Please note : This is just only a reference material . Please read the prescribed books in the syllabus and any other resources for knowledge enhancement in this subject.(NLP)

Sentiment Analysis

Sentiment analysis is a natural language processing (NLP) technique that automatically determines the emotional tone of a piece of text to determine if it's positive, negative, or neutral. It's also known as opinion mining or emotion artificial intelligence.

For e.g.

Positive sentiment : I love pizza.

Negative sentiment : Pizza was very bad.

Neutral sentiment : I ordered the pizza today.

Sentiment analysis is a subcategory of NLP, which is a field that gives computers the ability to understand human language. It uses machine learning to analyze text data, which involves algorithms that learn from training data.

Sentiment analysis is a useful tool for businesses to understand their customers and make data-driven decisions. It can help organizations: Understand customer opinions and experience, Improve brand reputation, Understand worker attitudes, Make informed marketing strategies, and Stay up to date on market trends.

Sentiment analysis can be applied to a variety of online sources, including:

- Surveys
- News articles
- Tweets
- Blog posts
- Customer support chat transcripts
- Social media comments
- Reviews

The steps for performing sentiment analysis using natural language processing (NLP) are:

- Data collection: Gather data from multiple sources
- Text preprocessing: Clean and prepare the data for analysis. This includes tokenization, stemming, lemmatization, stop-word removal, and part-of-speech tagging.
- Feature extraction: Extract valuable information from the text data. This helps to encapsulate the most essential features of the text.

- Model training and optimization: Train and optimize the sentiment analysis model.
- Model evaluation: Assess the performance of the model using relevant metrics. This involves generating predictions using the trained model on a test dataset, and then creating a classification report.

Sentiment analysis is a major NLP task that involves extracting the feeling from a given text and determining whether it is **positive, negative, or neutral.**

Difference between Jacard and Cosine Similarity :

- Jaccard similarity is primarily used for sets or binary data. It compares the presence or absence of elements between two sets.
- Cosine similarity is commonly used for vector representations, particularly in text analysis or document comparisons.

- Jaccard similarity is calculated by dividing the size of the intersection of two sets by the size of their union. It ranges from 0 (no common elements) to 1 (identical sets).
- Cosine similarity is calculated by taking the dot product of two vectors and dividing it by the product of their magnitudes. It ranges from -1 (opposite directions) to 1 (identical directions), with 0 indicating orthogonality.

- Jaccard similarity is commonly used in applications involving set comparison, document similarity, collaborative filtering, social network analysis, and data deduplication.
- Cosine similarity is widely used in text analysis, document comparisons, information retrieval, recommendation systems, and dimensionality reduction tasks. It is particularly useful when capturing semantic or contextual similarity in textual data.

Another example of Jacard

Jaccard similarity is a measure of how two sets (of n-grams in your case) are similar.

Let us take example of how to find Jacard similarity between two strings by using char bigram model

For example if you have 2 strings "**abcde**" and "**abdcde**" it works as follow :

```
ngrams (n=2) :  'abcde' & 'abdcde'
   ab bc cd de dc bd
A  1  1  1  1  0  0
B  1  0  1  1  1  1
```

J(A, B) = (A∩B) / (A∪B)

J(A, B) = (3 / 6) = 0.5

There is also the **Jaccard distance** which captures the dissimilarity between two sets, and is calculated by taking one minus the Jaccard coefficient (in this case, `1 - 0.5 = 0.5`)

Jaccard Distance = 1 - Jaccard Coefficient

Language : **Language** is a structured system of communication that consists of grammar and vocabulary. It is the primary means by which humans convey meaning, both in spoken and signed forms, and may also be conveyed through writing.

CFG (Context Free Grammar)

In natural language processing, a context-free grammar (CFG) is a set of production rules used to generate all the possible sentences in a given language. A CFG is a formal grammar in the sense that it consists of a set of terminals, which are the basic units of the language, and a set of non-terminals, which are used to generate the terminals through a set of production rules. CFGs are often used in natural language parsing and generation. They are also used in natural language understanding, where a CFG can be used to analyze the syntactic structure of a sentence.

$G = \langle T, N, S, R \rangle$

- T is set of terminals (lexicon)
- N is set of non-terminals
- S is start symbol (one of the non terminals)
- R is rules/productions of the form $X \rightarrow \gamma$, where X is a nonterminal and γ is a sequence of terminals and nonterminals (may be empty).
- A grammar G generates a language L.

An example context-free grammar

$G = \langle T, N, S, R \rangle$

$T = \{that,\ this,\ a,\ the,\ man,\ book,\ flight,\ meal,\ include,\ read,\ does\}$

$N = \{$S, NP, NOM, VP, Det, Noun, Verb, Aux$\}$

$S = S$

$R = \{$

S → NP VP	Det → *that* | *this* | *a* | *the*
S → Aux NP VP	Noun → *book* | *flight* | *meal* | *man*
S → VP	Verb → *book* | *include* | *read*
NP → Det NOM	Aux → *does*
NOM → Noun	
NOM → Noun NOM	
VP → Verb	
VP → Verb NP	

$\}$

Application of grammar rewrite rules

S → NP VP	Det → *that* | *this* | *a* | *the*
S → Aux NP VP	Noun → *book* | *flight* | *meal* | *man*
S → VP	Verb → *book* | *include* | *read*
NP → Det NOM	Aux → *does*
NOM → Noun	
NOM → Noun NOM	
VP → Verb	
VP → Verb NP	

S → NP VP
→ Det NOM VP
→ *The* NOM VP
→ *The* Noun VP
→ *The man* VP
→ *The man* Verb NP
→ *The man read* NP
→ *The man read* Det NOM
→ *The man read this* NOM
→ *The man read this* Noun
→ *The man read this book*

Parsing

In natural language processing, parsing is the process of analyzing a sentence to determine its grammatical structure, and there are two main approaches to parsing: top-down parsing and bottom-up parsing.

What is Parser ?

A parser is a software component that takes input data (typically text) and builds a data structure – often some kind of parse tree, abstract syntax tree or other hierarchical structure, giving a structural representation of the input while checking for correct syntax. Parser also requires CFG for generating parse tree.

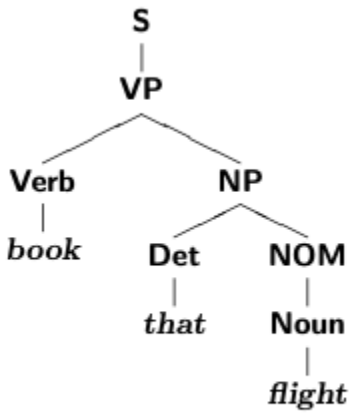
A parser in NLP uses the grammar rules to verify if the input text is valid or not syntactically. The parser helps us to get the meaning of the provided text. As the parser helps us to analyze

the syntax error in the text; so, the parsing process is also known as the syntax analysis or the Syntactic analysis.

Top-down parsing is a parsing technique that starts with the highest level of a grammar's production rules, and then works its way down to the lowest level. It begins with the start symbol of the grammar and applies the production rules recursively to expand it into a parse tree.

Bottom-up parsing is a parsing technique that starts with the sentence's words and works its way up to the highest level of the grammar's production rules. It begins with the input sentence and applies the production rules in reverse, reducing the input sentence to the start symbol of the grammar.

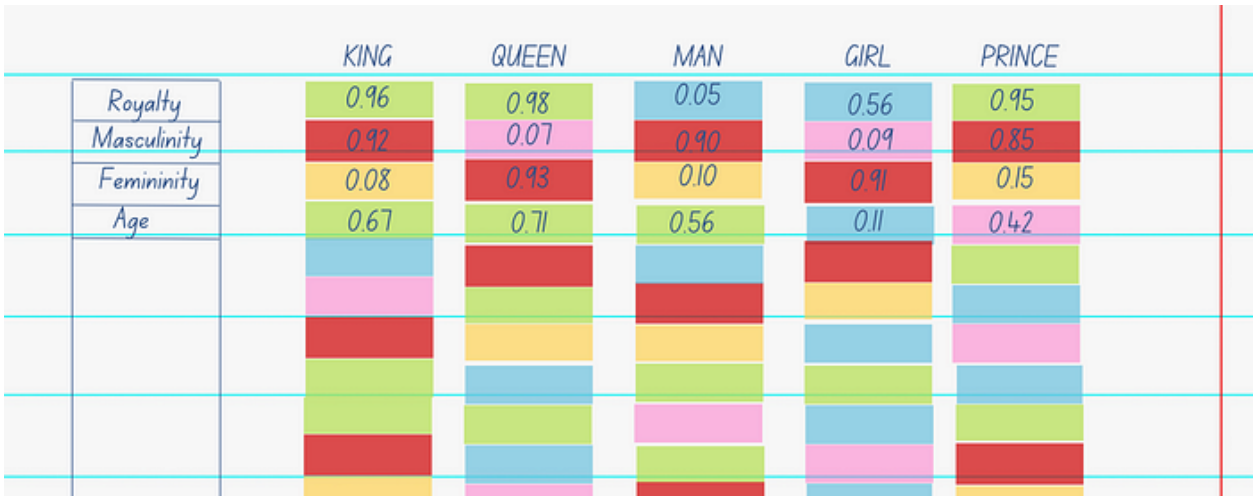
Example of Top Down Parsing (Parse Tree)



Word2Vec

Word2Vec creates a representation of each word present in our vocabulary into a vector. Words used in similar contexts or having semantic relationships are captured effectively through their closeness in the vector space- effectively speaking similar words will have similar word vectors! History. Word2vec was created, patented, and published in 2013 by a team of researchers led by Tomas Mikolov at Google.

Let us consider a classic example: “king”, “queen”, “man”, “girl”, “prince”



Hypothetical features to understand word embeddings

In a hypothetical world, vectors could then define the weight of each criteria (for example royalty, masculinity, femininity, age etc.) for each of the given words in our vocabulary.

What we then observe is:

- As expected, “king”, “queen”, “prince” have similar scores for “royalty” and “girl”, “queen” have similar scores for “femininity”.
- An operation that removes “man” from “king”, would yield in a vector very close to “queen” (“king”- “man” = “queen”)
- Vectors “king” and “prince” have the same characteristics, except for age, telling us how they might possibly be semantically related to each other.

Word2Vec forms word embeddings that work in a similar fashion except for the fact that the criterion we have used for each of the words are not clearly determinable. What matters to us is the semantic and

syntactic relations between words which can still be determined by our model without explicitly having defining features for units of the vector.

Word2Vec has also shown to identify relations like country-capital over larger datasets showing us how powerful word embeddings can be.

Model Architecture

Word2Vec essentially is a shallow 2-layer neural network trained.

- The input contains all the documents/texts in our training set. For the network to process these texts, they are represented in a 1-hot encoding of the words.
- The number of neurons present in the hidden layer is equal to the length of the embedding we want. That is, if we want all

our words to be vectors of length 300, then the hidden layer will contain 300 neurons.

- The output layer contains probabilities for a target word (given an input to the model, what word is expected) given a particular input.
- At the end of the training process, the hidden weights are treated as the word embedding. Intuitively, this can be thought of as each word having a set of n weights (300 considering the example above) “weighing” their different characteristics (an analogy we used earlier).

There are two ways in which we can develop these embeddings:

1. **Continuous Bag-Of-Words (CBOW) :** CBOW predicts the target-word based on its surrounding words. The input layer contains the context words and the output layer contains the

current word. The hidden layer contains the dimensions we want to represent the current word present at the output layer.

For example, consider the sentence, “The cake was chocolate flavoured”.

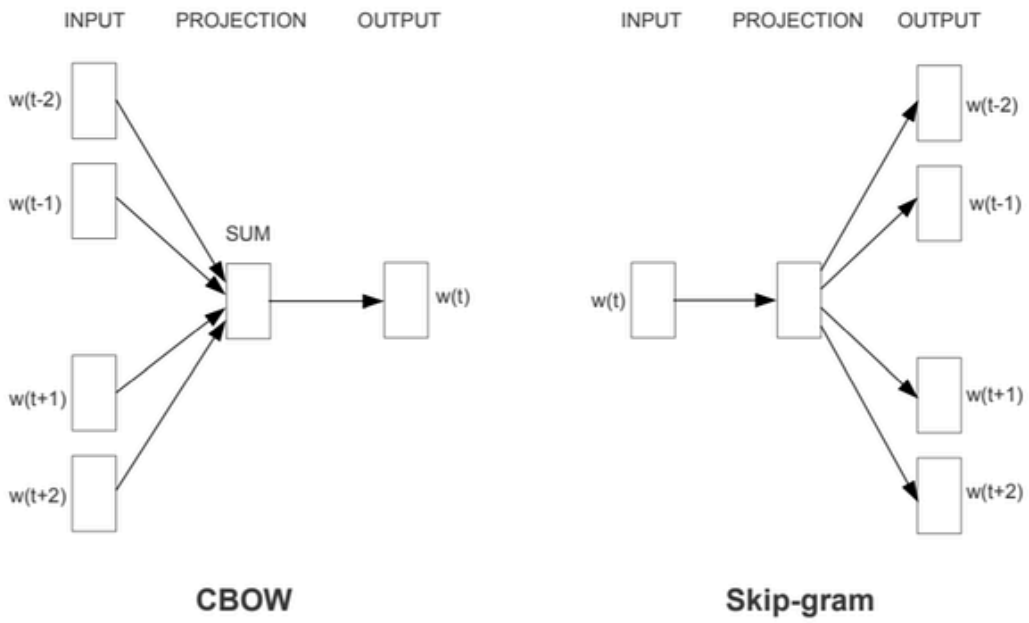
The model will then iterate over this sentence for different target words, such as:

“The _____ was chocolate flavoured” being inputs and “cake” being the target word.

2. Skipgram

Skipgram works in the exact opposite way to CBOW. Here, we take an input word and expect the model to tell us what words it is expected to be surrounded by. The input layer contains the current word and the output layer contains the context words. The hidden layer contains the number of dimensions in which we want to represent current word present at the input layer.

Taking the same example, with “cake” we would expect the model to give us “The”, “was”, “chocolate”, “flavoured” for the given instance.



Difference between Bag of Words and Word2Vec

Bag of words (BoW) and Word2Vec are both natural language processing (NLP) techniques that use embeddings to represent text data, but they differ in how they process text:

Bag of words

A statistical model that counts the number of times each word appears in a document and represents the text as a fixed-length vector. BoW disregards word order and syntax, and only considers whether a word appears in a document, not where.

Word2Vec

A two-layer neural network that processes text by converting words into vectors that represent their meaning, semantic similarity, and relationship to other words.

Word2Vec takes in batches of raw text data and outputs a vector space where each word is assigned a corresponding vector.

Here are some other differences between BoW and Word2Vec:

Implementation
BoW can be implemented as a Python dictionary, while Word2Vec uses a dense neural network.

Limitations
BoW has several limitations, including not understanding word order, context, or meaning, and always representing text in the same way.

Applications
BoW is used in information retrieval, while Word2Vec can help computers learn the context of expressions and keywords from large text collections.

Understanding Linguistic :

Linguistics is the study of language, its structure, and the rules that govern its structure. Linguists, specialists in linguistics, have traditionally analyzed language in terms of several subfields of study. Speech-language pathologists study these subfields of language and are specially trained to assess and treat language and its subfields. These include morphology, syntax, semantics, pragmatics and phonology.

Morphology

Morphology is the study of word structure. It describes how words are formed out of more basic elements of language called morphemes. A morpheme is the smallest unit of a language. Morphemes are considered minimal because if they were subdivided any further, they would become meaningless. Each morpheme is different from the others because each singles a distinct meaning. Morphemes are used to form words. *Base*, *root* or *free* morphemes are words that have meaning, cannot be broken-down into smaller parts, and can have other morphemes added to them. Examples of free morphemes are ocean, establish, book, color, connect, and hinge. These words mean something, can stand by themselves, and cannot be broken down into smaller units. These words can also have other morphemes added to them.

Bound or grammatical morphemes, which cannot convey meaning by themselves, must be joined with free morphemes in order to have meaning. In the following examples, the free morphemes are underlined; the bound morphemes are in capital letters: establishMENT, bookED, colorFUL, DISconnect. Common bound or grammatical morphemes include the following: -ing (the present progressive), -s (the regular plural; e.g., cats), -s (the possessive inflection; e.g., man's), and –ed (the regular past tense; e.g., washed). Morphemes are a means of modifying word structures to change

meaning. The morphology of a given language describes the rules of such modifications.

Syntax :

Syntax and morphology are concerned with two major categories of language structure. Morphology is the study of word structure. **Syntax is the study of sentence structure.** The basic meaning of the word syntax is “to join,” “to put together.” In the study of language, syntax involves the following:

1. The arrangement of words to form meaningful sentences
2. The word order and overall structure of a sentence

A collection of rules that specify the ways and order in which words may be combined to form sentences in a particular language. As they mature in syntactic development, children begin to use compound and complex sentences, which can be defined as follows:

1. Compound sentence: two or more independent clauses joined by a common and a conjunction or by a semicolon. There are no subordinate clauses in a compounded sentence. A clause contains a subject and a predicate. An independent or main clause has a subject and a predicate and can stand alone (e.g., “The policeman held up the sign, and the cars stopped.”)
2. Complex sentence: contains one independent clause and one or more dependent or subordinate clauses. A dependent or subordinate clause has a subject and predicate but cannot stand alone. (e.g., “I will drive my car to Reno if I have enough gas.”)

Syntax rules differ by language. Speakers of a language do not produce structures with random and meaningless word order. If they do, speech and language therapy may be warranted. For example, an English speaker could say, “He said he was going to come but didn’t.” Due to syntactic rules, a speaker could not say, “He’s going to was said he didn’t but come.” Languages have different syntactic structures. In English, the basic syntactic structure is subject + verb + object. This structure, usually called the “kernel sentence”, can also be called the phrase structure or base structure.

Semantic analysis

Semantic analysis is a crucial component of natural language processing (NLP) that concentrates on understanding the meaning, interpretation, and relationships between words, phrases, and sentences

It is a crucial component of Natural Language Processing (NLP) and the inspiration for applications like chatbots, search engines, and text analysis tools using machine learning.

Tools based on semantic analysis can assist businesses in automatically extracting useful information from unstructured data, including emails, support requests, and consumer comments.

Advantages of Semantic Analysis

Semantic analysis offers several advantages across various fields and applications:

Improved Understanding of Text:

It helps understand the true meaning of words, phrases, and sentences, leading to a more accurate interpretation of text.

Enhanced Search and Information Retrieval:

Search engines can provide more relevant results by understanding user queries better, considering the context and meaning rather than just keywords.

Better Natural Language Processing (NLP):

Semantic analysis forms the backbone of many NLP tasks, enabling machines to understand and process language more effectively, leading to improved machine translation, sentiment analysis, etc.

Improved Machine Learning Models:

In AI and machine learning, semantic analysis helps in feature extraction, sentiment analysis, and understanding relationships in data, which enhances the performance of models.

Enhanced User Experience:

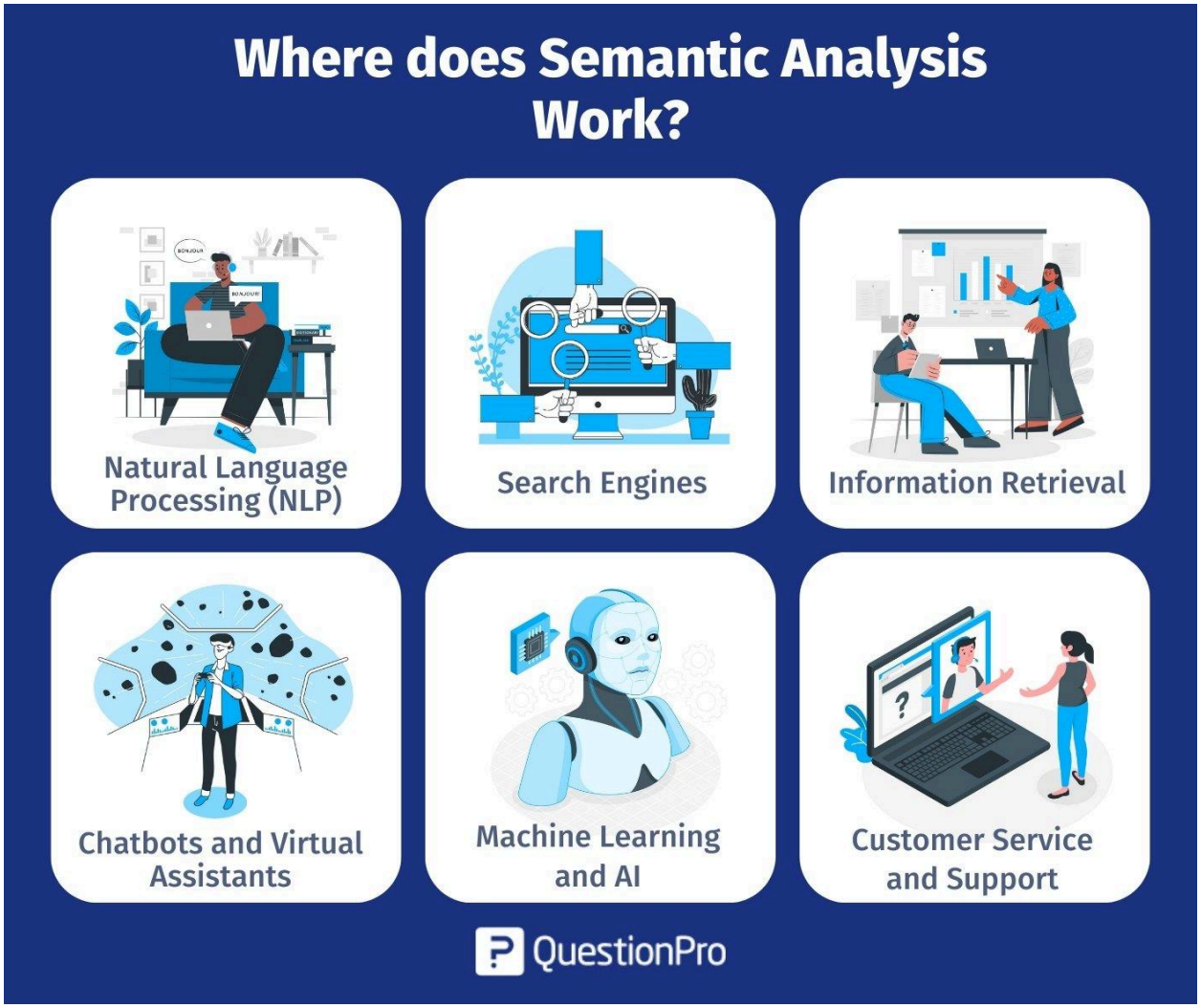
Chatbots, virtual assistants, and recommendation systems benefit from semantic analysis by providing more accurate and context-aware responses, thus significantly improving user satisfaction.

Personalization and Recommendation Systems:

Semantic analysis allows for a deeper understanding of user preferences, enabling personalized recommendations in e-commerce, content curation, and more.

Where does Semantic Analysis Work?

Semantic analysis finds applications in various fields, including:



Natural Language Processing (NLP):

It’s used extensively in NLP tasks like sentiment analysis, document summarization, machine translation, and question answering, thus showcasing its versatility and fundamental role in processing language.

Search Engines:

Semantic analysis aids search engines in comprehending user queries more effectively, consequently retrieving more relevant results by considering the meaning of words, phrases, and context.

Information Retrieval:

In libraries or databases, it helps retrieve documents based on their semantic relevance rather than just keyword matching.

Chatbots and Virtual Assistants:

Semantic analysis enables these systems to comprehend user queries, leading to more accurate responses and better conversational experiences.

Machine Learning and AI:

It recreates a crucial role in enhancing the understanding of data for machine learning models, thereby making them capable of reasoning and understanding context more effectively.

Customer Service and Support:

Semantic analysis aids in analyzing and understanding customer queries, helping to provide more accurate and efficient support.

Pragmatic Analysis

Pragmatics in NLP is the study of contextual meaning. It examines cases where a person's statement has one literal and another more profound meaning. It tells us how different contexts can change the meaning of a sentence. It is a subfield of linguistics that deals with interpreting utterances in communication. Pragmatics considers the intentions of the speaker and writer. Additional information is also considered to form the context.

There are many aspects of language that require knowledge derived from pragmatic analysis. It helps us understand whether the sentence "Give me a glass of water" is an order or a request. We cannot say this unless we have some context.
Thus pragmatics in NLP helps the computer to understand the real meaning of the sentences in certain situations. Pragmatics is also concerned with the roles the speaker and the listener play in creating sense.

For example, if in a school the teacher asks the student coming late in the class as "What time do you call this?!" It does not mean "the teacher is asking her for the time". This is the semantic meaning of the sentence. Here it is referred to as "Why are you so late?" which is the pragmatic meaning of the sentence.

Importance of Pragmatics in NLP

The concept of pragmatics holds excellent importance for Natural Language Processing(NLP). It makes machines understand spoken and written language better. Pragmatics in NLP analyzes the context and helps machines understand how it changes the meaning of utterances. NLP systems have been facing problems in their regard for a long time.

Through a few examples, let's try to understand what would happen if pragmatics didn't exist.

Example 1

'Can you pull the car over?'

- **Actual Meaning**: *Are you capable of pulling(literally pulling) the car*?
- **Pragmatic Meaning**: *This means 'Can you stop the car' .*

Example 2
“Wow, that’s just what I needed.”

- **Actual Meaning:** *This may mean that someone was looking for something, and they found it. The sentence gives a positive sentiment here*.

- **Pragmatic Meaning**: *This could express sarcasm and negative sentiment in a specific situation.*

Example 3

An example of pragmatic meaning is: “*It's hot in here! Can you crack a window?*"

Here we can infer that the speaker wants the window to be opened a little and does not want the window to be physically damaged.

Look at the situation/context to understand the actual meaning of the sentence spoken. If you were told to “**crack the window**” and the room was a little stuffy. The speaker had just said before that they were **feeling hot**. We would know that the speaker wants you to “open the window” and not literally crack it.

Tokenization

Tokenization in **natural language processing (NLP)** is a technique that involves dividing a sentence or phrase into smaller units known as tokens. These tokens can encompass words, dates, punctuation marks, or even fragments of words.

What is Tokenization in NLP?

Tokenization is the process of dividing a text into smaller units known as tokens. **Tokens** are typically words or sub-words in the context of natural language processing. Tokenization is a critical step in many NLP tasks, including text processing, language modelling, and machine translation. The process involves splitting a string, or text into a list of tokens. One can think of tokens as parts like a word is a token in a sentence, and a sentence is a token in a paragraph.

Tokenization involves using a tokenizer to segment unstructured data and natural language text into distinct chunks of information, treating them as different elements. The tokens within a document can be used as vector, transforming an unstructured text document into a numerical data structure suitable for machine learning. This rapid conversion enables the immediate utilization of these tokenized elements by a computer to initiate practical

actions and responses. Alternatively, they may serve as features within a machine learning pipeline, prompting more sophisticated decision-making processes or behaviors.

Types of Tokenization

Tokenization can be classified into several types based on how the text is segmented. Here are some types of tokenization:

Word Tokenization:

Word tokenization divides the text into individual words. Many NLP tasks use this approach, in which words are treated as the basic units of meaning.

Example:

```
Input: "Tokenization is an important NLP task."
```

```
Output: ["Tokenization", "is", "an", "important", "NLP", "task", "."]
```

Sentence Tokenization:

The text is segmented into sentences during sentence tokenization. This is useful for tasks requiring individual sentence analysis or processing.

Example:

```
Input: "Tokenization is an important NLP task. It helps break down text into smaller units."
```

```
Output: ["Tokenization is an important NLP task.", "It helps break down text into smaller units."]
```

Character Tokenization:

This process divides the text into individual characters. This can be useful for modelling character-level language.

Example:

```
Input: "Tokenization"
```

```
Output: ["T", "o", "k", "e", "n", "i", "z", "a", "t", "i", "o", "n"]
```

Need of Tokenization

Tokenization is a crucial step in text processing and natural language processing (NLP) for several reasons.

- **Effective Text Processing:** Tokenization reduces the size of raw text so that it can be handled more easily for processing and analysis.
- **Feature extraction:** Text data can be represented numerically for algorithmic comprehension by using tokens as features in machine learning models.

- **Language Modelling:** Tokenization in NLP facilitates the creation of organized representations of language, which is useful for tasks like text generation and language modelling.
- **Information Retrieval:** Tokenization is essential for indexing and searching in systems that store and retrieve information efficiently based on words or phrases.
- **Text Analysis:** Tokenization is used in many NLP tasks, including sentiment analysis and named entity recognition, to determine the function and context of individual words in a sentence.
- **Vocabulary Management:** By generating a list of distinct tokens that stand in for words in the dataset, tokenization helps manage a corpus's vocabulary.
- **Task-Specific Adaptation:** Tokenization can be customized to meet the needs of particular NLP tasks, meaning that it will work best in applications such as summarization and machine translation.
- **Preprocessing Step:** This essential preprocessing step transforms unprocessed text into a format appropriate for additional statistical and computational analysis.

Stemming

Stemming is a linguistic normalization process in natural language processing and information retrieval. Its primary goal is to reduce words to their base or root form, known as the stem. Stemming helps group words with similar meanings or roots together, even if they have different inflections, prefixes, or suffixes.

The process involves removing common affixes (prefixes, suffixes) from words, resulting in a simplified form that represents the word's core meaning. Stemming is a heuristic process and may only sometimes produce a valid word. Still, it is effective for tasks like information retrieval, where the focus is on matching the essential meaning of words rather than their grammatical correctness.

For example:

- running -> run
- jumps -> jump
- swimming -> swim

Stemming algorithms use various rules and heuristics to identify and remove affixes, making them widely applicable in text-processing tasks to enhance information retrieval and analysis.

Lemmatization

Lemmatization is a linguistic process that involves reducing words to their base or root form, known as the lemma. The goal is to normalize different inflected forms of a word so that they can be analyzed or compared more easily. This is particularly useful in natural language processing (NLP) and text analysis.

For example:

- running -> running
- jumps -> jumps
- swimming -> swimming

Here's how lemmatization generally works:

- Tokenization: The first step is to break down a text into individual words or tokens. This can be done using various methods, such as splitting the text based on spaces.
- POS Tagging: Parts-of-speech tagging involves assigning a grammatical category (like noun, verb, adjective, etc.) to each token. Lemmatization often relies on this information, as the base form of a word can depend on its grammatical role in a sentence.
- Lemmatization: Once each word has been tokenized and assigned a part-of-speech tag, the lemmatization algorithm uses a lexicon or linguistic rules

to determine the lemma of each word. The lemma is the base form of the word, which may not necessarily be the same as the word's root. For example, the lemma of "running" is "run," and the lemma of "better" (in the context of an adjective) is "good."

- Applying Rules: Lemmatization algorithms often rely on linguistic rules and patterns. For irregular verbs or words with multiple possible lemmas, these rules help in making the correct lemmatization decision.
- Output: The result of lemmatization is a set of words in their base or dictionary form, making it easier to analyze and understand the underlying meaning of a text.

Lemmatization is distinct from stemming, another text normalization technique. While stemming involves chopping off prefixes or suffixes from words to obtain a common root, lemmatization aims for a valid base form through linguistic analysis. Lemmatization tends to be more accurate but can be computationally more expensive than stemming.

Difference between stemming Vs. lemmatization

Stemming	Lemmatization
Stemming is a process that stems or removes the last few characters from a word, often leading to incorrect meanings and spelling.	Lemmatization considers the context and converts the word to its meaningful base form, which is called Lemma.
For instance, stemming the word ‘History’ would return ‘Histori‘.	For instance, lemmatizing the word ‘History‘ would return ‘History‘.
Stemming is used in cases of large datasets where performance is an issue.	Lemmatization is computationally expensive since it involves look-up tables and what not.

Stop words and Keywords Identification

In natural language processing (NLP), stop words are commonly used words that are removed from text processing tasks because they don't carry much meaning. Key words are identified using methods like YAKE, which analyzes a text's structure, word usage, and co-occurrence patterns.

	Stop words	**Key words**
Definition	Words that are commonly used but don't carry much meaning	Words that are most relevant to the content of a document
Examples	"A", "an", "the", "and", "is", "in", "on", "at", "with", "he", "she", "it", "they"	Identified by methods like YAKE
Purpose	Reduced computational load, storage requirements, and make data more manageable	Summarize a document's content

Stop words are often removed from text processing tasks to ensure that topically relevant documents rank highly in search results. For example, if a search query is "what is a stop word?", a search system can eliminate the words "what is a" and focus on documents that talk about stop words.

YAKE is a method that analyzes a text's structure, word usage, and co-occurrence patterns to identify the most relevant keywords and key phrases.

```
# Stopwords

import nltk

from nltk.corpus import stopwords

# Download NLTK stopwords and tokenizer models

nltk.download('stopwords')

nltk.download('punkt')

# Get English Language  stopwords

english_stopwords = set(stopwords.words('english'))

print(english_stopwords)
```

The above program will give all stop words in English.

What's the Difference Between Phonemes, Graphemes, and Morphemes?

What's A Phoneme?

A phoneme is the smallest unit of sound. A phoneme is the smallest unit of spoken sound and is often the one thing that distinguishes one word from another. For example, cat and rat are only differentiated by the first phoneme. In many cases, a single letter represents a single phoneme, but in most cases, there are multiple ways of representing a particular phoneme in English spelling. And in the case of the letter x, it is comprised of two phonemes /k/ & /s/.

Generally, the accepted belief is that there are 44 phonemes in English. This includes short vowel sounds, long vowel sounds, digraph sounds such as /sh/, /th/ (voiced and unvoiced), and /ch/, and single consonant sounds. Most people consider the diphthong sounds /oy/ and /ou/ to be single phonemes as well. Linguistically, /ng/ and /ar/, /or/, /er/, /ear/, /oar/, and schwa are also phonemes.

So, a single phoneme such as /n/ may be represented by letters in numerous different ways such as n, nn, kn, gn, or pn. Phonemes can be indicated through the International Phonemic Alphabet or by indicating a sound between slanted lines.

What's A Grapheme?

In linguistics, a grapheme is the smallest unit of a written language whether or not it carries meaning or corresponds to a single phoneme. In different languages, a grapheme may represent a syllable or unit of meaning.

Graphemes can include other printed symbols such as punctuation marks. In this example, the grapheme <x> represents the phonemes /k//s/ while a single character in Japanese may represent a syllable. Different types/fonts of a single letter are considered the same grapheme. A basic alphabetic understanding and rapid recognition is a necessary first step to learning to read. Your student's ability to quickly, accurately, and easily write graphemes is necessary for fluent writing and spelling.

When we are speaking purely about English, you will often see another definition of grapheme. **In this case, a grapheme is a letter or group of letters that represent a single phoneme.** This is a term used more or less synonymously with phonograms**.**

There are often numerous graphemes (or phonograms) that can represent a single phoneme. For example, the /ā/ sound is a phoneme that can be represented by numerous graphemes including ai, ay, ey, ei, eigh, a-e.
To make things even more confusing for young learners, a single grapheme such as ea may represent three different phonemes /ē/, /ā/, or /ĕ/. While English has 26 letters and 44 phonemes, there are approximately 250 graphemes.

What's A Morpheme?

Perhaps the most neglected term and concept in the study of teaching reading is the morpheme. **A morpheme is the smallest unit of meaning that cannot be further divided.** So, a base word might be a morpheme, but a suffix, prefix, or root also represents a morpheme. For example, the word red is a single morpheme, but the word unpredictable is made of the morphemes un + pre + dict + able. While none of these units stand alone as words, they are the smallest units of meaning.
Morphemes can vary tremendously in length from a single letter such as the prefix a- to a place name derived from another language such as Massachusetts, which in English represents a single morpheme. As students move into reading and writing more sophisticated academic language, the concept of morphemes becomes increasingly important to their decoding and their spelling as well as their ability to infer meanings of new vocabulary. The root of a word makes tremendous differences in spelling words with the -ion suffix for instance.