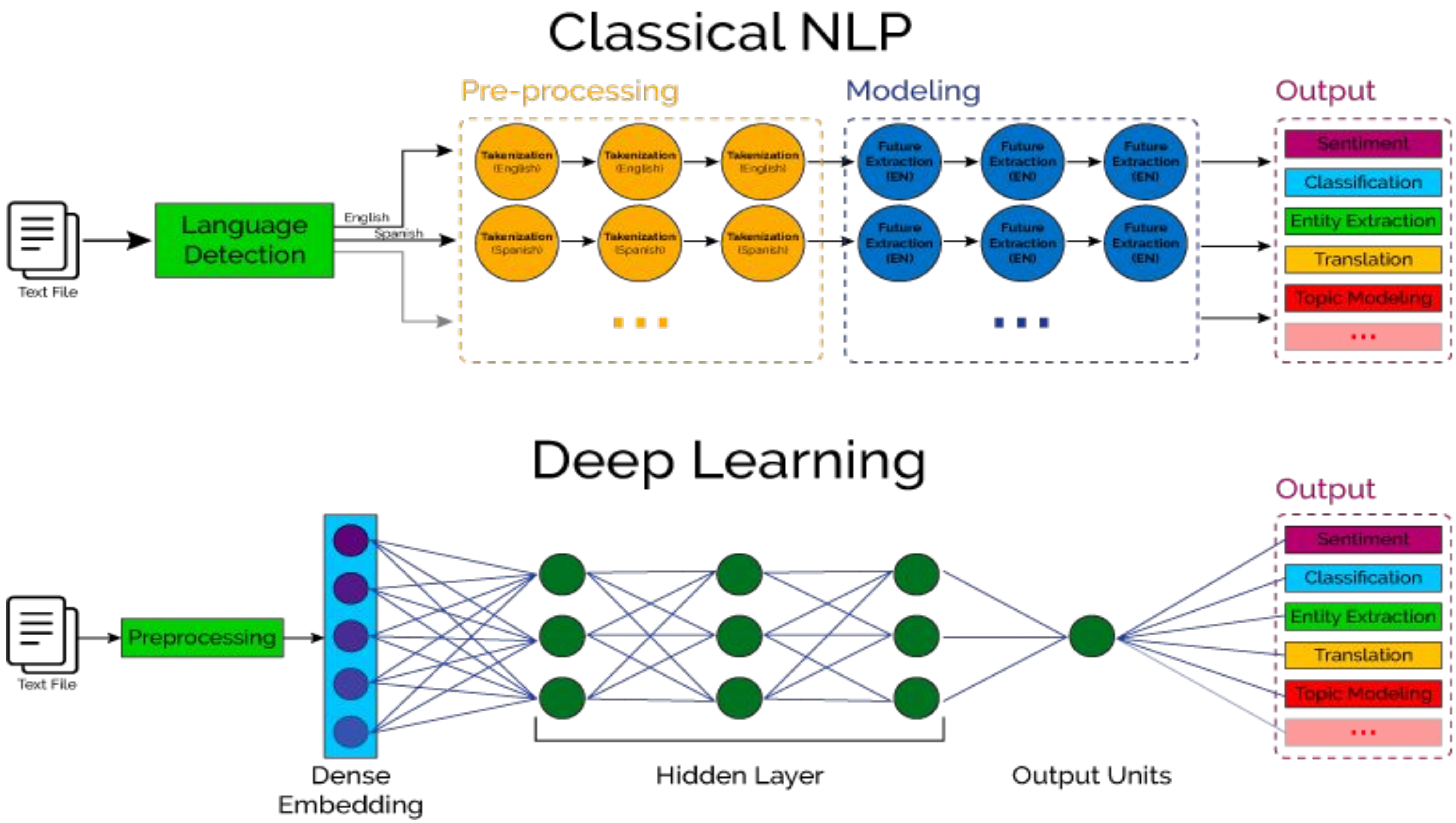
# Unit 1

# Introduction to Natural Language Processing

Prof. Pranjal Pandit
CSE AI

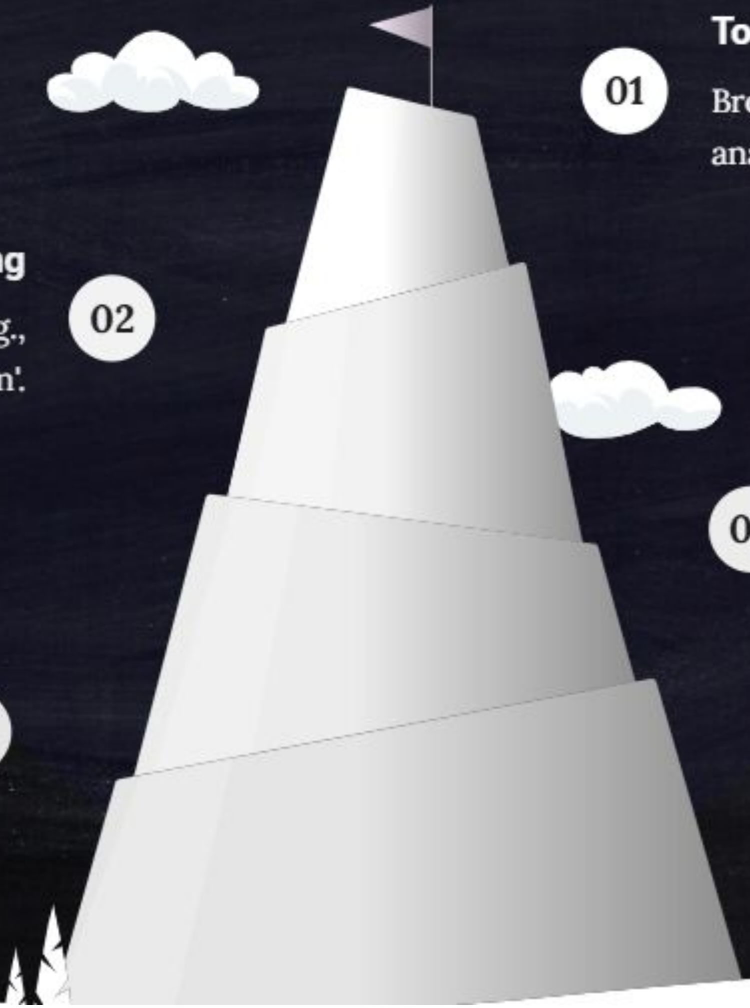
# Unit 1: Introduction to Natural Language Processing

- **Introduction: Scope, Applications and Challenges.**
- **Brief history and evolution of NLP: Programming languages Vs Natural Languages, Are Natural Languages Regular? Finite automata for NLP**
- **Stages of NLP**
- NLP Basics of text processing: Tokenization, Stemming, Lemmatization, Part of Speech Tagging;
- NLP Components: Syntax, Semantics, and Pragmatics;
- Introduction to Regular Expressions and their Applications in NLP.


Classical NLP
Pre-processing
Modeling
Output
Text File
Language Detection
English
Spanish
Takenization (English)
Takenization (Spanish)
Future Extraction (EN)
Sentiment
Classification
Entity Extraction
Translation
Topic Modeling
...
Deep Learning
Output
Text File
Preprocessing
Dense Embedding
Hidden Layer
Output Units
Sentiment
Classification
Entity Extraction
Translation
Topic Modeling
...

# NLP Basics: Tokenization and Beyond

Understanding fundamental NLP techniques



### Tokenization

01

Breaking text into individual words or phrases for analysis.

### Stemming

02

Reducing words to their base or root form, e.g., 'running' to 'run'.

### Lemmatization

03

Converting words to their dictionary form, considering context.

### Parts of Speech Tagging

04

Assigning word types (noun, verb, etc.) to each token in text.

# NLP Basics of Text Processing

**Tokenization**

- **Tokenization in NLP (Natural Language Processing)** refers to the process of breaking down text into smaller units, called tokens.
- These tokens can be words, phrases, characters, or subwords, depending on the granularity required for the task at hand.
- Tokenization is a **fundamental step in text preprocessing** and plays a critical role in preparing textual data for machine learning models.

# Types of Tokenization

**1. Word Tokenization:**

Splitting text into words.

“NLP is amazing!" → [“NLP", "is", "amazing", "!"]

**2. Sentence Tokenization:**

Dividing text into sentences.

"Hello world. Welcome to AI." → ["Hello world.", "Welcome to AI."]

**3. Sub word Tokenization:**

Breaking words into smaller units. Common in NLP for handling rare or unknown words.

"unbelievable" → ["un", "believ", "able"]

**4. Character Tokenization:**

Splitting text into individual characters.

"AI" → ["A", "I"]

# Challenges in Tokenization

## Ambiguity in Language:

Words like "New York" could be one token or two depending on context.

Handling contractions like "don't" (should it be ["do", "n't"] or ["don't"]?)

## Special Character:

Deciding if punctuation marks should be part of tokens or separate. Managing punctuation, symbols, or URLs within text

## Language Complexity:

Languages without clear boundaries (e.g., Chinese, Japanese) make tokenization harder.

## Morphologically Rich Languages:

Languages like Turkish or Finnish, **with many inflected word forms**, require specialized tokenization methods.

# Chinese

- **Nature of Writing**:
  Chinese is written using characters (汉字, *Hanzi*), where each character represents a syllable and often has meaning on its own. Words are typically composed of one or more characters, but there are no spaces between words in standard writing.

- **Example**:
  Sentence: 我喜欢学习语言 (I like learning languages)
  Tokenized: ["我" (I), "喜欢" (like), "学习" (learning/study), "语言" (languages)]

- **Challenge**:
  Identifying **boundaries between multi-character words**, especially when a sequence of characters can form different valid words depending on context.

元来日本語は漢文に倣い、文字を上から下へ、また行を右から左へと進めて表記を行っていた。漢字と仮名の筆順も縦書きを前提としており、横書き不能な書体も存在する。

**Japanese uses Comma(、), Period(。), Quotation marks(「 」)**

**Chinese occasionally uses question mark or exclamation point**

**Korean script uses same punctuation marks as European languages**

모든 인간은 태어날 때부터 자유로우며 그 존엄과 권리에 있어 동등하다. 인간은 천부적으로 이성과 양심을 부여받았으며 서로 형제애의 정신으로 행동하여야 한다.

# Japanese

- **Nature of Writing**:
  Japanese uses a mix of three scripts:
  - **Kanji** (漢字): Logographic characters derived from Chinese.
  - **Hiragana** (ひらがな): Phonetic script for grammatical elements and native words.
  - **Katakana** (カタカナ): Phonetic script for foreign words and emphasis.

- Like **Chinese, Japanese is often written without spaces in sentences**. However, the mixture of scripts can sometimes help in identifying word boundaries.
- **Example**:
  Sentence: 私は日本語を勉強しています (I am studying Japanese)
  Tokenized: ["私" (I), "は" (topic marker), "日本語" (Japanese language), "を" (object marker), "勉強" (study), "しています" (am doing)]
- **Challenge**:
  **Complex grammar** and the **use of multiple scripts make tokenization context-dependent**. For instance, a sequence of Hiragana could represent multiple words or a single grammatical form.

# Tools and Libraries for Tokenization

**1.NLTK (Natural Language Toolkit)**

- Functions: word_tokenize, sent_tokenize

  Example:

```
from nltk.tokenize import word_tokenize
text = "Tokenization is crucial for NLP."
print(word_tokenize(text))
```

**2. spaCy**

Provides efficient tokenization as part of its NLP pipeline.

Example:

```
import spacy
nlp = spacy.load("en_core_web_sm")
doc = nlp("Tokenization with spaCy is easy.")
print([token.text for token in doc])
```

**3.Transformers (Hugging Face)**

- Tokenizers for modern pre-trained models like BERT, GPT, and RoBERTa.

```
from transformers import AutoTokenizer
tokenizer = AutoTokenizer.from_pretrained("bert-base-uncased")
tokens = tokenizer("Tokenization for transformers.")
print(tokens)
```

**4.Stanford CoreNLP**

- A robust suite for tokenization and other NLP tasks.

# Applications of Tokenization

**Text Classification**:
Tokenized words are used as input to classify text into categories (e.g., spam detection).

**Machine Translation**:
Helps convert source language text into tokens for translation models.

**Sentiment Analysis**:
Enables models to analyze opinions in tokenized sentences.

**Search Engines**:
Tokenization assists in indexing and retrieving relevant documents.

**Speech Recognition**:
Tokenizes spoken language for text transcription.

# Stemming in NLP

- Stemming is a text normalization technique in Natural Language Processing (NLP) that reduces words to their root or base form (called the "**stem**"). The stem may not always be a valid word but is intended to capture the essential meaning of related words.

**Why is Stemming Used?**

1. **Reduce Redundancy**: It helps group similar words together for easier analysis, such as "running," "runner," and "runs" being reduced to "run."
2. **Improve Performance**: By reducing words to their stems, search engines, information retrieval systems, and text mining tasks process fewer unique tokens.
3. **Enhance Consistency**: It aligns different forms of a word, making text data cleaner for tasks like classification, clustering, or sentiment analysis.

Stemming algorithms apply heuristic rules to strip suffixes (e.g., "ing," "ed," "es") from words. For example:

**Word**: "Playing"
**Stem**: "Play"

**Word**: "Studies"
**Stem**: "Studi" (not always a valid word)

## Common Stemming Algorithms

1. **Porter Stemmer** (Most Popular): Focuses on removing common English suffixes systematically.
2. **Lancaster Stemmer**: More aggressive, often produces shorter stems.
3. **Snowball Stemmer**: An improved version of the Porter Stemmer, supporting multiple languages.

```python
from nltk.stem import PorterStemmer

# Initialize stemmer
stemmer = PorterStemmer()

# Sample words
words = ["running", "runner", "ran", "runs", "easily", "fairness"]

# Apply stemming
stems = [stemmer.stem(word) for word in words]

# Print stems
print(stems)
```

Output

```
['run', 'runner', 'ran', 'run', 'easili', 'fair']
```

## Limitations of Stemming

1. **Over-Stemming**: Sometimes stems are too aggressive, leading to loss of meaning.
   Example: "Studies" → "Studi“

2. **Under-Stemming**: Some forms of a word may not be reduced to the same root.
   Example: "Relational" and "Relation" remain distinct.

3. **Language Dependency**: Most stemming algorithms are designed for specific languages, e.g., English.

**Why "easily" was stemmed to "easili"?**

- The Porter Stemmer operates using a set of heuristic rules to strip suffixes. It doesn’t rely on vocabulary or dictionary checks, so it doesn’t always produce valid words.
- The suffix **"-ly"** is identified and removed from "easily," resulting in "easili."
  - This is a **mechanical transformation** and doesn't consider that "easily" has a semantic root "easy.“

- **Limitations**: The Porter Stemmer doesn't verify if the stemmed result is meaningful, leading to "easili," which is not a valid word.

**Why "runner" was not further stemmed?**

- The word "runner" **already looks like its base form**, as per the heuristic rules of the Porter Stemmer.
  - The Porter Stemmer focuses on suffix removal (like "-ing", "-ed", "-ly"), but "runner" has the suffix **"-er"**, which is often considered part of the root meaning (e.g., a "doer" or "runner").
  - Aggressively stripping "-er" might lead to incorrect or overly generalized stems, so the algorithm avoids doing so.
- **Contrast with Other Words**:
  - For example, "running" → "run" because "-ing" is a suffix commonly stripped by the rules.
  - "runner" is retained because it's already close to its base form and stripping further might distort its meaning.

- **"easily" → "easili"**: A byproduct of heuristic rules that don’t guarantee meaningful stems.

- **"runner" unchanged**: The algorithm heuristically decides not to stem further to avoid over-stemming.

# Comparison

- Stemming is fast and simple but doesn’t always yield meaningful or accurate results. For more precise processing:
- Use **lemmatization** instead, as it considers the meaning and grammatical role of words.
- Example comparison of stemming vs. lemmatization:

# Example comparison of stemming vs. lemmatization

```
from nltk.stem import PorterStemmer
from nltk.stem import WordNetLemmatizer
import nltk
nltk.download('wordnet')

stemmer = PorterStemmer()
lemmatizer = WordNetLemmatizer()

word = "easily"
print("Stemmed:", stemmer.stem(word))  # Output: easili
print("Lemmatized:", lemmatizer.lemmatize(word))  # Output: easy
```

- Lemmatization is a text normalization technique in **Natural Language Processing (NLP)** that reduces words to their base or canonical form, called the **lemma**.
- Unlike stemming, lemmatization uses a **dictionary or linguistic knowledge** to ensure that the root word is meaningful and grammatically correct.

- Lemmatization considers:

1. **Word Context**: It **uses part-of-speech (POS) tagging** to determine the grammatical role of a word (noun, verb, etc.).
2. **Dictionary Lookup**: It relies on vocabulary and morphological analysis to map inflected forms to their base forms.
    1. Example: "Running" → "Run" (verb) but "Better" → "Good" (adjective).

## Stemming

adjustable → adjust

formality → formaliti

formaliti → formal

airliner → airlin

## Lemmatization

was → (to) be

better → good

meeting → meeting

## Why Lemmatization is Used?

1. **Improves Data Consistency**: Reduces word variations like "running," "ran," and "runs" to their base form "run."
2. **Enhances Accuracy**: Produces grammatically valid words, which is critical for downstream tasks like machine translation, information retrieval, and sentiment analysis.
3. **Language-Specific Normalization**: Lemmatization is adaptable to the linguistic rules of different languages.

# Lemmatization vs. Stemming

| Feature | Lemmatization | Stemming |
|---|---|---|
| **Output** | Produces valid words | May produce invalid words |
| **Approach** | Rule-based (linguistic) | Heuristic (suffix removal) |
| **Accuracy** | High | Moderate |
| **Speed** | Slower | Faster |
| **Example ("better")** | "good" (based on POS) | "better" (unchanged) |

## Common Lemmatization Algorithms

1. **WordNet Lemmatizer**:
    - Uses the WordNet lexical database for linguistic analysis.
    - Requires specifying the POS tag (default is "noun").

2. **SpaCy Lemmatizer**:
    - Built into the SpaCy library and supports multiple languages.
    - Automatically detects POS and applies lemmatization.

## Applications of Lemmatization

1. **Search Engines**: Improves query matching by normalizing search terms.
    1. E.g., "running" and "ran" → "run".
2. **Text Summarization**: Reduces word forms for better semantic understanding.
3. **Sentiment Analysis**: Consistent word forms lead to more accurate sentiment predictions.
4. **Chatbots**: Helps bots understand user input by normalizing variations of words.
5. **Machine Translation**: Ensures grammatically correct translations.

## Advantages of Lemmatization

- Produces meaningful base forms.
- Helps preserve semantics in text analysis.
- Works well for language-specific tasks.

## Limitations of Lemmatization

1. **Slower than Stemming**: Requires POS tagging and dictionary lookups.
2. **Dependency on Context**: May produce incorrect lemmas if the context is misinterpreted.
   1. E.g., "flies" as a noun → "fly" (insect) vs. verb → "fly" (to soar).
3. **Language-Specific**: Needs separate models or databases for each language.

# Understanding Parts of Speech Tagging

The Process of Identifying Words' Roles

### What is Parts of Speech Tagging?

It identifies the grammatical category of each word in a sentence.

### Noun (NN)

Nouns represent people, places, things, or ideas. Example: 'dog', 'city'.

### Verb (VB)

Verbs express actions or states of being. Example 'run', 'is'.

### Adjective (JJ)

Adjectives describe or modify nouns. Example: 'happy', 'blue'.

### Adverb (RB)

Adverbs modify verbs, adjectives, or other adverbs. Example: 'quickly', 'very'.

### Pronoun (PRP)

Pronouns replace nouns to avoid repetition. Example: 'he', 'they'.

# Parts of Speech (POS) Tagging

- POS Tagging is a **fundamental task** in Natural Language Processing (NLP) that involves **labeling each word in a text with its corresponding part of speech** based on its definition and context.

- POS tagging is **essential for many higher-level NLP tasks**, such as parsing, named entity recognition, machine translation, and sentiment analysis.

# Key Components of POS Tagging

1. **Parts of Speech**: Common parts of speech include:

    1. **Noun (NN)**: Represents a person, place, or thing (e.g., "cat," "city").
    2. **Verb (VB)**: Represents an action or state (e.g., "run," "is").
    3. **Adjective (JJ)**: Describes or modifies nouns (e.g., "beautiful").
    4. **Adverb (RB)**: Modifies verbs, adjectives, or other adverbs (e.g., "quickly").
    5. **Pronoun (PRP)**: Replaces nouns (e.g., "he," "they").
    6. **Preposition (IN)**: Indicates relationships between nouns (e.g., "in," "on").
    7. **Conjunction (CC)**: Connects clauses or sentences (e.g., "and," "but").
    8. **Determiner (DT)**: Points to nouns (e.g., "the," "an").
    9. **Interjection (UH)**: Expresses emotions (e.g., "wow!").

**2. Tagging Schemes**:

- Commonly used tagging standards include **Penn Treebank POS tags**, which consist of detailed tags for words.
- Example: "The cat sleeps" → **[**DT, NN, VBZ**]**

**3. Disambiguation**:

Many words have multiple possible tags (e.g., **"book" can be a noun or verb**). POS tagging **uses context to resolve ambiguities.**

# Methods of POS Tagging

1. **Rule-Based Tagging**:
    - Relies on a set of predefined linguistic rules.
    - Example: If a word ends with "-ly," it is likely an adverb.

2. **Statistical Tagging**:
    - Uses machine learning models and probabilities to predict the POS tag.
    - Examples:
        1. **Hidden Markov Models (HMMs)**: Predicts tags based on transition and emission probabilities.
        2. **Maximum Entropy Models**.

3. **Deep Learning Approaches**:

- Leverages neural networks, such as Recurrent Neural Networks **(RNNs),** Long Short-Term Memory **(LSTM), or Transformers.**
- **Requires large annotated datasets** but achieves **state-of-the-art accuracy**.

## Applications of POS Tagging

1. **Text Parsing**: Helps in syntactic and grammatical structure analysis.
2. **Named Entity Recognition (NER)**: Assists in identifying entities like names, locations, etc.
3. **Sentiment Analysis**: Identifies sentiment-revealing adjectives and adverbs.
4. **Machine Translation**: Provides structural insights for accurate translation.

## Example of POS Tagging

- **Input Sentence**:

  "The quick brown fox jumps over the lazy dog."

- **Tagged Output**:

  The/DT quick/JJ brown/JJ fox/NN jumps/VBZ over/IN the/DT lazy/JJ dog/NN.

## Tools for POS Tagging

1. **NLTK** (Natural Language Toolkit): Popular library in Python for basic tagging.
2. **spaCy**: Fast and efficient, with pre-trained models.
3. **Stanford NLP**: Advanced models for POS tagging and syntactic parsing.
4. **Hugging Face Transformers**: For deep learning-based tagging.

# POS Tagging with NLTK

```python
import nltk
from nltk.tokenize import word_tokenize
from nltk import pos_tag

# Download necessary resources
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')

# Input text
text = "The quick brown fox jumps over the lazy dog."

# Tokenize the text
tokens = word_tokenize(text)

# Apply POS tagging
tagged_words = pos_tag(tokens)

# Print the result
print("POS Tags using NLTK:")
print(tagged_words)
```

# POS Tagging with spaCy

```python
import spacy

# Load spaCy's English model
nlp = spacy.load("en_core_web_sm")

# Input text
text = "The quick brown fox jumps over the lazy dog."

# Process the text
doc = nlp(text)

# Extract tokens and their POS tags
print("POS Tags using spaCy:")
for token in doc:
    print(f"{token.text:10} {token.pos_:10} {token.tag_:10}")
```

**NLTK Output**:

[('The', 'DT'), ('quick', 'JJ'), ('brown', 'JJ'), ('fox', 'NN'), ('jumps', 'VBZ'), ('over', 'IN'), ('the', 'DT'), ('lazy', 'JJ'), ('dog', 'NN’)]

Each tuple contains the word and its POS tag:

DT: Determiner

JJ: Adjective

NN: Noun

VBZ: Verb (3rd person singular present)

IN: Preposition

# spaCy Output:

| | | |
|---|---|---|
| The | DET | DT |
| quick | ADJ | JJ |
| brown | ADJ | JJ |
| fox | NOUN | NN |
| jumps | VERB | VBZ |
| over | ADP | IN |
| the | DET | DT |
| lazy | ADJ | JJ |
| dog | NOUN | NN |