**NLP**

**Definition:** NLP stands for Natural Language Processing.

* Branch of AI.
* Gives ability to machine understand and process human languages.(text/audio)
* Started in 1950.
* Approaches for NLP.
  + Heuristics based NLP – initial approach based on defined rules.
    - Ex: regex.
  + Statistical Machine learning based NLP – based on statistical rules & ml algo.
    - Ex: SVM, Navie Bayes.
  + Neural Network based NLP – Latest approach, based on neural network. based learning, known as Deep learning.
    - Ex: RNNs, LSTMs,CNNs.

**Advantages of NLP:**

* Helps to analyse both structured and unstructured data soruces.
* Very fast and time efficient.
* Gives on point info about a question(exact/direct ans) in milliseconds.

**Disadvantages of NLP:**

* Needs lot of data and computation.
* Results are sometimes not to be accurate.

**Components of NLP:**

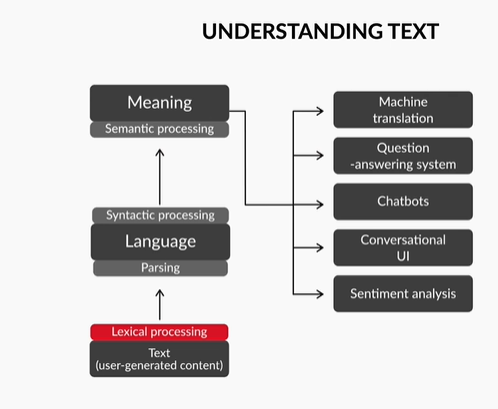
* Natural Language Understanding.
* Natural Language Generation.

**Application of NLP:**

* Text to speech – alexa, siri etc..
* Text classification – grammerly, microsoft docs
* Information extraction – google
* Chatbot – website bots
* Language translation – google translation
* Text summerization

**Process of NLP:**

Lexical analysis -> syntactic analysis -> semantic analysis -> pragmatic analysis -> Disclosure integration

****

**Lexcial analysis:**

* First step in NLP
* Breaking down input text into smaller units called tokens(words,puntuations,etc..)
* Remove irrelavent characters or noise from text

**Syntactic analysis:**

* Also known as parsing
* Analyze grammatical structure, to understand how tokens are arranged,how they relate to each other
* Ex: sub -verb – obj relationship

**Pragmatic analysis:**

* Understanding the implied meaning of text beyond its literal interpretation.
* Factors like context, speaker intention, inference, social cues

Ex: The dishes aren’t going to wash themselves

* + **Context:** You're at home, likely after a meal, and the dishes need to be cleaned.
  + **Speaker Intention:** Your mother is likely hinting that she wants someone to wash the dishes, possibly implying that it's your turn to do them.
  + **Inference:** Based on your knowledge of your family dynamics, you might understand that your mother is indirectly asking you to wash the dishes rather than explicitly stating it.
  + **Social Cues:** In many households, it's common for family members to take turns doing chores, and indirect language like this is often used to assign tasks.

**Discourse integration:**

* Combining the individual interpretations of sentences or phrases to understand entire text.
* Connects the related piecees of information, and organizing the the extracted information into structured representation.
* It ensures that overall meaning of the text is properly understood cand be used for further analysis

Ex:

* Mom: "The dishes aren't going to wash themselves."
* You: "I'll do them after I finish my homework."
* Your sister: "I can help you with the dishes."
* Dad: "That sounds like a good plan. I'll take care of vacuuming the living room while you two do the dishes."
* Now, let's integrate this dialogue into a coherent discourse:
* **Context:** The family is at home, and it's after dinner. The dishes need to be washed, and there are other household chores to be done.
* **Pragmatic Analysis:** Your mother's statement indicates that someone needs to wash the dishes. Your response acknowledges the task but also mentions your homework, indicating your availability. Your sister offers to help, and your father suggests a plan to divide the chores.
* **Inference:** Everyone in the family understands their roles and responsibilities. Your willingness to do the dishes after finishing homework is understood, and your sister's offer and your dad's plan show cooperation and teamwork.
* **Social Cues:** The conversation reflects common family dynamics where chores are shared, and tasks are coordinated based on each person's availability and abilities.
* So, through discourse integration, the family members collaboratively organize and assign tasks, ensuring that everyone contributes to the household responsibilities in a coordinated and efficient manner.

**NLTK ( Natural Language ToolKit )**

* It can perform variety of operations on textual data
  + **Classification** – *label the data based on its content*
  + **Tokenization** – *breaking down text into individual words/phrases/tokens*
  + **Stemming** – *reducing words to their base form to normalize variations of words.*
  + **Tagging** – *assigning parts – of – speech tags to words in a sentence*
  + **Leparsing** – analyzing the grammatical structure of sentences to determine syntactic relationships.
  + **Semantic reasoning** – *inferring meaning from text by analyzing context*

**Types of data used in NLP:**

1. **Sturctured dataset** – fixed dimensions

Ex: Tabular format data

1. **Unstructured dataset** – no fixes dimensions, no sturcture

Ex: Images, Audios, Videoes, Text data

**Text data:**

* Words arranged in a meaningful manner
* Written form of language
* Grammer and defined structures

**Examples:**

* **social media**: tweets, posts, comments
* **Conversations**: messages, emails, chats
* **Articles**: news, blogs, transcripts

**What are Regular Expressions?**

Regular Expressions or Regex are the combination of special characters which are also called patterns(represents a textual pattern)

* Also called as wild card expressions uses for matching, searching, and parsing strings.
* Used for writing rule based information mining systems.

**Why we need Regex?**

To do

* Segmentation of words from sentences.
* Segmentation of sentences from paragraphs.
* This process is also called as Tokenization.
* Cleaning text – noise removal
* Information retrieval from texts ( ex: chatbots, news datasets, search engine queries )

**Types of Regex:**

**Regular Expression Functions:**

* **Match:** finds the first occurrence of pattern in the string
* **Search:** locates the pattern In the string
* **Findall:** find all occurrences of patterns in the string
* **Sub:** search and replace
* **Split:** split the text by the given regular expression pattern

**Anchors:**  
In regular expressions (regex), anchors are special characters used to specify the position of a pattern within a string. They do not match any characters themselves but rather represent specific positions in the string. Here's a brief overview of the most commonly used anchors:

1. **^ (Caret)**: Matches the start of a line or string.
   * Example: "^hello" matches "hello" only if it appears at the start of a line or string.
2. **$ (Dollar)**: Matches the end of a line or string.
   * Example: "world$" matches "world" only if it appears at the end of a line or string.
3. **\b (Word Boundary)**: Matches a word boundary, which is the position between a word character (\w) and a non-word character (\W).
   * Example: "\bword\b" matches "word" only if it appears as a whole word, not as part of another word.

**Meta Characters:**

Meta characters in regular expressions (regex) are special characters that have a special meaning and are used to define patterns for matching text. Here's a concise explanation with examples:

1. **. (Dot)**: Matches any single character except newline.
   * Example: "b.t" matches "bat", "bit", "bot", etc.
2. **^ (Caret)**: Matches the start of a line or string.
   * Example: "^hello" matches "hello" only if it appears at the start.
3. **$ (Dollar)**: Matches the end of a line or string.
   * Example: "world$" matches "world" only if it appears at the end.
4. **\ (Backslash)**: Escapes the next character, treating it as a literal character.
   * Example: "$" matches the dollar sign "$" literally.
5. **[] (Character Class)**: Matches any one of the characters inside the brackets.
   * Example: "[abc]" matches "a", "b", or "c".
6. **| (Alternation)**: Matches either the pattern on its left or the pattern on its right.
   * Example: "hello|world" matches "hello" or "world".
7. **() (Grouping)**: Groups multiple tokens together and creates a capturing group.
   * Example: "(abc)+" matches "abc", "abcabc", "abcabcabc", etc.
8. **\d**: Matches any digit character (equivalent to [0-9]).
   * Example: "\d+" matches one or more digits.
9. **\w**: Matches any word character (equivalent to [a-zA-Z0-9\_]).
   * Example: "\w+" matches one or more word characters.
10. **\s**: Matches any whitespace character (space, tab, newline).
    * Example: "\s+" matches one or more whitespace characters.

Meta characters provide powerful tools for creating complex regex patterns to match specific text patterns within a string

**Quantifiers:**

Quantifiers in regular expressions (regex) specify the number of occurrences of a preceding element in a pattern. Here's a concise explanation with examples:

1. **\* (Zero or More)**: Matches zero or more occurrences of the preceding element.
   * Example: "ab\*" matches "a", "ab", "abb", "abbb", etc.
2. **+ (One or More)**: Matches one or more occurrences of the preceding element.
   * Example: "ab+" matches "ab", "abb", "abbb", etc., but not "a".
3. **? (Zero or One)**: Matches zero or one occurrence of the preceding element.
   * Example: "ab?" matches "a" or "ab", but not "abb".
4. **{n} (Exactly n)**: Matches exactly n occurrences of the preceding element.
   * Example: "a{3}" matches "aaa".
5. **{n,} (At Least n)**: Matches n or more occurrences of the preceding element.
   * Example: "a{2,}" matches "aa", "aaa", "aaaa", etc.
6. **{n,m} (Between n and m)**: Matches between n and m occurrences of the preceding element, inclusive.
   * Example: "a{2,4}" matches "aa", "aaa", or "aaaa".

Quantifiers provide flexibility in defining the repetition of elements in a regex pattern, allowing for precise matching of text patterns with varying lengths.

**Flags:**

Regex flags are used to modify the behavior of regular expressions in Python. Here are some common regex flags with examples:

1. **re.IGNORECASE (or re.I)**: Ignores case when matching.
   * Example: **re.search("hello", "Hello, world!", re.IGNORECASE)**
2. **re.MULTILINE (or re.M)**: Treats the beginning and end of lines (^ and $) as the start and end of the entire string.
   * Example: **re.findall("^hello", "hello\nworld", re.MULTILINE)**
3. **re.DOTALL (or re.S)**: Allows the dot (.) to match newline characters.
   * Example: **re.search("hello.\*world", "hello\nworld", re.DOTALL)**
4. **re.VERBOSE (or re.X)**: Allows the use of whitespace and comments within the regex pattern for readability.
   * Example:

python code

pattern = r''' hello # Match 'hello' \s+ # Match one or more whitespace characters world # Match 'world' ''' re.search(pattern, "hello world", re.VERBOSE)

1. **re.ASCII (or re.A)**: Makes \w, \W, \b, \B, \d, \D, \s, and \S perform ASCII-only matching.
   * Example: **re.findall(r"\w+", "hello world", re.ASCII)**

These flags can be combined using the bitwise OR operator (|) to achieve the desired behavior in regex matching.

**Wildcards and character sets** are essential components of regular expressions (regex) used for pattern matching. Here's a concise explanation of each with examples:

1. **Wildcards**:
   * Wildcards are symbols that match any character or set of characters in a string.
   * The most common wildcard is the dot (.), which matches any single character except newline (\n).
   * Example:
     + **h.t** matches "hat", "hot", "hit", etc.
     + **h.\*t** matches "hat", "hoot", "hott", "hit", "how many t's", etc.
2. **Character Sets**:
   * Character sets allow you to match specific characters or ranges of characters.
   * They are enclosed in square brackets ([]).
   * Example:
     + **[aeiou]** matches any vowel.
     + **[0-9]** matches any digit.
     + **[A-Za-z]** matches any uppercase or lowercase letter.
   * You can also use negation (^) within a character set to match any character not listed.
   * Example:
     + **[^0-9]** matches any non-digit character.

Using wildcards and character sets in combination with other regex elements like quantifiers and anchors allows for powerful pattern matching capabilities in text processing and data extraction tasks.