NLP Answers (unit 1, 2):

1. What is NLP and it's Features

- Natural Language Processing (NLP) is a subfield of Artificial Intelligence (AI) that allows machines to understand human language.
- NLP analyzes the grammatical structure of sentences as well as the specific meanings of words, then use algorithms to extract meaning and deliver results.
- Its purpose is to create systems that can understand language and perform tasks like translation, spell checking, and topic classification automatically.
- Natural language processing also known as NLP is a subfield of Linguistics, Computer science and Artificial intelligence, used for dealing with textual data.
- NLP is an attempt to give AI an ability of understanding, processing, and generating written and spoken(auditory) data in a similar way a human does.

Here are some key features of NLP:

1. Text Preprocessing

- **Tokenization**: Splitting text into individual units such as words or phrases.
- **Lemmatization and Stemming**: Reducing words to their base or root form.
- **Stopword Removal**: Eliminating common words (e.g., "the," "is") that may not add meaningful information.
- **Part-of-Speech (POS) Tagging**: Identifying grammatical parts of speech in sentences.

2. Text Analysis and Understanding

- **Named Entity Recognition (NER)**: Identifying and classifying entities like names, locations, dates, etc.
- **Dependency Parsing**: Understanding the syntactic structure of a sentence and how words relate to each other.
- **Sentiment Analysis**: Determining the sentiment (positive, negative, neutral) expressed in a piece of text.

3. Language Generation

- **Text Summarization**: Generating a concise summary of a longer text.
- **Machine Translation**: Automatically translating text from one language to another.
- **Text Generation**: Creating human-like text based on input, like chatbots and content creation tools.

4. Speech Processing

- **Speech Recognition**: Converting spoken language into text.
- **Speech Synthesis**: Generating spoken language from text (Text-to-Speech or TTS).

5. Semantic Analysis

- **Word Embeddings**: Mapping words into a continuous vector space where similar words are close together (e.g., Word2Vec, GloVe).
- **Topic Modeling**: Identifying topics present in a set of documents.
- **Semantic Similarity**: Measuring how similar two pieces of text are in meaning.

6. Conversational Interfaces

- **Chatbots and Virtual Assistants**: Systems like Siri, Alexa, and chatbots that engage in human-like conversation.
- **Intent Recognition**: Understanding the goal or intent behind a user's input.
- **Context Management**: Handling and maintaining context during conversations for better interaction.

7. Text Classification

- **Spam Detection**: Classifying text as spam or non-spam (e.g., emails).
- **Document Categorization**: Automatically classifying documents into predefined categories.

8. Question Answering

- Systems that can understand questions in natural language and provide relevant answers, such as search engines and automated customer support.

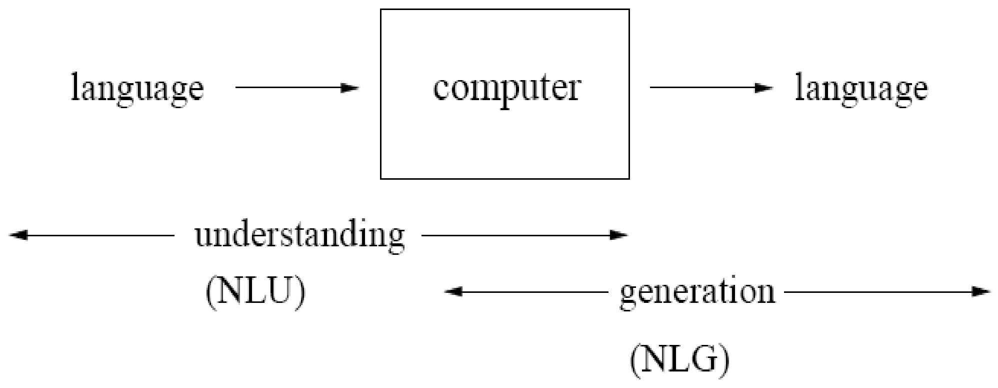
9. Error Handling and Correction

- **Spell Checking**: Detecting and correcting spelling errors.
- **Grammar Correction**: Identifying and fixing grammatical mistakes in text.

10. Multilingual Processing

- NLP supports processing across various languages, allowing translation, understanding, and generation of text in multiple languages.

2. Components of NLP:



- Natural Language Understanding (NLU)
- Natural Language Generation (NLG)

Natural language understanding (NLU):

Natural Language Understanding (NLU) is a field of computer science which analyses what human language means, rather than simply what individual words say.

Natural Language Understanding seeks to intuit many of the connotations and implications that are innate in human communication such as the emotion, effort, intent, or goal behind a speaker's statement. It uses algorithms and artificial intelligence, backed by large libraries of information, to understand our language.

Rather than relying on computer language syntax, Natural Language Understanding enables computers to comprehend and respond accurately to the sentiments expressed in natural language text.

Example of NLU: A virtual assistant like Siri or Google Assistant can understand commands like "What's the weather like today?" Here, the system needs to recognize that the user is asking for a weather forecast for a specific time ("today") and possibly a location.

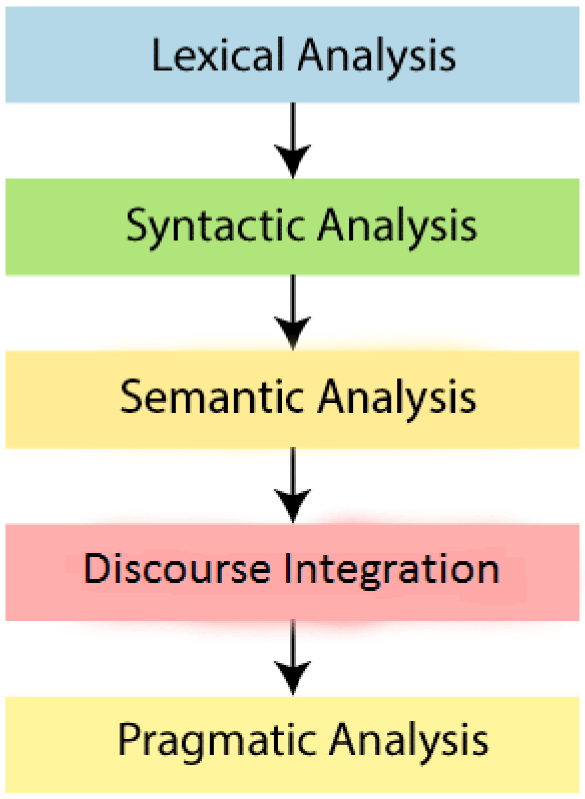
Natural language generation (NLG):

Natural language generation (NLG) is the use of artificial intelligence (AI) programming to produce written or spoken narratives from a data set. NLG is related to human-to-machine and machine-to-human interaction, including computational linguistics, natural language processing (NLP) and natural language understanding (NLU).

Research about NLG often focuses on building computer programs that provide data points with context. Sophisticated NLG software can mine large quantities of numerical data, identify patterns and share that information in a way that is easy for humans to understand. The speed of NLG software is especially useful for producing news and other time-sensitive stories on the internet. At its best, NLG output can be published verbatim as web content.

Example of NLG: A weather chatbot could respond to the query above by saying, "The weather today is sunny with a high of 25°C." Here, the system is generating a human-readable and conversational response based on weather data.

3. Phases of NLP:



1-Morphological Analysis/ Lexical Analysis:

- Morphological or Lexical Analysis deals with text at the individual word level.
- It looks for morphemes, the smallest unit of a word.
- For example, irrationally can be broken into ir (prefix), rational (root) and -ly (suffix).
- Lexical Analysis finds the relation between these morphemes and converts the word into its root form.
- A lexical analyzer also assigns the possible Part-Of-Speech (POS) to the word.
- It takes into consideration the dictionary of the language.
- For example, the word "character" can be used as a noun or a verb.

2-Syntax Analysis:

- Syntax Analysis ensures that a given piece of text is correct structure.
- It tries to parse the sentence to check correct grammar at the sentence level.
- Given the possible POS generated from the previous step, a syntax analyzer assigns POS tags based on the sentence structure.
- Correct Syntax: Sun rises in the east.
- Incorrect Syntax: Rise in sun the east.

3-Semantic Analysis:

- Consider the sentence: "The apple ate a banana".
- Although the sentence is syntactically correct, it doesn't make sense because apples can't eat.
- Semantic analysis looks for meaning in the given sentence.
- It also deals with combining words into phrases.
- For example, "red apple" provides information regarding one object; hence we treat it as a single phrase.
- Similarly, we can group names referring to the same category, person, object or organization.
- "Robert Hill" refers to the same person and not two separate names – "Robert" and "Hill".

4-Discourse analysis:

- Discourse deals with the effect of a previous sentence on the sentence in consideration.
- In the text, "Jack is a bright student. He spends most of the time in the library."
- Here, discourse assigns "he" to refer to "Jack".

5-Pragmatics analysis:

- The final stage of NLP, Pragmatics interprets the given text using information from the previous steps.

- Given a sentence, “Turn off the lights” is an order or request to switch off the lights.

4. Ambiguity in NLP:

Ambiguity in Natural Language Processing (NLP) refers to situations where a word, phrase, sentence, or text has multiple possible interpretations. Human language is inherently ambiguous, and dealing with this is one of the key challenges for NLP systems. Here are the different types of ambiguity commonly encountered in NLP:

1. Lexical Ambiguity:

The word “lexical” refers to words or vocabulary. “Lexical ambiguity” is when the meaning of a word is unclear. It’s also known as semantic ambiguity. Lexical ambiguity happens when a word has more than one meaning, causing a word or phrase to be interpreted differently from how the speaker or writer intended. It typically occurs when the reader or listener doesn’t have the same context as the writer or speaker.

Here are a few examples of lexical ambiguity.

- It was cool.

In this case, “cool” could either mean a chilly temperature, or it could mean good or interesting.

- It’s on the house.

Without enough context, “on the house” could mean something is free, or an object could literally be found on the house.

- The club is dead.

In this case, “dead” could mean the club has closed down, or it could mean that the club is lacking an exciting environment.

- There’s a bat in the attic.

In this sentence, “bat” could refer to an animal, or it could refer to the wooden object that’s used in baseball.

2. Syntactic or Structural Ambiguity:

The word syntax refers to how words are arranged in a sentence. Ambiguity is when something is unclear or open to interpretation. Syntactic ambiguity, also known as structural ambiguity, is when a sentence can be interpreted in two (or more) ways due to the structure of the sentence. Syntactic ambiguity in **writing** is undesirable as it can confuse your readers.

Here are some examples of syntactic ambiguity:

- The duck is ready for dinner.

This sentence could have two different meanings. It could mean that someone's pet duck is ready to eat dinner, or it could mean someone has cooked a duck for dinner.

- I saw the man wearing my jeans.

In this sentence, it could mean that I saw a man wearing the same jeans that I own, or it could mean that I was wearing jeans when I saw the man.

- Sarah bought green shirts and shoes.

This sentence could mean that both the shirts and shoes that Sarah bought were green, or it could mean that just the shirts were green.

- Andrew cooked the chicken with rice.

This sentence could mean that Andrew cooked chicken and served it with rice. It could also mean that rice was used to cook the chicken.

3. Referential Ambiguity:

Referential Ambiguity occurs when it is unclear which entity or object a word or phrase refers to in a sentence, typically involving pronouns or other referential terms like "this," "that," "he," "she," or "it." This type of ambiguity arises because multiple potential referents exist, making it difficult to determine the specific one being referenced without further context.

Example of Referential Ambiguity:

- **Sentence**: "John told David that he won the game."
  - **Ambiguity**: It is unclear whether "he" refers to John or David. Did John win the game or did David?

        - To resolve the ambiguity, additional context would be needed. For instance:
            - "John told David that *John* won the game." (Now it's clear that John is the winner.)
            - "John told David that *David* won the game." (Now it's clear that David is the winner.)

4. **Vagueness and Imprecision:**

Vagueness:
Vagueness refers to a lack of specificity or detail in a statement, making it unclear or open to interpretation. A vague statement provides insufficient information to fully understand the meaning, often because the terms used are broad or undefined.
Example of Vagueness:

- **Sentence**: "She'll arrive sometime soon."
    - **Vagueness**: The term "soon" is vague because it doesn't provide a specific time. "Soon" could mean minutes, hours, or even days, depending on the context, leaving the listener unclear about when exactly she will arrive.

Imprecision:
Imprecision refers to a lack of exactness in a statement. The statement is not entirely wrong, but it lacks accuracy or is not specific enough, leading to a general or approximate idea instead of a clear, defined one.
Example of Imprecision:

- **Sentence**: "The room is about 20 feet wide."
    - **Imprecision**: The word "about" indicates that the width is not exactly 20 feet. It might be 19.5 or 21 feet, but the speaker isn't giving an exact measurement. While the meaning is fairly clear, the lack of precision leaves some uncertainty about the true size.

Key Difference:

- **Vagueness** leaves a statement open to multiple interpretations because it lacks detail or specificity (e.g., "soon").
- **Imprecision** provides an approximation but is not fully accurate (e.g., "about 20 feet").

5. Remove Stopwords and removal of Punctuations, URL, E-mails etc.

Removing stopwords, punctuation, URLs, emails, and similar elements from text is a common preprocessing step in natural language processing (NLP). Here's an explanation of each:

1. Removing Stopwords

Stopwords are common words in a language that carry little meaningful content on their own but serve grammatical purposes. Examples include "is," "the," "and," "in," etc. Removing them simplifies text analysis by focusing on the more informative words.

- **Why it's done**: Stopwords occur frequently and are often not useful for tasks like text classification, sentiment analysis, or topic modeling. Removing them reduces the text size and computational effort.
- **Example**:
  - Original sentence: "The cat is sitting on the mat."
  - After removing stopwords: "cat sitting mat."

2. Removing Punctuation

Punctuation marks (e.g., periods, commas, question marks) help structure sentences but don't convey meaningful content in many text analysis tasks. They are often removed to focus on the actual words.

- **Why it's done**: Punctuation does not contribute to the semantic content in tasks like sentiment analysis, text summarization, or topic modeling. Removing it reduces noise.
- **Example**:
  - Original sentence: "Hello, world! How are you?"
  - After removing punctuation: "Hello world How are you"

3. Removing URLs

URLs are web links that often appear in text data, particularly from social media, blogs, or web scraping. URLs are typically not relevant to the meaning or analysis of text.

- **Why it's done**: URLs are often irrelevant and can distract from the actual content. Removing them reduces noise and improves processing efficiency.
- **Example**:
  - Original sentence: "Check out this website: https://example.com"
  - After removing the URL: "Check out this website"

4. Removing E-mails

Email addresses often appear in text, especially when dealing with communication records like email data. However, they are usually not important for textual analysis.

- **Why it's done**: Similar to URLs, emails don't contribute to the semantic content of the text and are usually noise.
- **Example**:

    - Original sentence: "Contact me at john.doe@example.com"
    - After removing the email: "Contact me at"

5. Removing Special Characters

Special characters include symbols like @, #, $, etc., which are often used for formatting or social media tagging (hashtags, mentions) but do not carry useful information for general text analysis.

- **Why it's done**: They can clutter the text and make it harder to process for certain NLP tasks. Removing them simplifies the text.
- **Example**:
    - Original sentence: "This product costs $50 and it's great!"
    - After removing special characters: "This product costs 50 and its great"

6. Removing Numbers

Numbers can sometimes be irrelevant for certain NLP tasks (like sentiment analysis or text classification) where they don't contribute to understanding the sentiment or topic.

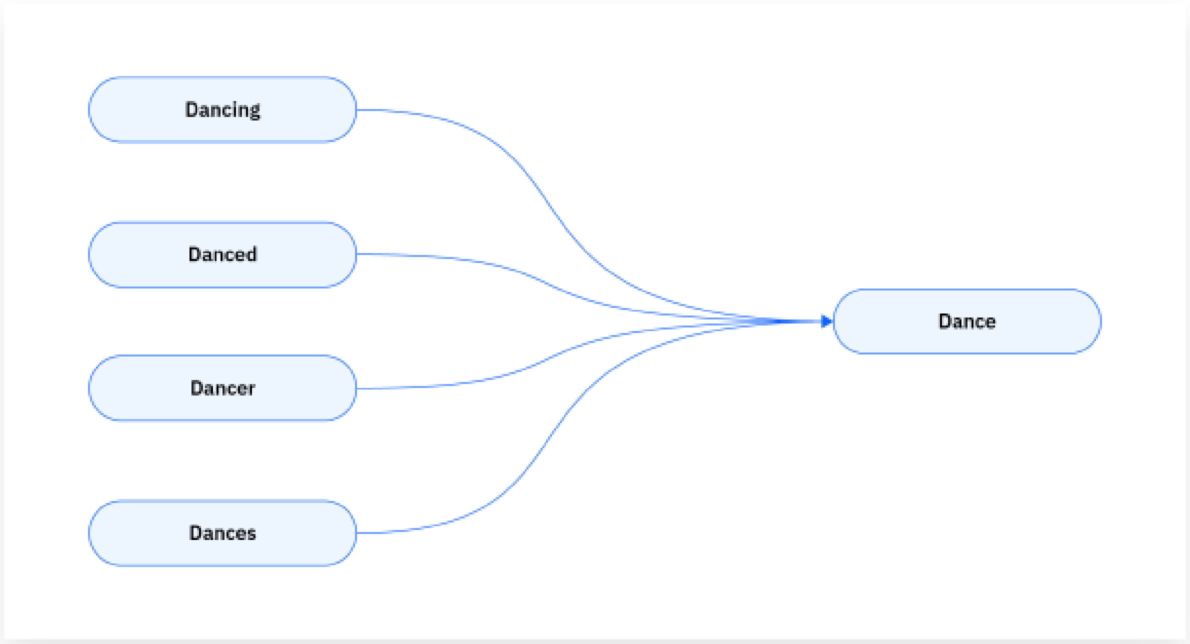
- **Why it's done**: Numbers might not be useful unless they carry meaning in specific contexts like financial analysis.
- **Example**:
    - Original sentence: "I bought 10 apples."
    - After removing numbers: "I bought apples"

6. Stemming

Stemming, in Natural Language Processing (NLP), refers to the process of reducing a word to its word stem that affixes to suffixes and prefixes or the roots.

While a stemming algorithm is a linguistic normalization process in which the variant forms of a word are reduced to a standard form. It is a technique used to extract the base form of the words by removing affixes from them. It is just like cutting down the branches of a tree to its stems. For example, the stem of the words "eating," "eats," "eaten" is "eat."

Search engines use stemming in NLP for indexing the words. That's why rather than storing all forms of a word, a search engine can store only the stems. In this way, stemming reduces the size of the index and increases retrieval accuracy.



What are the errors that could occur in stemming?

There are mainly two errors that could occur in stemming:

1. **Over-stemming -** When two words are stemmed from the same root that are of different stems. Over-stemming can also be regarded as false positives
2. **Under-stemming -** When two words are stemmed from the same root that are not of different stems. Under-stemming can be interpreted as false negatives.

Code:

```
import nltk

from nltk.stem import PorterStemmer

# Download NLTK resources (only required the first time)

nltk.download('punkt')

# Example text
```

```
text = "I am learning to write Python programs. I wrote a program yesterday and will be writing more programs tomorrow."

# Tokenize the text into words

words = nltk.word_tokenize(text)

# Initialize the Porter Stemmer

stemmer = PorterStemmer()

# Apply stemming to each word in the text

stemmed_words = [stemmer.stem(word) for word in words]

# Output the original and stemmed words

print("Original words:", words)

print("Stemmed words:", stemmed_words)
```

<u>**Output:**</u>

```
Original words: ['I', 'am', 'learning', 'to', 'write', 'Python', 'programs', '.', 'I', 'wrote', 'a', 'program', 'yesterday', 'and', 'will', 'be', 'writing', 'more', 'programs', 'tomorrow', '.']

Stemmed words: ['I', 'am', 'learn', 'to', 'write', 'python', 'program', '.', 'I', 'wrote', 'a', 'program', 'yesterday', 'and', 'will', 'be', 'write', 'more', 'program', 'tomorrow', '.']
```

7. Lemmatization:

Lemmatization is a natural language processing (NLP) technique that reduces words to their base or dictionary form, called the **lemma**. Unlike **stemming**, which cuts off word endings to get to a base form (which may not always be a valid word), lemmatization takes into account the word's meaning and context, resulting in grammatically correct root forms.

For example:

- **"running" → "run"**
- **"better" → "good"**
- **"children" → "child"**

Key Differences from Stemming:

- **Stemming**: Produces root forms by chopping off suffixes, which can sometimes result in non-words (e.g., "happily" → "happi").
- **Lemmatization**: Uses a dictionary and the part of speech (POS) of the word to find the proper root form (e.g., "better" → "good" based on its meaning, not just its spelling).

Example of Lemmatization:

For the word **"dogs"**:

- **Stemming**: May reduce it to **"dog"**.
- **Lemmatization**: Recognizes that the lemma of **"dogs"** is **"dog"**.

For the word **"better"**:

- **Stemming**: Might result in **"bet"**.
- **Lemmatization**: Correctly identifies the lemma as **"good"** because it understands the meaning of the word.

Lemmatization is generally more accurate but computationally more expensive than stemming because it involves looking up word forms in a lexical database like WordNet and determining the word's part of speech.

Use Cases:

- **Text Analysis**: To normalize words so that similar terms (run, running, runs) are treated the same.
- **Search Engines**: Improves search accuracy by mapping different forms of a word to a common root.
- **Machine Translation**: Helps by converting words into their root forms for better translation across languages.

Code:

```
import spacy

# Load the English model

nlp = spacy.load('en_core_web_sm')

# Example text

sentence = "The children are playing in the gardens"
```

```
# Process the sentence

doc = nlp(sentence)

# Lemmatize the sentence

lemmatized_sentence = [token.lemma_ for token in doc]

print("Lemmatized sentence:", " ".join(lemmatized_sentence))
```

<u>Output:</u>

```
Lemmatized sentence: The child be play in the garden
```

8. Tokenization:

Tokenization is a fundamental process in natural language processing (NLP) where text is broken down into smaller units called **tokens**. These tokens can be words, phrases, or even characters, depending on the level of granularity required. Tokenization is the first step in many NLP tasks such as text mining, machine translation, and sentiment analysis.

Types of Tokenization:

1. **Word Tokenization**: Splits the text into individual words. This is the most common form of tokenization.
    - Example:
        - Input: "The cat sat on the mat."
        - Output: ["The", "cat", "sat", "on", "the", "mat", "."]
2. **Sentence Tokenization**: Splits the text into individual sentences.
    - Example:
        - Input: "Hello! How are you?"
        - Output: ["Hello!", "How are you?"]
3. **Character Tokenization**: Breaks down the text into individual characters.
    - Example:
        - Input: "Hello"
        - Output: ['H', 'e', 'l', 'l', 'o']

Challenges in Tokenization:

- **Handling punctuation**: Deciding whether to separate punctuation marks from words or treat them as part of the word.
    - Example: Should "I'm" be split into ["I", "'m"] or treated as a single token?

- **Handling contractions**: In English, contractions like "can't" can be tricky. Some tokenizers split it into ["can", "'t"], while others treat it as one token.
- **Language-specific issues**: Languages like Chinese or Japanese don't use spaces to separate words, making word tokenization more complex.

Tokenization Approaches:

1. **Rule-based Tokenization**: Uses predefined rules (like spaces or punctuation) to split the text. It's simple but may not handle edge cases well.
2. **Statistical Tokenization**: Uses machine learning models that are trained on large datasets to better handle ambiguities in tokenization.
3. **Subword Tokenization**: Splits words into smaller meaningful units. This is useful for handling rare or unknown words.
    - Example: Byte-Pair Encoding (BPE) and WordPiece tokenization (used in models like BERT) tokenize a word into subwords (e.g., "unbelievable" might become "un", "believe", "able").

Applications of Tokenization:

- **Text Preprocessing**: Tokenization is often the first step in preparing text for further analysis, like removing stopwords or applying stemming/lemmatization.
- **Search Engines**: Tokenization helps in indexing text and improving search accuracy.
- **Machine Translation**: Tokenizing text is crucial for aligning words and phrases across different languages for accurate translations.

Code:

```
import spacy

# Load the small English model

nlp = spacy.load("en_core_web_sm")

text = "Tokenization is the first step in text processing."

# Process the text

doc = nlp(text)

# Tokenize the text

tokens = [token.text for token in doc]

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

<u>Output:</u>

Tokens: ['Tokenization', 'is', 'the', 'first', 'step', 'in', 'text', 'processing', '.']

9. Part of Speech Tagging:

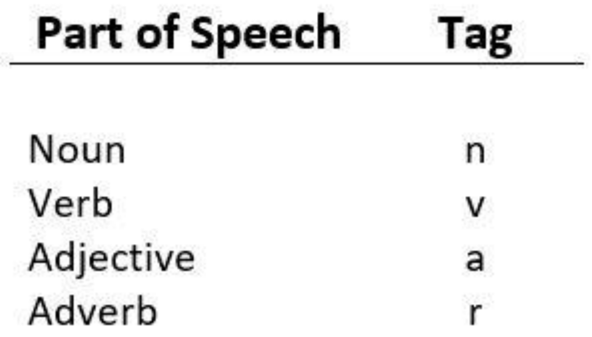
One of the core tasks in **Natural Language Processing (NLP)** is **Parts of Speech (PoS) tagging**, which is giving each word in a text a grammatical category, such as nouns, verbs, adjectives, and adverbs. Through improved comprehension of phrase structure and semantics, this technique makes it possible for machines to study and comprehend human language more accurately.

In many NLP applications, including machine translation, sentiment analysis, and information retrieval, PoS tagging is essential. PoS tagging serves as a link between language and machine understanding, enabling the creation of complex language processing systems and serving as the foundation for advanced linguistic analysis.

What is POS(Parts-Of-Speech) Tagging?

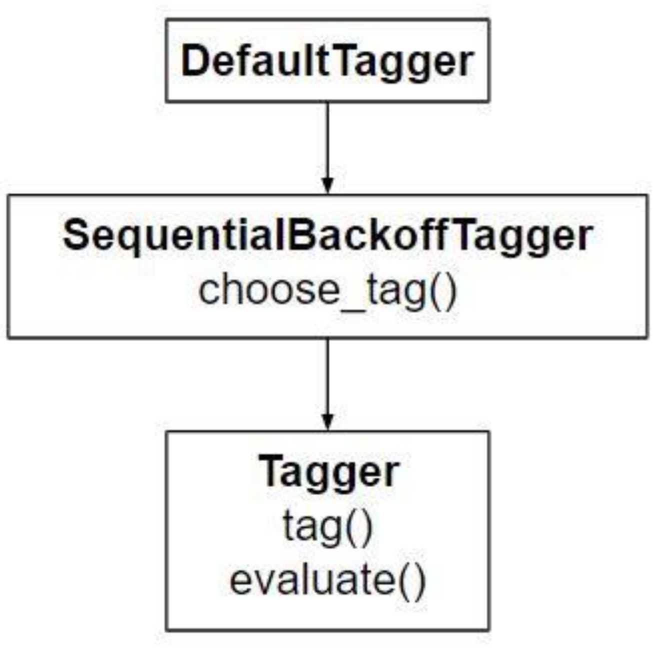
Parts of Speech tagging is a linguistic activity in Natural Language Processing (NLP) wherein each word in a document is given a particular part of speech (adverb, adjective, verb, etc.) or grammatical category. Through the addition of a layer of syntactic and semantic information to the words, this procedure makes it easier to comprehend the sentence's structure and meaning.

In NLP applications, POS tagging is useful for machine translation, named entity recognition, and information extraction, among other things. It also works well for clearing out ambiguity in terms with numerous meanings and revealing a sentence's grammatical structure.



Part of Speech	Tag
Noun	n
Verb	v
Adjective	a
Adverb	r

Default tagging is a basic step for the part-of-speech tagging. It is performed using the DefaultTagger class. The DefaultTagger class takes 'tag' as a single argument. **NN** is the tag for a singular noun. DefaultTagger is most useful when it gets to work with most common part-of-speech tag. that's why a noun tag is recommended.



Example of POS Tagging

Consider the sentence: “The quick brown fox jumps over the lazy dog.”

After performing POS Tagging:

- “The” is tagged as determiner (DT)
- “quick” is tagged as adjective (JJ)
- “brown” is tagged as adjective (JJ)
- “fox” is tagged as noun (NN)
- “jumps” is tagged as verb (VBZ)
- “over” is tagged as preposition (IN)
- “the” is tagged as determiner (DT)
- “lazy” is tagged as adjective (JJ)
- “dog” is tagged as noun (NN)

By offering insights into the grammatical structure, this tagging aids machines in comprehending not just individual words but also the connections between them inside a phrase. For many NLP applications, like text summarization, sentiment analysis, and machine translation, this kind of data is essential.

Workflow of POS Tagging in NLP(only for end term)

The following are the processes in a typical natural language processing (NLP) example of part-of-speech (POS) tagging:

- Tokenization: Divide the input text into discrete tokens, which are usually units of words or subwords. The first stage in NLP tasks is tokenization.

- **Loading Language Models:** To utilize a library such as NLTK or SpaCy, be sure to load the relevant language model. These models offer a foundation for comprehending a language's grammatical structure since they have been trained on a vast amount of linguistic data.

- **Text Processing**: If required, preprocess the text to handle special characters, convert it to lowercase, or eliminate superfluous information. Correct PoS labeling is aided by clear text.

- **Linguistic Analysis**: To determine the text's grammatical structure, use linguistic analysis. This entails understanding each word's purpose inside the sentence, including whether it is an adjective, verb, noun, or other.

- **Part-of-Speech Tagging:** To determine the text's grammatical structure, use linguistic analysis. This entails understanding each word's purpose inside the sentence, including whether it is an adjective, verb, noun, or other.

- **Results Analysis:** Verify the accuracy and consistency of the PoS tagging findings with the source text. Determine and correct any possible problems or mistagging.

Code:

```
import nltk

from nltk import word_tokenize

# Download necessary NLTK data files

nltk.download('punkt')

nltk.download('averaged_perceptron_tagger')

# Sample sentence
```

```
sentence = "The quick brown fox jumps over the lazy dog."

# Tokenize the sentence into words

words = word_tokenize(sentence)

# Perform POS tagging

pos_tags = nltk.pos_tag(words)

# Output the tokens with their corresponding POS tags

print(pos_tags)
```

Output:

```
[('The', 'DT'), ('quick', 'JJ'), ('brown', 'JJ'), ('fox', 'NN'),

('jumps', 'VBZ'), ('over', 'IN'), ('the', 'DT'), ('lazy', 'JJ'), ('dog', 'NN')]
```

10. Named Entity Recognition and Visualization using Displacy

-Application

-Text summarization

Named Entity Recognition (NER) is a natural language processing (NLP) technique that identifies and classifies entities in text into predefined categories such as people, organizations, locations, dates, and more. Tools like **spaCy** provide built-in capabilities for NER and visualization using **displaCy**, a powerful visualization tool to highlight named entities within a text.

Here’s a breakdown of the key concepts and applications:

Named Entity Recognition (NER)

NER is a key task in NLP that seeks to locate and categorize **named entities** into specific categories such as:

- **Person names** (e.g., "Barack Obama")
- **Organizations** (e.g., "Google")
- **Locations** (e.g., "New York City")

- **Dates** (e.g., "2024-10-01")
- **Monetary values** (e.g., "$100 million")

Applications of NER:

1. **Information Extraction**: Automatically extracting names, places, dates, and other key entities from documents for further analysis.
2. **Search Optimization**: Improving search engines by recognizing and focusing on relevant entities.
3. **Customer Feedback Analysis**: Identifying key subjects or entities mentioned in customer reviews or surveys.
4. **Text Summarization**: Recognizing key entities and their relationships, helping to summarize content by focusing on significant information.
5. **Healthcare and Biomedical Text Mining**: Extracting medical conditions, drug names, or genes from clinical or scientific documents.

Visualization using DisplaCy

DisplaCy is a web-based visualization tool included with spaCy that allows users to easily visualize dependency parses and named entities in a text.

Example of NER Visualization:

You can visualize NER using displaCy with just a few lines of code in Python:

```
import spacy

from spacy import displacy

# Load the pre-trained NER model

nlp = spacy.load("en_core_web_sm")

# Input text for NER

text = "Apple is planning to open a new office in San Francisco by next month."

# Apply the NER model

doc = nlp(text)

# Visualize named entities using displaCy

displacy.render(doc, style="ent", jupyter=True)
```

This code processes the text and highlights entities like **"Apple"** as an organization and **"San Francisco"** as a location in the output.

Customizing DisplaCy Visualizations:

You can also customize the colors of entities or add labels for different types of entities, making it easier to interpret the results visually. This is especially useful in reports or presentations.

Text Summarization:

Text Summarization is the task of creating a concise summary of a longer text. There are two main types of text summarization:

1. **Extractive Summarization**: Involves selecting key sentences, phrases, or parts directly from the source text. It doesn't generate new content but extracts the most important points.
   - **Use case**: Generating bullet points from articles or documents.
2. **Abstractive Summarization**: Generates new sentences that convey the core meaning of the original text. This approach uses deep learning models like transformers to understand the text and generate human-like summaries.
   - **Use case**: Summarizing news articles, research papers, or long documents with more nuanced, fluent summaries.

Applications of Text Summarization:

1. News Aggregators: Providing brief news snippets for users to quickly understand key information.
2. Legal Document Summarization: Condensing lengthy contracts, case studies, or legal proceedings into digestible summaries.
3. Customer Service: Summarizing customer conversations for faster resolution of issues.
4. Academic Research: Condensing long research papers or books for quick review.

11. Feature Extraction:

1. Bag of Words:

Bag of words is a Natural Language Processing technique of text modelling. In technical terms, we can say that it is a method of feature extraction with text data. This approach is a simple and flexible way of extracting features from documents.

A bag of words is a representation of text that describes the occurrence of words within a document. We just keep track of word counts and disregard the grammatical details and the word order. It is called a “bag” of words because any information about the order or structure of words in the document is discarded. The model is only concerned with whether known words occur in the document, not where in the document.

Key Concepts:

1. **Vocabulary Creation**:
   - The first step in BoW is to create a vocabulary of unique words from the entire corpus (the collection of documents). Each unique word is assigned an index.
2. **Tokenization**:
   - The text is split into individual words (tokens).
3. **Vector Representation**:
   - Each document is then represented as a vector based on the presence (or absence) of words in the vocabulary.
   - The vector can be binary (1 if the word is present, 0 if absent) or can use frequency counts (the number of times the word appears in the document).

Example of Bag of Words

Consider the following three sentences:

1. "I love programming."
2. "I love coding."
3. "Programming is fun."

Step 1: Create Vocabulary

From the above sentences, the vocabulary might be:

- ["I", "love", "programming", "coding", "is", "fun"]

Step 2: Vector Representation

Now, each sentence can be represented as a vector based on the vocabulary:

1. **"I love programming."**
   - Vector: [1, 1, 1, 0, 0, 0] (corresponding to "I", "love", "programming", "coding", "is", "fun")
2. **"I love coding."**
   - Vector: [1, 1, 0, 1, 0, 0]
3. **"Programming is fun."**
   - Vector: [0, 0, 1, 0, 1, 1]

Advantages of Bag of Words

- **Simplicity**: The model is straightforward to implement and understand.
- **Efficiency**: It works well with many classical machine learning algorithms (e.g., Naive Bayes, Logistic Regression).
- **Flexibility**: BoW can be applied to any text data without needing domain knowledge.

Disadvantages of Bag of Words

- **Loss of Context**: BoW ignores the order of words, which can lead to loss of contextual meaning.
- **High Dimensionality**: The vocabulary can become very large, leading to high-dimensional feature vectors that can be computationally expensive.
- **Sparse Representation**: Most documents will contain only a small subset of the vocabulary, resulting in sparse vectors (lots of zeros).

Applications of Bag of Words

- **Text Classification**: BoW is commonly used for tasks such as spam detection, sentiment analysis, and topic classification.
- **Information Retrieval**: Search engines can use BoW representations to find and rank documents based on user queries.
- **Natural Language Processing**: Used as a baseline model for various NLP tasks.

Code:

```
# Importing required libraries

from sklearn.feature_extraction.text import CountVectorizer

# Sample text data

documents = [

  "I love programming.",

  "Programming is fun.",

  "I love coding.",

  "Coding is great." ]

# Creating the CountVectorizer
```

```
vectorizer = CountVectorizer()

# Fitting and transforming the documents

X = vectorizer.fit_transform(documents)

# Converting the result to an array

X_array = X.toarray()

# Getting the feature names (vocabulary)

feature_names = vectorizer.get_feature_names_out()

# Displaying the results

print("Vocabulary:", feature_names)

print("Bag of Words representation:\n", X_array)
```

Output:

```
Vocabulary: ['coding' 'fun' 'great' 'i' 'is' 'love' 'programming']

Bag of Words representation:

 [[0 0 0 1 0 1 1]

 [0 1 0 0 1 0 1]

 [1 0 0 1 0 1 0]

 [1 0 1 0 1 0 0]]
```

2. TF- IDF:

TF-IDF (Term Frequency-Inverse Document Frequency) is a widely used statistical measure in natural language processing (NLP) and information retrieval that evaluates the importance of a word in a document relative to a collection of documents (corpus). It helps in transforming textual data into a format that can be used for various machine learning tasks, such as text classification and clustering.

Components of TF-IDF

TF-IDF consists of two main components:

1. **Term Frequency (TF)**:

   - This measures how frequently a term appears in a document. It is often calculated using the following formula:

TF(t,d)= Number of times term t appears in document d/ Total number of terms in document d

   - The idea is that the more a term appears in a document, the more relevant it is to that document.

2. **Inverse Document Frequency (IDF)**:

   - This measures how important a term is across the entire corpus. It gives higher weights to rare terms and lower weights to common terms. It is calculated using the formula:

IDF(t,D)=log(Total number of documents D/Number of documents containing term t)

   - If a term appears in many documents, its IDF value will be low, indicating it’s less informative.

Combined TF-IDF Score

The overall TF-IDF score for a term “t” in document “d” is calculated by multiplying its TF and IDF scores:

TF-IDF(t,d)=TF(t,d)×IDF(t,D)

This combined score helps identify the terms that are not only frequent in a particular document but also rare across the entire corpus, making them more significant for that document.

Example of TF-IDF Calculation

Consider the following documents:

- **Document 1**: "I love programming."

- **Document 2**: "I love coding."

- **Document 3**: "Programming is fun."

Step 1: Calculate Term Frequencies (TF)

Assuming we are interested in the term "programming":

- **TF for Document 1**:

  - Appears 1 time out of 3 total words: TF=1/3

- **TF for Document 2**:

  - Does not appear: TF=0

- **TF for Document 3**:

  - Appears 1 time out of 3 total words: TF=1/3

Step 2: Calculate Inverse Document Frequencies (IDF)

- Total documents (D) = 3

- "Programming" appears in 2 documents (Doc 1 and Doc 3).

IDF("programming")=log(3/2) ≈ 0.1761

Step 3: Calculate TF-IDF Scores

- **Document 1**:

TF-IDF("programming",Doc 1)=1/3×0.1761≈0.0587

- **Document 2**:

TF-IDF("programming",Doc 2)=0×0.1761=0

- **Document 3**:

TF-IDF("programming",Doc 3)=1/3×0.1761≈0.0587

Advantages of TF-IDF

- **Focus on Relevant Terms**: It emphasizes important words in documents while downweighting common words.

- **Simple and Effective**: Easy to implement and works well for many NLP tasks.

- **Handles Sparse Data**: Efficient for representing large text corpora with a high number of unique terms.

Disadvantages of TF-IDF

- **Ignores Word Order**: Similar to Bag of Words, TF-IDF disregards the order of words, which can lead to loss of context.

- **Limited Semantic Understanding**: TF-IDF does not capture the meanings of words or their relationships; it treats each term independently.

- **Sensitive to Corpus Changes**: Changes in the document corpus can significantly affect TF-IDF scores.

Applications of TF-IDF

- **Document Classification**: TF-IDF features are commonly used in classification algorithms to categorize text documents.

- **Search Engines**: Enhances search results by ranking documents based on the relevance of query terms.

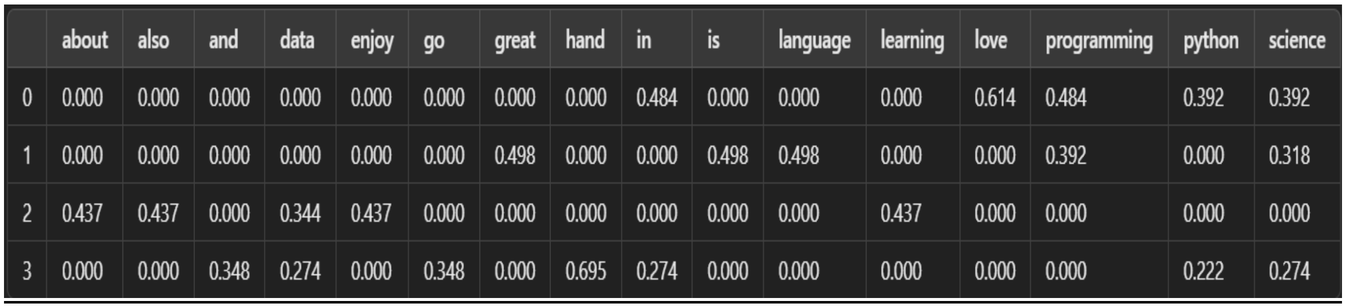
- **Text Clustering**: Used in clustering algorithms to group similar documents together.

- **Recommender Systems**: Can be applied to recommend articles or products based on content similarity.

<u>**Code:**</u>

```
import pandas as pd

from sklearn.feature_extraction.text import TfidfVectorizer

# Sample documents

documents = [

  "I love programming in Python.",

  "Python is a great programming language.",

  "I also enjoy learning about data science.",

  "Data science and Python go hand in hand."

]

# Initialize the TF-IDF Vectorizer

tfidf_vectorizer = TfidfVectorizer()

# Fit and transform the documents

tfidf_matrix = tfidf_vectorizer.fit_transform(documents)

# Convert the TF-IDF matrix to a DataFrame for better visualization

tfidf_df = pd.DataFrame(tfidf_matrix.toarray(),
columns=tfidf_vectorizer.get_feature_names_out())

# Display the TF-IDF scores

print(tfidf_df)
```

Output:



	about	also	and	data	enjoy	go	great	hand	in	is	language	learning	love	programming	python	science
0	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.484	0.000	0.000	0.000	0.614	0.484	0.392	0.392
1	0.000	0.000	0.000	0.000	0.000	0.000	0.498	0.000	0.000	0.498	0.498	0.000	0.000	0.392	0.000	0.318
2	0.437	0.437	0.000	0.344	0.437	0.000	0.000	0.000	0.000	0.000	0.000	0.437	0.000	0.000	0.000	0.000
3	0.000	0.000	0.348	0.274	0.000	0.348	0.000	0.695	0.274	0.000	0.000	0.000	0.000	0.000	0.222	0.274

3. Word Embedding (not in mid term)
-Sentimental Analysis
-Movie on Product review analysis.

Word Embedding is a technique in natural language processing (NLP) that represents words as vectors in a continuous vector space. These embeddings capture semantic relationships between words, meaning words with similar meanings have similar vector representations. This helps models understand context, relationships, and meanings in textual data.

Applications:

1. **Sentiment Analysis:**
   - Word embeddings can be used in sentiment analysis, where the objective is to classify a text based on the sentiment (positive, negative, or neutral). For instance, embeddings can represent words like "happy" or "good" closer to each other in the vector space, allowing models to better capture the sentiment of a sentence or document.
   - A common workflow includes preprocessing the text, applying word embedding techniques like Word2Vec, GloVe, or BERT, and then training a classifier (like logistic regression, neural networks, etc.) on the embedded vectors to predict sentiment.
2. **Movie or Product Review Analysis:**
   - In movie or product review analysis, word embeddings can convert review texts into dense vector representations, which then help machine learning models predict sentiment or extract valuable insights (e.g., feature-based analysis of products).

- For example, reviews like "This product is amazing" would have embeddings reflecting positive sentiment, while "This product is awful" would reflect negative sentiment.
- Embeddings also enable clustering reviews based on similarity, identifying common opinions about movies or products, and performing recommendations based on semantic similarity.

Steps Involved in Using Word Embeddings for Sentiment Analysis or Review Analysis:

1. **Text Preprocessing**:
   - Tokenization, removing stop words, lowercasing, etc.
2. **Embedding Generation**:
   - Apply Word2Vec, GloVe, or contextual embeddings like BERT or GPT embeddings to convert words or documents into vectors.
3. **Training a Model**:
   - Feed the word embeddings to classifiers such as SVM, logistic regression, or deep learning models to classify the sentiment of the review.
4. **Evaluation**:
   - Validate the model performance using metrics such as accuracy, precision, recall, and F1 score.

Word embedding with example, concise similarity(word 2 vec)