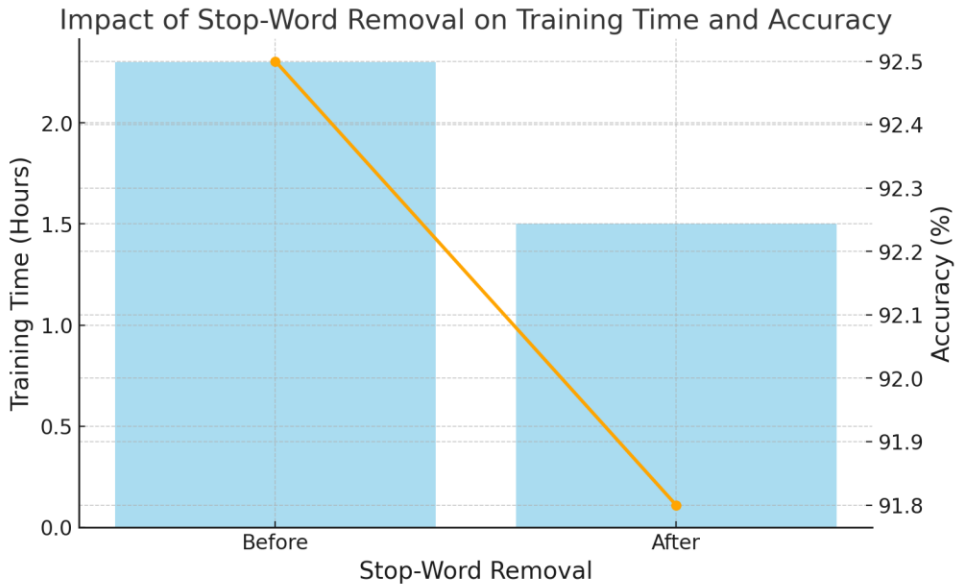
In machine translation, stop words contribute to grammar and therefore the dataset should be analyzed based on specific solutions to retain contextually important stop words

- When Stop Words Add Value: Sentiment Analysis
  - Original: "I do not like this product."
  - After stop word removal: "I like this product."
  - Impact: Removing 'not' changes the sentiment entirely, therefore retain words critical to polarity
- Syntax and Grammar: In Machine Translation, stop words contribute to syntactic accuracy
  - Original: "The cat is on the mat."
  - After stop word removal: "cat mat."
  - Impact: Loss of syntax can lead to incorrect translations, therefore use language-specific stop-words lists
- Potential Over-Filtering
  - General stop-word lists may remove important domain-specific words
  - "Trade stocks and buy low to maximize returns." – Removing domain-specific terms like 'trade' and 'buy' could lead to loss of semantic structure, therefore combine domain expertise with automated tools for customizing lists

Impact of Stop-word Removal

Stop-word removal improves model training efficiency by reducing feature dimensionality.





	Dimensionality Reduction	Training Time (Hours)	Accuracy
Token Before	500000	2.3	92.50%
Tokens After	300000	1.5	91.80%
Reduction	40%	35%	1%

Ref:

- *Speech and Language Processing (Jurafsky & Martin), Chapter 4 (Text Classification and Naive Bayes).*
- *Natural Language Processing with Python, Chapter 6 (Text Classification).*
- *Data Source: Text classification (20 Newsgroups)*

Conclusion

- Stop-Word Removal Improves Efficiency:
  - Reduces dimensionality and noise.
  - Speeds up model training without significant loss in performance.
- Tailoring to Task:
  - Use general lists for standard tasks.
  - Customize for domain-specific needs.
- Critical Thinking:
  - Always analyze the impact of stop-word removal on the specific NLP task.

Key Takeaway: Customizing stop-word lists ensures important words are retained or excluded depending on the task, improving model performance and context comprehension.

Stemming & Lemmatization

Stemming and Lemmatization

Removing **stop words** simplifies the text but retains redundant variations of same word forms (e.g., “running,” “ran,” “runner”). Stemming and lemmatization reduce these variations by converting the words to their root forms, lowering dimensionality and improving model efficiency

Example:

- **Input:** "The runners are running quickly, while one runner ran past the finish line to claim the prize."
- **Tokenization:** ["The", "runners", "are", "running", "quickly", ",", result, "one", "runner", "ran", "past", "the", "finish", "line", "to", "claim", "the", "prize", "."]
- **Stop Word Removal:** ["runners", "running", "quickly", ",", "runner", "ran", "finish", "line", "claim", "prize", "."]
- **Stemming:** ["runner", "run", "quick", ",", "runner", "ran", "finish", "line", "claim", "prize", "."]
- **Lemmatization:** ["run", "run", "quick", ",", "run", "run", "finish", "line", "claim", "prize", "."]

What is Stemming and Lemmatization:

Stemming is the process of reducing words to their root forms by removing suffixes or prefixes based on predefined rules. As a result, it may result in stems that lack linguistic correctness.

Example:

- “runners” → “runner”: Stemming removes the plural suffix “-s.”
- “running” → “run”: The heuristic rule strips the continuous tense suffix “-ing.”
- “runner” → “runner”: The word is already close to its stem.
- “ran” → “ran”: The past tense form isn’t reduced to “run,” as stemming rules don’t always handle irregular verbs.

Stemming

Types of Stemmers:

- Porter Stemmer: Iterative suffix-stripping algorithm with fixed rules
  - Reduces suffixes consistently while preserving the linguistic base
  - Example: “organization” → “organ” (removes -ization but may lose semantic depth)
  - Example: “bunnies” → “bunni” (handles plurals but creates non-dictionary stems)
- Lancaster Stemmer: Aggressive and fast but can over-stem (e.g., “maximum” → “max”)
  - Aggressively reduces words, often over-stemming to non-informative roots
  - Example: “stabilize” → “stab” (removes too much information)
  - Example: “favorable” → “fav” (loss of semantic clarity)
- Snowball Stemmer: An extension of Porter with support for multiple languages.
  - Balances between reduction and semantic preservation
  - Example: “organization” → “organiz” (less aggressive suffix removal)
  - Example: “favorable” → “favor” (retains meaning)

Advantages:

- Fast and computationally efficient
- Useful for search engines and information retrieval i.e., systems where the stemmed results aren’t required to be fed back

Stemmer	Ruleset	Use Case	Example
Porter	Rule-based suffix	Search Engines, indexing	"running" -> "run"
Snowball	Multilingual	International corpora	"running" -> "run"
Lancaster	Aggressive suffix	Fast preprocessing	“maximum” → “max”

Word	Porter Stemmer	Lancaster Stemmer	Snowball Stemmer
bunnies	bunni	bunny	bunni
organization	organ	organ	organiz
polarize	polar	pol	polar
jaguar	jaguar	jaguar	jaguar
stabilize	stabil	stab	stabil
democratic	democrat	democr	democrat
kingdoms	kingdom	king	kingdom
dramatic	dramat	dram	dramat
favorable	favor	fav	favor
international	intern	intern	intern

Lemmatization

Lemmatization is the process of reducing words to their base or dictionary form by considering their meaning and part of speech (POS). Unlike stemming, it ensures that the reduced forms are valid words, making it more accurate for semantic tasks.

Example:

- Maps all forms (“runners,” “running,” “runner,” “ran”) to the base form “run” using linguistic awareness
- Produces context-aware results by resolving irregular forms (e.g., “ran” → “run”) using POS tagging

Types of Lemmatization:

- Dictionary based (e.g., WordNet in NLTK, SpaCy)
- Rule based (e.g., SpaCy)
- Statistical based (e.g.: stanza) trained on large corpora, providing context aware lemma

Word	Porter Stemmer	Lancaster Stemmer	Snowball Stemmer	Lemmatization
bunnies	bunni	bunny	bunni	bunny
organization	organ	organ	organiz	organization
polarize	polar	pol	polar	polarize
jaguar	jaguar	jaguar	jaguar	jaguar
stabilize	stabil	stab	stabil	stabilize
democratic	democrat	democr	democrat	democratic
kingdoms	kingdom	king	kingdom	kingdom
dramatic	dramat	dram	dramat	dramatic
favorable	favor	fav	favor	favorable
international	intern	intern	intern	international

Advantages:

- Accurate and semantically meaningful.
- Essential for tasks requiring precise word relationships such as machine translation and sentiment analysis

Limitations:

- Slower than stemming due to linguistic overhead.
- Requires grammatical context.

Choice of Lemmatization Approach

- Project Requirements:
  - General Text: Use dictionary-based lemmatization (e.g., NLTK, SpaCy).
  - Context-Sensitive Applications: Use statistical lemmatization (e.g., Stanza).
  - Domain-Specific Text: Use Hybrid or custom lemmatization.
- Accuracy vs. Speed:
  - Accuracy: Statistical or hybrid approaches are better for specialized applications.
  - Speed: Rule-based or dictionary-based lemmatizers are faster.
- Linguistic Context:
  - If the project requires POS tagging for accuracy, SpaCy or Stanza is preferable.
- Ease of Use:
  - For beginners or lightweight tasks, libraries like NLTK are simpler to implement.

Morphemes & Graphemes

Language structure is divided into phonetics, phonology, morphology, syntax, and pragmatics. Morphology investigates the formation of words in a language.

Morpheme is the smallest meaningful unit of grammar and cannot be broken down into smaller units. Morphemes are either;

- **Free:** 'Night', 'dog', 'cat', 'girl' words that can stand by themselves and cannot be cut into smaller morphemes. Others are 'and', 'but', 'after' for grammatical structure
- **Bound:** prefixes ('un' in unhappy), suffixes ('s' in pictures, 'ish' in childish) – these morphemes cannot stand by themselves

Lemmatization considers morphemes to find most appropriate base form (semantically meaningful reductions) and uses POS to resolve ambiguities

Grapheme is a letter or a group of letters that make up a single sound. In other words, a grapheme is the written form of a sound.

- Example: the word '***tap***' consists of three graphemes ***t, a, and p***. The word '***trap'*** consists of four graphemes ***t, r, a, and p***.
    - A grapheme can also consist of more than one letter; for example, ***tch*** in catch is a single grapheme because it corresponds to a single sound.

Stemming processes words purely based on simple Suffix rules. Essentially, stemming operates more on the level of graphemes (letters) by chopping off suffixes or prefixes based on patterns without understanding semantics

Ref: https://study.com/academy/lesson/morphemes-examples-definition-types.html

Morphemes & Graphemes Example

Text: "The children are happily playing in the playground while parents are discussing the upcoming events."

Word	Graphemes	Stemmed Output (Porter Stemmer)	Explanation
children	[c, h, i, l, d, r, e, n]	child	Plural suffix ”-ren” removed.
happily	[h, a, p, p, i, l, y]	happi	Adverb suffix ”-ly” stripped.
playing	[p, l, a, y, i, n, g]	play	Verb suffix ”-ing” removed.
playground	[p, l, a, y, g, r, o, u, n, d]	playground	Compound word preserved, no suffix.
parents	[p, a, r, e, n, t, s]	parent	Plural suffix ”-s” removed.
discussing	[d, i, s, c, u, s, s, i, n, g]	discuss	Verb suffix ”-ing” removed.
upcoming	[u, p, c, o, m, i, n, g]	upcom	Truncated incorrectly (”-ing” removed).
events	[e, v, e, n, t, s]	event	Plural suffix ”-s” removed.

Word	Morpheme Breakdown	Lemmatized Output	Explanation
children	[child] (root) + [-ren] (plural suffix)	child	Handles plural correctly using context.
happily	[happy] (root) + [-ly] (adverb suffix)	happy	Recognizes base form despite suffix.
playing	[play] (root) + [-ing] (continuous suffix)	play	Converts verb to its base form.
playground	[play] (root) + [ground] (compound noun)	playground	Recognizes compound structure, no reduction.
parents	[parent] (root) + [-s] (plural suffix)	parent	Handles pluralization correctly.
discussing	[discuss] (root) + [-ing] (continuous suffix)	discuss	Converts verb to base form.
upcoming	[up] (prefix) + [come] (root) + [-ing] (suffix)	upcoming	Keeps word intact; context prevents reduction.
events	[event] (root) + [-s] (plural suffix)	event	Converts plural to singular form.

Ref: https://study.com/academy/lesson/morphemes-examples-definition-types.html

Text Normalization

Text Normalization

After tokenization splits raw text into smaller units (tokens), normalization ensures these units are consistent for further processing.

What is Text Normalization?

Text normalization is the process of transforming text into a uniform and standardized format, making it easier for algorithms to analyze.

- Example 1:
  - Raw Text: "The QUICK Brown FoX jumps Over the lazy DOG."
  - Normalized: "the quick brown fox jumps over the lazy dog"

- Example 2:
  - Tokens like ***"U.S."*** and ***us*** may cause ambiguity.
  - Variants like Running, running, and RUNNING increase vocabulary unnecessarily.

Key Objectives:

- Reduce variability in textual data.
- Ensure consistency in token representation.
- Minimize noise to enhance downstream processing.

Key Techniques in Text Normalization

- Lowercasing:
  - Converts all tokens to lowercase.
  - Example: ”Running”, “RUNNING” → ”running"
- Removing Special Characters:
  - Eliminates non-alphanumeric characters.
  - Example: "Hello, World!" → "Hello World"
- Expanding Contractions:
  - Replaces contractions with their full forms.
  - Example: "can't" → "cannot"
- Numeric Normalization:
  - Standardizes numeric data into a consistent format.
  - Example: "$5" → "five dollars"

Challenges in Text Normalization

- Removing Special Characters: Strips out non-alphanumeric characters.
  - Example:
    - Input: “Hello, World!”
    - Output: “Hello World”
- Semantic Loss:
  - Lowercasing can obscure meaning or can be problematic
  - Example: "US" (United States) vs. "us" (pronoun) and “Apple” (brand” vs, “apple” (fruit)
- Domain-Specific Needs:
  - Financial or medical texts may require preserving symbols like “$” or specific abbreviations
- Handling Numeric Data:
  - Ambiguity in normalizing numbers (e.g., "2024" as a year vs. a count)

Step by Step Example

Input Sentence: "The QUICK Brown FoX jumps Over the lazy DOG."

- **Step 1: Tokenization:**
  - **Tokens:** ["The", "QUICK", "Brown", "FoX", "jumps", "Over", "the", "lazy", "DOG"]
- **Step 2: Lowercasing:**
  - **Normalized Tokens:** ["the", "quick", "brown", "fox", "jumps", "over", "the", "lazy", "dog"]
- **Step 3: Removing Special Characters:**
  - **Example (if applicable):** ["the", "quick", "brown", "fox", "jumps", "over", "the", "lazy", "dog"]

"Normalization reduces variability and noise, leading to a measurable reduction in vocabulary size and enhanced token consistency"

- *Speech and Language Processing, Chapter 2, pages 17–20.*
- *Natural Language Processing with Python, Chapter 3, pages 32–35."*