# **Natural language Processing (NLP)**

## **List all NLP Steps. Create a detail list.**

Natural Language Processing (NLP) involves a series of steps to understand, interpret, and generate human language. Below is a detailed list of the typical NLP steps:

**Text Collection:**

* Gather the text data from various sources such as books, articles, websites, social media, or any other relevant documents.

**Text Cleaning:**

* Remove unnecessary characters, symbols, and formatting.
* Handle special cases like URL links, email addresses, and numbers.
* Convert text to lowercase for uniformity.
* Correct spelling errors if necessary.

**Tokenization:**

* Break the text into smaller units called tokens, which can be words, phrases, or even characters.
* This step helps in representing the text in a structured and analyzable format.

**Stopword Removal:**

* Eliminate common words (stopwords) that don't carry significant meaning, such as "the," "and," "is," etc.

**Stemming and Lemmatization:**

* Reduce words to their base or root form to simplify analysis.
* Stemming: Cutting off prefixes or suffixes to get to the root.
* Lemmatization: Mapping words to their base form using language dictionaries.

**Part-of-Speech (POS) Tagging:**

* Assign grammatical labels (such as noun, verb, adjective) to each token in the text.

**Named Entity Recognition (NER):**

* Identify and classify entities in the text, such as names of people, organizations, locations, dates, and more.

**Syntax and Dependency Parsing:**

* Analyze the grammatical structure of sentences by determining the relationships between words, including subject, object, and modifiers.

**Text Representation:**

* Convert the tokenized and processed text into a numerical format suitable for machine learning models.
* Techniques include Bag-of-Words, TF-IDF (Term Frequency-Inverse Document Frequency), and word embeddings like Word2Vec or GloVe.

**Feature Engineering:**

* Extract relevant features from the text, which could include sentiment scores, named entity counts, or other custom features.

**Model Building:**

* Select an appropriate NLP model based on the task at hand, such as classification, sentiment analysis, or language generation.
* Common models include recurrent neural networks (RNNs), long short-term memory networks (LSTMs), and transformer models like BERT.

**Training:**

* Train the selected model using a labeled dataset, allowing it to learn patterns and relationships within the data.

**Evaluation:**

* Assess the performance of the trained model using metrics like accuracy, precision, recall, and F1 score.

**Fine-Tuning:**

* Adjust model parameters or features based on the evaluation results to improve performance.

**Deployment:**

* Integrate the trained model into a production environment for real-world applications.

**Monitoring and Maintenance:**

* Continuously monitor the model's performance and update it as needed to adapt to changes in the data or requirements.

These steps represent a general framework for NLP tasks, but the specific processes may vary depending on the task and the complexity of the language involved.

**------------------------------------------------------------------------------------------------------------------------------------------**

## **Diff b/w Stemming and Lemmatization:**

Stemming and lemmatization are both techniques used in natural language processing (NLP) to reduce words to their base or root form. However, they differ in their approaches and the results they produce.

**Stemming:**

* Stemming involves removing prefixes or suffixes from words to obtain their root form, which may not necessarily be a valid word.
* It is a heuristic and rule-based method that aims to truncate words to their common base, often by chopping off the ends of words.
* For example, applying stemming to the word "running" might result in "run."

**Example:**

* Original: "running"
* Stemmed: "run"

**Libraries and Algorithms:**

Common stemming algorithms include Porter Stemmer and Snowball Stemmer.

**Lemmatization:**

* Lemmatization, on the other hand, involves reducing words to their base or root form using a vocabulary and morphological analysis, ensuring that the resulting word is valid.
* It uses language dictionaries and considers the context of the word in a sentence to provide a more accurate and meaningful base form.
* For example, lemmatizing the word "better" might result in "good."

**Example:**

* Original: "better"
* Lemmatized: "good"

**Libraries and Algorithms:**

* Common lemmatization algorithms include WordNet Lemmatizer and spaCy's lemmatization.

**Key Differences:**

**Output Validity:**

* Stemming may result in words that are not valid, as it uses a set of rules to truncate words.
* Lemmatization ensures that the resulting word is valid as it relies on language dictionaries.

**Context Consideration:**

* Stemming doesn't consider the context of the word in a sentence and may produce stems that don't convey the actual meaning.
* Lemmatization considers the context, providing a more meaningful base form that retains the intended sense of the word.

**Algorithm Complexity:**

* Stemming algorithms are generally simpler and rule-based.
* Lemmatization algorithms often involve more complex linguistic rules and analysis.

**Use Cases:**

* Stemming is often used in information retrieval and search engines where speed and simplicity are crucial.
* Lemmatization is preferred in applications where accuracy and a deeper understanding of the language are essential, such as in natural language understanding tasks.

In summary, stemming and lemmatization serve the common purpose of reducing words to their base forms, but they differ in their approaches, the validity of the results, and the consideration of linguistic context. The choice between stemming and lemmatization depends on the specific requirements of the NLP task at hand.

**------------------------------------------------------------------------------------------------------------------------------------------**

## **What is Part-of-Speech (POS) Tagging Explain with Example?**

Part-of-Speech (POS) tagging is a natural language processing (NLP) task that involves assigning grammatical categories (tags) to each word in a sentence based on its syntactic role and relationship to other words. The goal is to understand the grammatical structure of a sentence, identifying whether a word is a noun, verb, adjective, adverb, etc. POS tagging is crucial for various NLP applications, such as information extraction, machine translation, and sentiment analysis.

Here's an explanation of POS tagging with an example:

**Example Sentence**: "ChatGPT is a powerful language model developed by OpenAI."

**POS Tagged Sentence:**

|  |  |
| --- | --- |
| **Word** | **POS Tag** |
| **ChatGPT** | NNP (Proper Noun, Singular) |
| **is** | VBZ (Verb, 3rd person singular present) |
| **a** | DT (Determiner) |
| **powerful** | JJ (Adjective) |
| **language** | NN (Noun, singular or mass) |
| **Model** | NN (Noun, singular or mass) |
| **Developed** | VBN (Verb, past participle) |
| **By** | IN (Preposition or subordinating conjunction) |
| **OpenAI** | NNP (Proper Noun, Singular) |
|  | . (Punctuation - period) |

In this example, each word in the sentence is associated with a POS tag. Here are some common POS tags used in the example:

* **NNP (Proper Noun, Singular):** "ChatGPT" and "OpenAI" are proper nouns referring to specific entities.
* **VBZ (Verb, 3rd person singular present):** "is" is a verb in the third person singular present tense.
* **DT (Determiner):** "a" is a determiner indicating a singular noun follows.
* **JJ (Adjective):** "powerful" is an adjective describing the noun "language."
* **NN (Noun, singular or mass):** "language" and "model" are both nouns.
* **VBN (Verb, past participle):** "developed" is a verb in its past participle form.
* **IN (Preposition or subordinating conjunction):** "by" is a preposition indicating the agent performing the action.
* **. (Punctuation - period):** The period at the end of the sentence is assigned a punctuation tag.

These POS tags provide information about the syntactic role of each word, enabling downstream NLP tasks to better understand the structure and meaning of the text. POS tagging is typically performed using pre-trained models or rule-based algorithms that leverage linguistic patterns and context to assign appropriate tags to each word in a given sentence.

**------------------------------------------------------------------------------------------------------------------------------------------**

## **Represent the meaning of a word** / **What is WordNet?:**

**WordNet** is a **lexical database** of the **English language** that organizes words into sets of **synonyms** called "**synsets**" and describes **relationships** between them. It is a lexical resource that provides a structured and semantically linked collection of words, offering a rich network of connections among words and their meanings. WordNet was developed at Princeton University and has been widely used in natural language processing (NLP), computational linguistics, and cognitive science.

**Key features of WordNet include:**

1. **Synsets:**
   * The core unit in WordNet is the synset, which is a group of synonymous words that represent a distinct concept or meaning.
   * Each synset is associated with a unique identifier and includes a set of words that share similar meanings.
2. **Hyponymy and Hypernymy:**
   * WordNet captures hierarchical relationships between synsets through hyponymy and hypernymy.
   * A hyponym is a more specific term that represents a subclass of a more general term, known as the hypernym. For example, "rose" is a hyponym of "flower," where "flower" is the hypernym.
3. **Meronymy and Holonymy:**
   * WordNet also includes meronymic and holonymic relationships, indicating part-whole associations.
   * A meronym is a term that represents a part of a whole, while a holonym represents the whole. For instance, "petal" is a meronym of "flower."
4. **Antonymy:**
   * WordNet includes information about antonyms, representing words with opposite meanings. For example, "hot" and "cold" are antonyms.
5. **Verb Hierarchies:**
   * WordNet extends beyond nouns and includes hierarchies for verbs and adjectives, allowing the representation of relationships between different word classes.
6. **Word Senses:**
   * Each word in WordNet may have multiple senses, and each sense is associated with a specific synset.
   * This sense disambiguation is beneficial in tasks where understanding the context and choosing the correct meaning of a word is crucial.

**WordNet has been widely used in various NLP applications, including:**

* Sense Disambiguation: Determining the correct sense of a word in context.
* Information Retrieval: Improving search results by considering synonymy and related terms.
* Machine Translation: Enhancing translation quality by capturing semantic relationships.
* Natural Language Understanding: Providing a structured representation of word meanings for computational analysis.

**------------------------------------------------------------------------------------------------------------------------------------------**

## **Problems with resources like WordNet:**

While WordNet is comprehensive and valuable, it does have limitations, such as not covering all words, lacking information about word frequencies, and not capturing dynamic changes in language over time. Despite these limitations, WordNet remains a foundational resource in the field of natural language processing.

**------------------------------------------------------------------------------------------------------------------------------------------**

## **Give an example of Word Senses:**

Certainly! Let's take the word "bank" as an example, which has multiple senses. In WordNet, each sense is associated with a distinct meaning or concept. Here are a few word senses for "bank" in WordNet:

1. **Bank (Financial Institution):**
   * **Synset**: bank.n.01
   * **Definition**: A financial institution that accepts deposits from the public, creates credit, and provides loans.
   * **Example**: "I have an account at the local bank."
2. **Bank (Sloping Land by a Water Body):**
   * **Synset**: bank.n.02
   * **Definition**: The land alongside or sloping down to a river or lake.
   * **Example**: "The children played by the riverbank."
3. **Bank (Mound or Pile):**
   * **Synset**: bank.n.03
   * **Definition**: A long, elevated mound or pile.
   * **Example**: "The snow formed a bank along the side of the road."
4. **Bank (Row or Line):**
   * **Synset**: bank.n.04
   * **Definition**: A row or line, especially of trees or plants.
   * **Example**: "A bank of flowers adorned the garden."
5. **Bank (Turn or Tilt to One Side):**
   * **Synset**: bank.v.01
   * **Definition**: To incline or tilt to one side.
   * **Example**: "The airplane began to bank as it turned."
6. **Bank (Heap or Store):**
   * **Synset**: bank.v.02
   * **Definition**: To form into a bank or heap.
   * **Example**: "The snowplow will bank the snow on the roadside."

In this example, "bank" is associated with different senses, ranging from a financial institution to a sloping land by a water body or a mound of snow. Word senses allow us to disambiguate the meaning of a word based on context, and they are essential for tasks such as natural language understanding and machine translation.

**------------------------------------------------------------------------------------------------------------------------------------------**

## **Token in NLP:**

In natural language processing (NLP), a token is a unit of text that has been extracted from a larger corpus. Tokens can be as short as a single character or as long as an entire word. The process of breaking down text into tokens is called tokenization, and it serves as a fundamental step in many NLP tasks, such as language modeling, sentiment analysis, and machine translation.

## **Segmentation in NLP:**

Segmentation in NLP refers to the process of dividing a continuous sequence of text into smaller, meaningful units, often referred to as segments or chunks. This can involve breaking down text into sentences, phrases, words, or even sub-word units, depending on the specific context and requirements of the task.

Sentence segmentation involves splitting a paragraph into individual sentences, while word segmentation involves breaking down a sentence into its constituent words. Segmentation is a crucial step in many NLP tasks, as it helps in analyzing and understanding the structure of the text, making it easier to extract meaningful information for further processing.

**------------------------------------------------------------------------------------------------------------------------------------------**

## **Discrete vs continuous**

**Discrete:** The number of students in a class, the number of cars in a parking lot, or the number of books on a shelf.

**Continuous:** Examples include height, weight, temperature, and time

**------------------------------------------------------------------------------------------------------------------------------------------**

## **Representing words as discrete symbols:**

In traditional NLP, we regard words as **discrete symbols**:

* hotel, conference, motel – a localist representation

Such symbols for words can be represented by **one-hot vectors (**Means one 1, the rest 0s**)**:

**motel** = [0 0 0 0 0 0 0 0 0 0 1 0 0 0 0]

**hotel** = [0 0 0 0 0 0 0 1 0 0 0 0 0 0 0]

Vector dimension = number of words in vocabulary (e.g., 500,000+)

**------------------------------------------------------------------------------------------------------------------------------------------**

## **Problem with words as discrete symbols:**

In web search, if a user searches for “Seattle motel”, we would like to match  
documents containing “Seattle hotel”  
But:  
**motel** **= [0 0 0 0 0 0 0 0 0 0 1 0 0 0 0]**  
**hotel** **= [0 0 0 0 0 0 0 1 0 0 0 0 0 0 0]**  
These two vectors are orthogonal  
There is no natural notion of similarity for one-hot vectors!

The problem you're describing is related to the representation of words as discrete symbols, typically in the context of natural language processing (NLP) tasks like web search or document retrieval. In traditional NLP models, words are often represented using one-hot encoding, where each word is represented as a vector of zeros with a single one at the index corresponding to the word's position in the vocabulary.

In the example you provided, "motel" and "hotel" are represented as one-hot vectors:

**motel = [0 0 0 0 0 0 0 0 0 0 1 0 0 0 0]**

**hotel = [0 0 0 0 0 0 0 1 0 0 0 0 0 0 0]**

Since each word is represented by a unique vector, there is no inherent notion of similarity between these vectors. Mathematically, they are orthogonal to each other, meaning they have a dot product of zero. This makes it difficult for traditional models to understand that "motel" and "hotel" are related concepts and should be treated as similar in certain contexts, such as a web search query.

One approach to address this issue is to learn to encode similarity directly into the word vectors themselves. This can be achieved using techniques like word embeddings, which map words to dense, continuous vector representations in a lower-dimensional space. Word embeddings are learned from large text corpora using methods like Word2Vec, GloVe, or fastText.

In word embedding models, similar words tend to have similar vector representations, capturing semantic relationships between words. For example, in a well-trained word embedding model, the vectors for "motel" and "hotel" are likely to be close to each other in the vector space, indicating their semantic similarity.

By using word embeddings instead of one-hot encoding, NLP models can better capture the semantic relationships between words, leading to improved performance on tasks like web search or document retrieval where understanding word similarity is important.

**------------------------------------------------------------------------------------------------------------------------------------------**

# **Representing words by their context:**

This approach to representing words, known as **distributional semantics**, is based on the idea that a word's meaning is closely tied to the context in which it appears in text. In other words, **words that frequently appear together** in similar contexts are likely to have similar meanings. This concept is often summarized by the quote, **"You shall know a word by the company it keeps,"**

Instead of representing words as discrete symbols or using one-hot encoding, distributional semantics aims to capture the meaning of a word by analyzing the distribution of its surrounding words in a corpus of text. When a word appears in a text, its context is defined as the set of words that appear nearby within a fixed-size window.

For example, consider the following sentences:

1. "...government debt problems turning into **banking** crises as happened in 2009…"
2. "...saying that Europe needs unified **banking** regulation to replace the hodgepodge…"
3. "...India has just given its **banking** system a shot in the arm…"

In these sentences, the word "**banking**" appears in different contexts, surrounded by different sets of words. By analyzing the various contexts in which "**banking**" appears across a corpus of text, we can build up a representation of the word "**banking**" that captures its meaning based on the words that tend to co-occur with it.

This approach allows us to create dense, continuous vector representations of words, known as word embeddings, where words with similar meanings are represented by similar vectors in a lower-dimensional space. Word embeddings are typically learned from large text corpora using methods like Word2Vec, GloVe, or fastText.

By representing words based on their context, distributional semantics provides a more nuanced and flexible way to capture the semantic relationships between words compared to traditional methods like one-hot encoding. This approach has been highly successful in various natural language processing tasks, including word similarity, document classification, and machine translation.

**------------------------------------------------------------------------------------------------------------------------------------------**

## **Word vectors**

In the context of natural language processing (NLP), word vectors, also known as **word embeddings** or **neural word representations**, are dense, continuous vector representations of words in a lower-dimensional space. These vectors are designed in such a way that words with similar meanings or usage tend to have similar vector representations. This is achieved by training models on large corpora of text data and learning the relationships between words based on their context.

Here's how the process typically works:

1. **Contextual Representation**: Words are represented based on the context in which they appear in a corpus of text. This context could include the words that appear nearby within a fixed-size window or other linguistic features.
2. **Learning Word Embeddings**: Models are trained to predict the likelihood of a word occurring given its context, or vice versa, using techniques like Word2Vec, GloVe (Global Vectors for Word Representation), or fastText. During training, the model adjusts the word vectors to minimize the prediction error, effectively learning to represent words in a way that captures their semantic relationships.
3. **Distributed Representation**: Each word is represented by a dense vector with continuous values, typically consisting of hundreds of dimensions. This is in contrast to the sparse, one-hot encoding representation where each word is represented by a vector with only one non-zero element.
4. **Similarity Measurement**: The similarity between word vectors is often measured using mathematical operations such as the dot product, cosine similarity, or Euclidean distance. Words with similar meanings or usage tend to have word vectors that are closer to each other in the vector space, indicating their semantic similarity.
5. **Applications**: Word embeddings have proven to be highly useful in various NLP tasks, including word similarity, language modeling, sentiment analysis, document classification, machine translation, and more. By capturing semantic relationships between words, word embeddings enable NLP models to better understand and process natural language data.

Overall, word vectors provide a more nuanced and flexible representation of words compared to traditional methods like one-hot encoding, allowing NLP models to capture the rich semantic information present in natural language text.

**------------------------------------------------------------------------------------------------------------------------------------------**

## **Explain Word2Vec with Example:**

Word2Vec is a popular technique for learning **word embeddings** from large corpora of text data. It was introduced by Tomas Mikolov et al. at Google in 2013. Word2Vec essentially learns to map words from a high-dimensional space (the vocabulary space) to a lower-dimensional space (the embedding space) in a way that preserves semantic relationships between words. The key intuition behind Word2Vec is that words that appear in similar contexts tend to have similar meanings.

There are two main architectures for Word2Vec: **Continuous Bag of Words (CBOW) and Skip-gram**. Let's focus on the Skip-gram model, which is more widely used and generally performs better.

Here's how the Skip-gram model works with an example:

Suppose we have the following sentence:

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

1. **Tokenization**: First, we tokenize the sentence into words and build our training data. We define a context window size, which determines the number of words to consider as context for each target word.
2. **Training Data**: For each word in the sentence, we define a center word-context word pair. The center word is the target word we want to learn the embedding for, and the context words are the words that appear within a fixed-size window around the center word.

For example, if we choose a context window size of 2:

* + For the word "**quick**", the context words are **"the", "brown", and "fox".**
  + For the word **"brown",** the context words are **"quick", "the", "fox", and "jumps".**

1. **Training Objective**: Given a center word, the objective is to predict the surrounding context words, and vice versa. This is done by **maximizing the probability** of observing context words given the center word (or vice versa), as explained in the previous response.
2. **Learning Word Embeddings**: During training, the Skip-gram model adjusts the word vectors (embeddings) to maximize the probability of observing context words given the center word (or vice versa). This is typically done using stochastic gradient descent or other optimization algorithms.
3. **Embedding Space**: After training, each word in the vocabulary is represented by a dense vector in the embedding space. Words with similar meanings or usage tend to have similar vector representations, capturing semantic relationships between words.

For instance, after training, the word embeddings might represent the word "quick" as a vector **[0.2, -0.4, 0.7, ...]**, and the word "brown" as a vector **[0.1, -0.3, 0.6, ...]**.

These learned word embeddings can then be used as features in various NLP tasks such as sentiment analysis, machine translation, and document classification, providing a rich representation of the underlying semantics of the text data.

**------------------------------------------------------------------------------------------------------------------------------------------**

## **There are two main architectures for training Word2Vec models:**

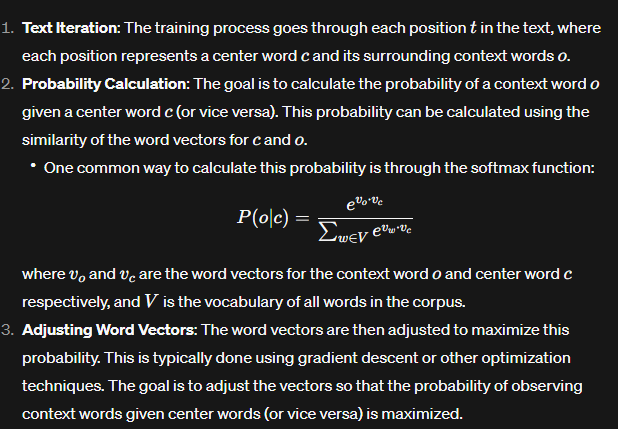
1. Continuous Bag of Words (CBOW): This architecture predicts the **target word** based on its **surrounding context words**. It takes a window of **neighboring words** around the target word and tries to predict the target word given these **context words.**
2. Skip-gram: In this architecture, the model predicts the **context words** (or surrounding words) given the **target word**. It aims to **maximize the probability** of **predicting context words** within a certain window given the **target word.**

Both CBOW and Skip-gram models are trained using a shallow neural network with a single hidden layer. The weights learned by this neural network form the word embeddings.

**------------------------------------------------------------------------------------------------------------------------------------------**

## **Word2Vec Overview:**

We have a large corpus (“body”) of text: a long list of words  
• Every word in a fixed vocabulary is represented by a vector  
• Go through each position **t** in the text, which has a center  
word **c** and context (“outside”) words **o**  
• Use the **similarity** of the **word vectors** for **c** and **o** to calculate  
the **probability** of **o given c** (or vice versa)  
• Keep adjusting the **word vectors** to **maximize this probability**



**------------------------------------------------------------------------------------------------------------------------------------------**

## **Explain GloVe?**

GloVe, which stands for **Global Vectors for Word Representation**, is a popular algorithm for learning **word embeddings**, which are **dense vector representations** of words in a **high-dimensional space**. These embeddings capture **semantic relationships** between words, allowing algorithms to better understand the meanings and context of words in natural language processing (NLP) tasks.

Here's a breakdown of how GloVe works:

1. **Co-occurrence Matrix**: GloVe starts by constructing a large **co-occurrence matrix** from a corpus of text. Each element in this matrix represents how often a particular word appears in the context of another word within a certain window size. For example, if the word **"apple"** appears near **"orange**" frequently in the text corpus, the corresponding entry in the co-occurrence matrix for "apple" and "orange" would have a higher value.
2. **Objective Function**: The core idea behind GloVe is to learn word embeddings by optimizing an objective function that captures the relationship between word co-occurrences and their vector representations. The objective is to learn vectors such that their dot product equals the logarithm of the observed co-occurrence probability between the corresponding words.
3. **Training**: GloVe uses gradient descent or similar optimization techniques to minimize the difference between the dot product of word vectors and the logarithm of the observed co-occurrence probabilities. This process adjusts the word vectors iteratively until they capture meaningful relationships between words in the given corpus.
4. **Word Embeddings**: After training, GloVe produces word embeddings, which are dense vector representations of words in the learned vector space. These embeddings capture semantic relationships between words, such as similarity and analogy. For instance, in the vector space learned by GloVe, words with similar meanings or contexts will have similar vector representations, and relationships like "king - man + woman ≈ queen" can be approximated.

GloVe has gained popularity in the NLP community due to its ability to capture global statistical information from large text corpora efficiently, resulting in high-quality word embeddings that can be used in various downstream NLP tasks such as sentiment analysis, machine translation, and named entity recognition.

**------------------------------------------------------------------------------------------------------------------------------------------**

## **Difference between token classification and NER**

Token classification and Named Entity Recognition (NER) are related concepts in natural language processing, but they have some key differences:

1. **Task Definition:**
   * **Token Classification:** In token classification, the goal is to classify individual tokens into predefined categories or classes. Each token is treated independently, and the focus is on labeling each token with a specific class.
   * **NER (Named Entity Recognition):** NER, on the other hand, is a specific application of token classification. In NER, the task is to identify and classify entities (such as names of people, organizations, locations) within a text. The entities often consist of multiple tokens, and the goal is to recognize and classify entire named entities.
2. **Scope:**
   * **Token Classification:** It can be a broader task where tokens are classified into various categories, which may not necessarily be named entities.
   * **NER:** Focuses specifically on extracting and classifying entities within the text.
3. **Example:**
   * **Token Classification:** Classifying each word in a sentence as "noun," "verb," "adjective," etc.
   * **NER:** Identifying entities like "Microsoft" as an organization, "John" as a person, or "New York" as a location in a given text.
4. **Output Format:**
   * **Token Classification:** Outputs a label for each token in the input sequence.
   * **NER:** Outputs identified entities along with their types.

In summary, token classification is a more general concept where individual tokens are classified into categories, while NER is a specific application of token classification focusing on recognizing and classifying named entities within a text.