2️⃣

NLP - Module 2 Gaurav Panjabi

What is morphology? Why do we need to do Morphological Analysis? <IMP full topic>

- **Morphology** is the branch of linguistics that deals with the internal structure and formation of words in a language.
- It involves the study of morphemes, which are the smallest units of meaning in a language.
- The morpheme is the smallest element of a word that has grammatical function and meaning.
- **Types of morphemes:**
  - **Root morphemes:** The core meaning-bearing elements, like "write" in "writer" or "happy" in "unhappy."
  - **Affixes:** Morphemes attached to a root to modify its meaning or grammar. They can be:
    - **Prefixes:** Added before the root, like "un-" in "unhappy" or "re-" in "rewrite."
    - **Suffixes:** Added after the root, like "-er" in "writer" or "-s" in "books."
- **Word formation processes:** How morphemes combine to form words:
  - **Inflection:** Modifying a word's grammatical features like tense, number, or case (e.g., "run," "runs," "running").
  - **Derivation:** Creating new words with different meanings from existing ones (e.g., "happy," "unhappy," "happiness").
- Morphology explores how these morphemes combine to create words and how words are further manipulated to convey different meanings.
- Morphology helps us understand how words are formed and modified in a language. It examines the rules and processes governing the addition of prefixes, suffixes, and infixes to base forms to create new words.
- Morphology often conveys information about the meaning of words. By analyzing morphological components, we can identify nuances in meaning, such as tense, number, gender, and case in inflected languages.
- For language learners, morphological analysis is a key aspect of understanding vocabulary and grammar. Recognizing morphological patterns allows learners to deduce the meanings of unfamiliar words and predict the grammatical behavior of words.
- Morphology plays a role in syntactic analysis by providing information about the grammatical function of words within sentences. For example, the morphological structure of a word can indicate its syntactic category (e.g., noun, verb, adjective).
- **Inflectional Morphology and Derivational Morphology are the two types of morphology.**

→ Types of Morphology

Inflectional Morphology

- Inflectional morphology refers to the modification of a base or root word through the addition of morphemes that convey grammatical information.
- These modifications typically indicate various grammatical features such as tense, aspect, mood, number, gender, case, and person.

- Unlike derivational morphology, which creates new words or changes the lexical category of a word, inflectional morphology doesn't fundamentally alter the word's meaning or category.

Mechanisms:

- **Affixation:** Adding prefixes, suffixes, or infixes to the word stem (e.g., un-happy, play-ing, man-kind)
- **Internal Ablaut:** Changing vowels within the word stem (e.g., sing / sang / sung)
- **Suppletion:** Replacing the entire stem with a different form (e.g., go / went)

Functions:

- **Number:** Indicating singularity or plurality (e.g., book / books, child / children)
- **Tense:** Marking timeframes for verbs (e.g., write / wrote / written)
- **Aspect:** Expressing duration or completion of actions (e.g., read / reading)
- **Case:** Defining a noun's role in a sentence (e.g., I see the boy / the boy sees me)
- **Degree:** Comparing adjectives and adverbs (e.g., happy / happier / happiest)
- **Gender:** Marking masculine, feminine, or neutral (in some languages)

Inflectional Morphology Examples:

- **Number (plural):**
  - Book -> books
  - Child -> children
- **Tense (past):**
  - Write -> wrote
  - See -> saw
- **Aspect (progressive):**
  - Read -> reading
  - Walk -> walking
- **Case (possessive):**
  - John's book
  - The cat's toy
- **Degree (comparative):**
  - Happy -> happier
  - Fast -> faster
- **Gender (feminine):**
  - Actor -> actress (in some languages)
  - Lion -> lioness

Derivational Morphology

- **Derivational morphology** is a branch of morphology that explores how new words are created from existing ones through the addition of **derivational morphemes**.
- These morphemes often change the word's meaning, part of speech, or both, expanding the possibilities of language expression.

- Unlike inflectional morphology, which typically conveys grammatical information such as tense, number, or gender, derivational morphology often results in a change in the lexical category (part of speech) or the fundamental meaning of a word.

Here are the key aspects of derivational morphology:

How It Works:

- **Affixation:**
  - **Prefixes:** Added before the root to modify meaning (e.g., **un-**happy, **re-**write, **pre-**determine).
  - **Suffixes:** Added after the root to change meaning or part of speech (e.g., teach**-er**, govern**-ment**, friend**-ship**).
- **Other Processes:**
  - **Conversion:** Changing a word's part of speech without altering its form (e.g., "water" as a noun and verb).
  - **Back-formation:** Creating a new word by removing a supposed affix (e.g., "edit" from "editor").
  - **Compounding:** Joining two or more words to create a new one (e.g., "sunflower," "blackboard").

Functions:

- **Creating New Lexemes:** Generating new words with distinct meanings or parts of speech (e.g., "happy" -> "happiness," "act" -> "actor").
- **Expressing Semantic Relations:** Conveying relationships between words (e.g., "legal" -> "illegal," "nation" -> "national").
- **Changing Word Class:** Transforming a word's grammatical category (e.g., "act" (verb) -> "action" (noun)).

Derivational Morphology Examples:

- **Adding prefixes:**
  - *un-**happy (changes meaning to negation)
  - *pre-**determine (changes meaning to before)
  - *re-**write (changes meaning to repetition)
- **Adding suffixes:**
  - teach**-er** (changes part of speech from verb to noun)
  - govern**-ment** (changes part of speech from verb to noun)
  - friend**-ship** (creates a noun expressing a relationship)
- **Conversion:**
  - "Water" (noun) -> "to water" (verb)
  - "Mail" (noun) -> "to mail" (verb)
- **Compounding:**
  - "Sunflower" (sun + flower)
  - "Blackboard" (black + board)
  - "Homework" (home + work)

Differentiate between Derivational and Inflectional morphemes.

Feature	Derivational Morphology	Inflectional Morphology
Function	Creates new words or changes the part of speech of existing words	Modifies existing words to convey grammatical information (e.g., tense, number,

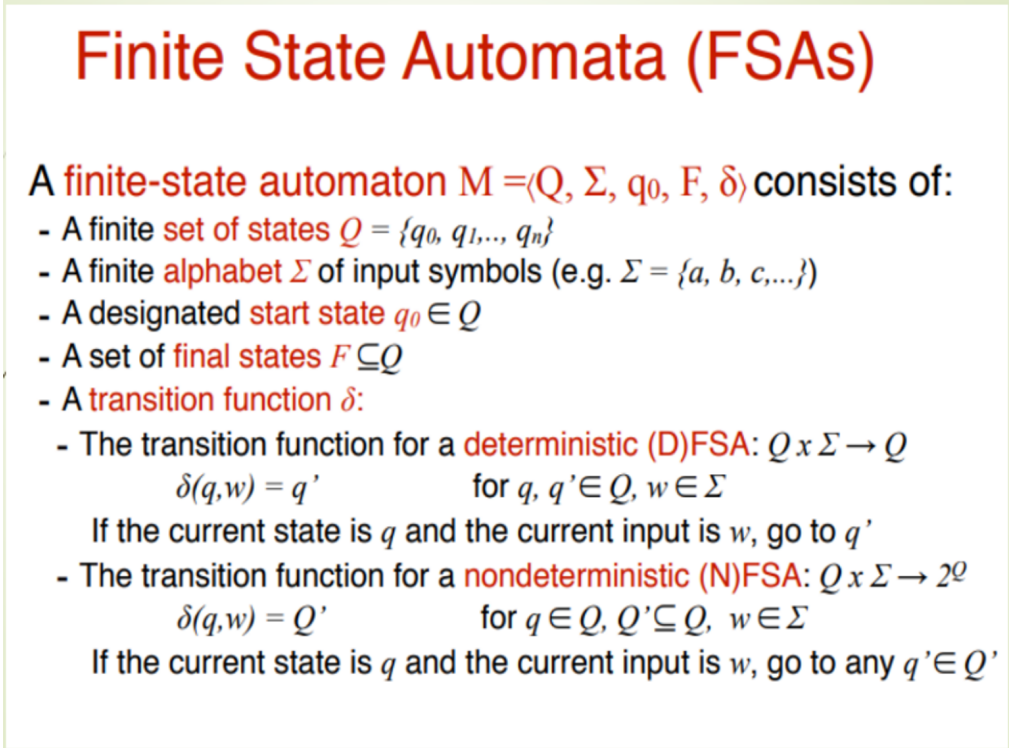
		case)
Meaning change	Significant change in meaning or creates a new one	Minimal or no change in meaning
Part of speech change	Can change or create a new part of speech	Does not change part of speech
Morpheme placement	Usually prefixes or suffixes, but can also involve internal changes or compounding	Usually suffixes, less often prefixes, and rarely internal changes
Number of forms	Creates many new forms or new words	Limited number of forms for each word
Productivity	Highly productive, can be applied to many roots	Relatively less productive, limited to specific patterns for each grammatical category
Predictability	Less predictable, meaning and part of speech change depend on the specific morpheme	More predictable, effect on meaning and part of speech is consistent for each grammatical category
Examples	**un-happy, book-case, water (verb), sunflower**	**books, wrote, reading, John's, happier**

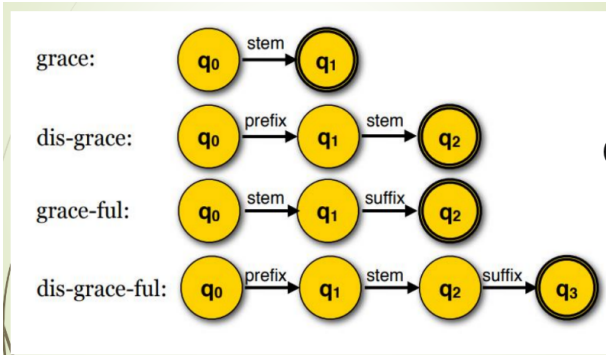
Represent output of morphological analysis for Regular verb, Irregular verb, singular noun, plural noun

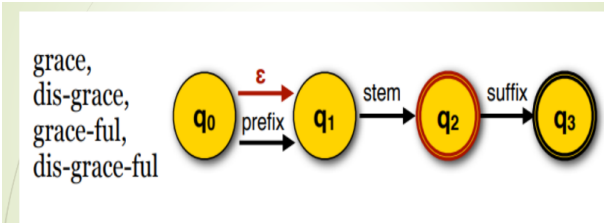
1. **Regular Verb:**
   - Morphological analysis of the regular verb "walk":
     - Base Form: "Walk"
     - Past Tense: "Walked"
     - Present Participle: "Walking"
   - Regular verbs follow a predictable pattern for tense and participle formation.
2. **Irregular Verb:**
   - Morphological analysis of the irregular verb "sing":
     - Base Form: "Sing"
     - Past Tense: "Sang"
     - Past Participle: "Sung"
   - Irregular verbs do not follow the standard patterns for tense and participle formation.
3. **Singular Noun:**
   - Morphological analysis of the singular noun "cat":
     - Nominative Form: "Cat"
     - Genitive Form: "Cat's"
   - Nouns typically have distinct forms for the singular and genitive (possessive) cases.
4. **Plural Noun:**
   - Morphological analysis of the plural noun "cats":
     - Nominative Form: "Cats"
     - Genitive Form: "Cats'"
   - Plural nouns often involve the addition of an "-s" for the nominative case and an apostrophe for the genitive case.

Explain the role of FSA in morphological analysis? (FSA BHI ISI M HE MENTIONED H) <V.IMP>

- The theory of automata provides efficient and convenient tools for the representation of linguistic phenomena.
- The theory of automata plays a significant role in providing solutions of many problems in natural language processing. For example ,speech recognition, spelling correction, information retrieval etc.
- Finite State Automata (FSA) is a mathematical model consisting of states, transitions, and input symbols. It operates in a step-by-step manner, moving through states based on input symbols, making it well-suited for sequential processing. This formal tool serves as a computational model for representing and processing morphological rules.







- **Formal Representation:**
  - FSA provides a structured model for morphological rules.
  - Example: FSA for English verb conjugation captures the transition from base form to past tense.

- **Efficient Parsing:**
  - FSA enables sequential processing for recognizing word patterns.
  - Example: FSA efficiently checks regular plural nouns by transitioning through states.
- **Recognition of Valid Forms:**
  - FSA distinguishes and recognizes valid word forms.
  - Example: FSA identifies regular and irregular verb forms based on defined patterns.
- **Generation of Inflected Forms:**
  - FSA can generate inflected forms based on defined transitions.
  - Example: FSA creates comparative and superlative forms of adjectives.
- **Rule-based Analysis:**
  - FSA facilitates rule-based analysis of morphological structures.
  - Example: FSA analyzes derivational suffixes to understand base-to-derived form transformations.
- **Handling Morpheme Concatenation:**
  - FSA adeptly represents morpheme concatenation, crucial in agglutinative languages.
  - Example: FSA transitions through prefixes, stems, and suffixes in constructing verb forms.
- **Adaptability to Linguistic Complexity:**
  - FSA can be adapted for different linguistic complexities.
  - Example: FSA flexibly represents fusion in verb conjugation for fusional languages.

Significance in Morphological Analysis

- Inflection Analysis: The FSA helps analyze the inflectional variations of the verb "talk." By following transitions based on observed input characters, the automaton can determine the tense of the verb form.
- Tense Recognition: Given an input word like "talked," the FSA can traverse the transitions and determine that the word is in the "Past" tense.
- Lemmatization: The FSA can assist in lemmatization by identifying the base form of a verb. For example, when given the input "talking," the automaton can recognize that it's a variation of the base form "talk."
- Pattern Matching: The FSA can be used for pattern matching to recognize valid verb forms. For instance, it can identify that "will talk" is a valid future tense construction.
- Morpheme Extraction: The transitions in the FSA help extract morphemes from the input. For "talked," the FSA can identify the root "talk" and the suffix "-ed."
- Error Detection: The FSA can help detect errors such as mismatched tenses. If the input is "will talked," the automaton can identify the inconsistency.
- Stemming: By considering the transitions and recognizing common suffixes, the FSA can be adapted for stemming. For instance, "talking" can be stemmed to "talk."

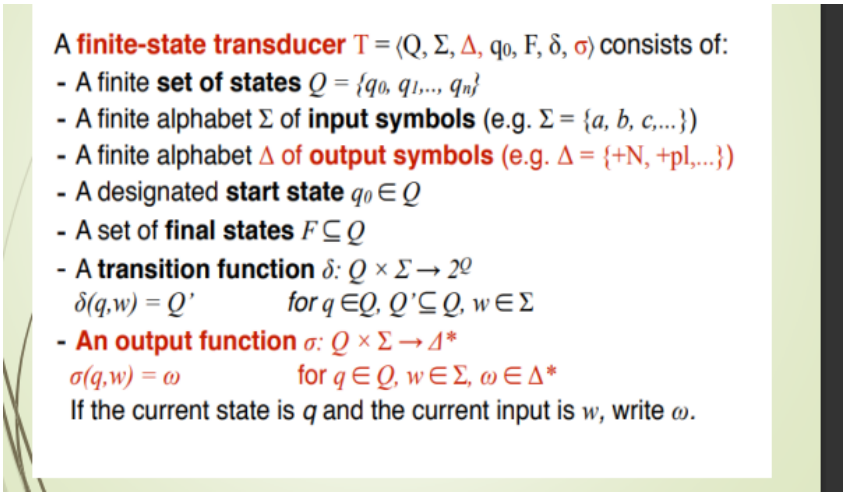
Explain finite state transducer(FST).

FSTs are devices that define a relationship between two sets of symbols, where each symbol in the input set corresponds to a sequence of symbols in the output set.

They are essentially state machines that process inputs and generate outputs. Unlike Finite-State Automata (FSAs) which only recognize strings, FSTs map one sequence of symbols (input) to another sequence of symbols (output). Think of it as a machine that reads one string and transforms it into another.

Here are the key components and features of a Finite State Transducer:

1. **States:**
   - Like other finite state machines, FSTs have a finite set of states. Each state represents a specific configuration or condition of the transducer during its operation.
2. **Transitions:**
   - Transitions describe how the FST moves from one state to another based on input and output symbols. A transition is defined by a triple (current state, input symbol, output symbol) that determines the next state and the symbols to be written to the output.
3. **Input and Output Alphabets:**
   - FSTs have input and output alphabets, which are sets of symbols that can be processed. These symbols can be individual characters, phonemes, or other units depending on the application.
4. **Initial State:**
   - The initial state is the starting point of the FST. It is the state the FST enters when processing begins.
5. **Final States:**
   - Some states are designated as final states. When the FST reaches a final state, it signals the completion of the input-output transformation.
6. **Transition Function:**
   - The transition function specifies the next state and output symbol for each combination of current state and input symbol. It defines the behavior of the FST.
7. **Input-Output Mapping:**
   - FSTs are designed to map sequences of input symbols to sequences of output symbols. The mapping is determined by the transitions and the input-output pairs associated with each transition.
8. **Acceptance Criteria:**
   - Depending on the application, an FST might accept or reject a particular input sequence based on its configuration when the sequence is processed.



Benefits of using FSTs:

- **Simplicity and efficiency:** FSTs are based on simple state transitions and can be computationally efficient for processing large amounts of text.

- **Modular and flexible:** They can be combined and customized for specific tasks, making them adaptable to different needs.
- **Formal framework:** FSTs provide a well-defined mathematical framework for analyzing and manipulating language data.

Explain use of CFG in Natural Language Processing with suitable example

In Natural Language Processing (NLP), Context-Free Grammars (CFGs) are widely used to describe the syntactic structure of natural languages. CFGs provide a formal framework for representing the hierarchical structure of sentences in terms of constituents, such as phrases and clauses. Here's how CFGs are utilized in NLP, along with a suitable example:

Context-free grammar G is a 4-tuple.

G = (V, T, S, P)

These parameters are as follows;

- **V** – ***Set of variables*** (also called as **Non-terminal symbols**)
- **T** – ***Set of terminal symbols*** (**lexicon**)
- The symbols that refer to words in a language are called **terminal symbols**.
- **Lexicon** is a set of rules that introduce these symbols.
- **S** – ***Designated start symbol*** (one of the non-terminals, S ∈ V)
- **P** – ***Set of productions*** (also called as **rules**).
  - Each rule in P is of the form **A** → **s**, where
  - A is a non-terminal (variable) symbol.
  - Each rule can have only one non-terminal symbol on the left hand side of the rule.
  - s is a sequence of terminals and non-terminals. It is from (T U V)*, infinite set of strings.

Use of CFG in NLP:

1. **Parsing:**
   - CFGs are used for syntactic parsing, the process of analyzing the grammatical structure of sentences. Parsing involves breaking down a sentence into its constituent parts and representing the hierarchical relationships between them.
2. **Grammar Development:**
   - Linguists and computational linguists use CFGs to develop grammars that capture the syntactic rules and structures of a particular language. These grammars serve as a foundation for building parsing models and understanding sentence structure.
3. **Sentence Generation:**
   - CFGs can be employed to generate valid sentences in a language. By starting with a sentence symbol and applying production rules, CFGs can generate a variety of sentences that conform to the specified syntactic structure.
4. **Ambiguity Resolution:**
   - CFGs help identify and handle syntactic ambiguity in natural language sentences. Ambiguity arises when a sentence can be parsed in multiple ways. CFGs provide a framework for disambiguating and selecting the most appropriate parse.

Example CFG for a Simple English Sentence:

Consider a simplified CFG for a basic English sentence structure with a subject, a verb, and an object:

1. **Non-terminal symbols:**

- S (sentence)
- NP (noun phrase)
- VP (verb phrase)
- Det (determiner)
- N (noun)
- V (verb)

2. **Terminal symbols:**
   - "the", "a" (determiners)
   - "cat", "dog" (nouns)
   - "chases", "catches" (verbs)
3. **Production rules:**
   - S → NP VP
   - NP → Det N
   - VP → V NP
   - Det → "the" | "a"
   - N → "cat" | "dog"
   - V → "chases" | "catches"

Using this CFG, we can generate sentences. Let's generate a sentence:

1. Start with the sentence symbol S.
2. Apply the rule S → NP VP.
3. Apply the rule NP → Det N to generate "the cat."
4. Apply the rule VP → V NP to generate "chases NP."
5. Apply the rule Det → "the" to generate "the."
6. Apply the rule N → "cat" to generate "cat."
7. Apply the rule V → "chases" to generate "chases."

Putting it all together, we get the sentence "the cat chases the cat."

What is lexicon, lexeme and explain the different types of relations the hold between lexemes with example.

Lexicon:

- The mental dictionary or vocabulary of a language, containing all the words and their associated information.
- It's like a vast storehouse of words, their meanings, pronunciations, grammatical properties, and relationships.

Lexeme:

- The basic unit of meaning in a lexicon.
- It represents a core concept or idea, independent of its specific grammatical form or pronunciation.
- Think of it as the abstract representation of a word, encompassing all its possible variations.

Examples:

- The lexeme "walk" includes forms like "walk," "walks," "walked," "walking."
- The lexeme "good" includes forms like "good," "better," "best."

Relationships Between Lexemes:

1. Synonymy:

- Two lexemes have the same or nearly the same meaning.
- **Example:** "happy" and "joyful"

2. Antonymy:

- Two lexemes have opposite meanings.
- **Example:** "hot" and "cold"

3. Hyponymy:

- One lexeme is a specific instance or subtype of another.
- **Example:** "dog" is a hyponym of "animal"

4. Hypernymy:

- One lexeme is a more general or superordinate category of another.
- **Example:** "animal" is a hypernym of "dog"

5. Meronymy:

- One lexeme denotes a part of another.
- **Example:** "wheel" is a meronym of "car"

6. Holonymy:

- One lexeme denotes a whole of which another is a part.
- **Example:** "car" is a holonym of "wheel"

7. Polysemy:

- A single lexeme has multiple related meanings.
- **Example:** "bank" can refer to a financial institution or the edge of a river

Stemming and Lemmatization

Stemming:

Stemming involves removing suffixes from words to obtain their root or base form. The goal is to reduce words to a common stem, even if the resulting stem may not be a valid word.

Example:
Consider the word "running." The stem obtained through stemming might be "run." Similarly, "flies" might be stemmed to "fli."

Popular stemming algorithms include the Porter Stemmer, Lancaster Stemmer, and Snowball Stemmer.

Advantages:

1. **Computational Efficiency:**
    - Stemming is generally faster and computationally less intensive compared to lemmatization. This makes it suitable for applications where speed is crucial, such as information retrieval or search engines.
2. **Simplicity:**

- Stemming algorithms are often simpler to implement and understand. They use heuristics to remove suffixes without requiring a detailed understanding of linguistic rules.

Disadvantages:

1. **Over-Stemming:**
   - Stemming can lead to over-stemming, where different words are reduced to the same stem even if they have different meanings. This may result in a loss of semantic information.
2. **Inaccuracy:**
   - Stemming may produce stems that are not valid words, making it less suitable for applications where the interpretability of results is crucial.

Use Case:

- Suitable for tasks where speed is a priority, and a rough normalization of words is sufficient.
- Often used in information retrieval, search engines, and text indexing.

Lemmatization:

Lemmatization is a more sophisticated process that involves reducing words to their base or dictionary form (lemma). Unlike stemming, lemmatization produces valid words, and it takes into account the context and part of speech of the word.

Example:
Consider the word "running" again. The lemma obtained through lemmatization would be "run." Similarly, "better" might be lemmatized to "good."

Lemmatization often requires access to a lexicon or a morphological analysis of words to determine their base form accurately.

Advantages:

1. **Semantic Accuracy:**
   - Lemmatization provides a more accurate representation of the base form of a word, preserving its semantic meaning. This is important in tasks that require a deeper understanding of language semantics.
2. **Context Awareness:**
   - Lemmatization takes into account the context and part of speech of a word, allowing for more precise normalization. This is beneficial in applications where word relationships and grammatical correctness matter.

Disadvantages:

1. **Computational Complexity:**
   - Lemmatization is generally more computationally complex compared to stemming, as it often requires access to a lexicon or morphological analysis to accurately determine the base form.
2. **Speed:**
   - Due to its complexity, lemmatization may be slower than stemming. In real-time applications where speed is crucial, this can be a limitation.

Use case:

- Preferred when semantic accuracy and context-awareness are crucial.
- Beneficial in applications like machine translation, sentiment analysis, and tasks requiring a deep understanding of language semantics.

Explain Porter's stemming algorithm (in detail. and with rules) with suitable examples. <IMP>

- Developed by Martin Porter in 1980.
- Algorithm for reducing English words to their root forms, called stems.
- Removes common suffixes in a sequence of steps, each containing a set of rules.
- Key applications in information retrieval (IR) and text mining.

Key Concepts:

- **Stem:** The base of a word after removing affixes (suffixes and prefixes).
- **Suffix:** A group of letters added to the end of a word to change its meaning or form.
- **Consonant (C):** Any letter except A, E, I, O, U, or Y unless Y follows a consonant.
- **Vowel (V):** A, E, I, O, U, or Y (unless Y follows a consonant).
- **Measure:** Number of VC sequences in the stem before a suffix.

Step-by-Step Breakdown:

- **Step 1:** Deals with simple suffixes like "-sses", "-ies", and "-s". Rules are applied based on consonant-vowel patterns and stem length.
- **Step 2:** Focuses on "-ed", "-ing", and related suffixes. Removal depends on measure and vowel-consonant configurations.
- **Step 3:** Targets longer suffixes like "-ational", "-tional", and "-izer". Rules consider measure and specific vowel-consonant patterns.
- **Step 4:** Handles suffixes like "-alism", "-iveness", and "-ousness". Measure and vowel-consonant patterns remain key factors in determining removal.
- **Step 5:** Addresses suffixes like "-icate", "-ative", and "-ical". Rules employ measure and vowel-consonant patterns like in previous steps.

Steps and Rules:

Step 1: Suffixes SSES, IES, SS, S

- **Rule 1a:** SSES -> SS if the stem ends with a consonant-vowel-consonant sequence (e.g., caresses -> caress)
- **Rule 1b:** IES -> I if the stem ends with a consonant-vowel-consonant sequence (e.g., ponies -> poni)
- **Rule 1c:** IES -> Y if the stem ends in a vowel (e.g., ties -> ti)
- **Rule 1d:** SS -> S (e.g., caress -> cares)
- **Rule 1e:** S -> (remove if the stem has more than one letter and ends with a consonant-vowel-consonant sequence) (e.g., cats -> cat)

Step 2: Suffixes (m>0) ED, ING, edly, ing, ingly

- **Rules based on measure and vowel-consonant patterns**
- Examples:
  - feed -> feed (measure = 0)
  - agreed -> agree (measure > 0)
  - motoring -> motor (measure > 0)
  - sing -> sing (measure = 0)

Step 3: Suffixes (m>0) ational, tional, enci, anci, izer, ization, ation, ator

- **Rules based on measure and vowel-consonant patterns**
- Examples:

- relational -> relate (measure > 0)
- rational -> rational (measure = 0)
- dependency -> depend (measure > 0)
- appliance -> appli (measure > 0)

Step 4: Suffixes alism, iveness, fulness, ousness

- **Rules based on measure and vowel-consonant patterns**
- Examples:
  - capitalism -> capitalist (measure > 0)
  - decisiveness -> decisive (measure > 0)
  - hopefulness -> hopeful (measure > 0)

Step 5: Suffixes icate, ative, alize, iciti, ical

- **Rules based on measure and vowel-consonant patterns**
- Examples:
  - triplicate -> triplic (measure > 0)
  - formative -> form (measure > 0)
  - formalize -> formal (measure > 0)
- **Strengths:**
  - Simple and efficient algorithm.
  - Improves precision and recall in IR tasks.
  - Widely used and implemented in various NLP tools.
- **Limitations:**
  - Rule-based, leading to exceptions and inconsistencies.
  - Over-stemming (reducing words to incorrect stems).
  - Not perfect for morphologically complex languages.

Define affixes. Explain the types of affixes.

Affixes are linguistic elements that are attached to a base or root word to modify its meaning or create a new word. They are an essential part of morphology, which is the study of the structure and formation of words. Affixes can be added to the beginning (prefixes), the end (suffixes), or in the middle (infixes) of a base word. In natural language processing (NLP), understanding and analyzing affixes is crucial for tasks such as stemming, lemmatization, and morphological analysis.

Here are the main types of affixes:

1. **Prefixes:**
   - **Definition:** Prefixes are affixes added to the beginning of a base word to change its meaning.
   - **Example:** "Un-" is a common prefix used to indicate the opposite or negation. For instance, "happy" becomes "unhappy" when the prefix "un-" is added.
2. **Suffixes:**
   - **Definition:** Suffixes are affixes added to the end of a base word to alter its meaning.
   - **Example:** "-ly" is a suffix often added to adjectives to form adverbs. For example, "quick" becomes "quickly" when the suffix "-ly" is added.

3. **Infixes:**
    - **Definition:** Infixes are affixes inserted into the middle of a base word.
    - **Example:** Infixes are less common in English, but they are found in some informal expressions, like "parser-s-by" where "s"' is an infix.
4. **Circumfixes:**
    - **Definition:** Circumfixes are affixes that consist of two parts, one placed at the beginning and the other at the end of a base word.
    - **Example:** The verb "form" can become "reformed" through the addition of the circumfix "re-" and "-ed" to indicate the past participle.
5. **Derivational Affixes:**
    - **Definition:** Derivational affixes create a new word with a different meaning or belonging to a different word class (e.g., changing a noun to a verb).
    - **Example:** Adding the suffix "-er" to the noun "teach" creates the verb "teacher."
6. **Inflectional Affixes:**
    - **Definition:** Inflectional affixes do not change the basic meaning or word class of a word but indicate grammatical features such as tense, number, or gender.
    - **Example:** In English, the plural suffix "-s" in "cats" is an inflectional affix.

Describe open class words and closed class words in English with examples.

In natural language processing (NLP), words are often categorized into two main classes: open class words and closed class words. These classifications are helpful for understanding the roles different types of words play in a language.

1. **Open Class Words:**
    - **Definition:** Open class words are lexical items that can easily accept new members, and the category is open to expansion. These words are typically content words that convey meaningful information and contribute to the core meaning of a sentence. New words are frequently added to open class categories as languages evolve.
    - **Examples:**
        - **Nouns:** dog, computer, happiness
        - **Verbs:** run, think, create
        - **Adjectives:** happy, blue, tall
        - **Adverbs:** quickly, often, well
    - **Characteristics:**
        - Open class words are subject to change and expansion.
        - New words can be easily coined and added to these categories.
        - These words are generally more concrete and specific.
2. **Closed Class Words:**
    - **Definition:** Closed class words, on the other hand, are grammatical words that serve a more structural or functional role in a sentence. These words have a relatively fixed set, and it is rare for new words to be added to these categories. They play a crucial role in the syntax and structure of sentences.
    - **Examples:**
        - **Pronouns:** I, you, he, she, it

- **Prepositions:** on, in, under, beside
- **Conjunctions:** and, but, or
- **Articles:** the, a, an

- **Characteristics:**
  - Closed class words have a more stable and limited set of members.
  - These words are often function words that connect or relate other elements in a sentence.
  - They are crucial for the grammatical structure and coherence of sentences.

Describe open class words and closed class words in English with examples.

Feature	Closed Classes	Open Classes
Definition	Word classes with a finite, fixed set of members	Word classes that can readily accept new members
Examples	Pronouns, prepositions, conjunctions, determiners, auxiliary verbs	Nouns, verbs, adjectives, adverbs
Role	Primarily serve grammatical functions, indicating relationships between other words	Carry most of the semantic content of a sentence
Size	Tend to be small and relatively unchanging	Tend to be large and constantly evolving
Acquisition	Usually learned early in language development	Acquired continuously throughout life as vocabulary expands
Flexibility	Less adaptable to new contexts and meanings	Highly adaptable to new contexts and meanings
Change over time	Relatively stable over time	More susceptible to change over time, reflecting cultural and technological shifts
Syntactic role	Often function as grammatical markers or function words	Often function as content words, carrying the main meaning of a sentence
Examples in a sentence	"The **cat** sat **on** the **mat and looked** at **the bird**." (closed class words in bold)	"The **fluffy kitten snoozed peacefully beneath** the **warm sunbeam**." (open class words in bold)

Explain regular expression in Natural language processing.

- Regular expressions is defined as a sequence of characters that are mainly used to find or replace patterns present in the text. In simple words, we can say that a regular expression is a set of characters or a pattern that is used to find substrings in a given string.
- A regular expression (RE) is a language for specifying text search strings. It helps us to match or extract other strings or sets of strings, with the help of a specialized syntax present in a pattern.
- **For Example,** extracting all hashtags from a tweet, getting email iD or phone numbers, etc from large unstructured text content.

Mathematically, we can define the concept of Regular Expression in the following manner:

1. ε is a Regular Expression, which indicates that the language is having an empty string.

2. φ is a Regular Expression which denotes that it is an empty language.

3. If X and Y are Regular Expressions, then the following expressions are also regular.

- X, Y
- X.Y(Concatenation of XY)
- X+Y (Union of X and Y)
- X*, Y* (Kleen Closure of X and Y)

4. If a string is derived from the above rules then that would also be a regular expression.

How can Regular Expressions be used in NLP

1. **Text Cleaning and Preprocessing:**
   - **Pattern Matching:** Regex excels at identifying and extracting specific patterns from text, such as:
     - URLs (e.g., `https?://[^\s]+`)
     - Email addresses (e.g., `\b[A-Z0-9._%+-]+@[A-Z0-9.-]+\.[A-Z]{2,}\b`)
     - Phone numbers (e.g., `\d{3}-\d{3}-\d{4}`)
     - Dates and times (e.g., `\d{4}-\d{2}-\d{2}`)
     - Numbers (e.g., `\d+`)
     - Currency symbols (e.g., `\$\d+`)
   - **Tokenization:** Regex can split text into words or sentences using delimiters like spaces, punctuation, or custom patterns.
   - **Normalization:** It can standardize text by removing extra whitespace, converting to lowercase, handling special characters, etc.
2. **Information Extraction:**
   - **Named Entity Recognition (NER):** Regex can be used to detect and extract named entities (people, organizations, locations) using patterns or in combination with machine learning models.
   - **Relation Extraction:** It can help identify relationships between entities by finding patterns that connect them in text.
3. **Text Classification:**
   - **Feature Engineering:** Regex can create features for text classification models by extracting patterns of words or characters indicative of specific classes.
4. **Parsing and Analysis:**
   - **Sentence Splitting:** It can divide text into sentences based on punctuation rules or custom patterns.
   - **Morphological Analysis:** Regex can assist in identifying morphemes (word parts) in conjunction with other techniques.
5. **Other NLP Tasks:**
   - **Sentiment Analysis:** Regex can help detect sentiment-bearing words or phrases.
   - **Machine Translation:** It can aid in identifying patterns for translation rules.
   - **Text Summarization:** Regex can assist in identifying important sentences or phrases.

What are basic regular expression patterns? Give brief answer for each with example.

In natural language processing (NLP), regular expressions (regex) are often used for pattern matching. Here are some basic regular patterns with brief explanations and examples:

1. **Literal Characters:**
   - **Pattern:** Matches the exact characters specified.
   - **Example:** `cat` matches "cat" in "The cat is cute."
2. **Character Classes:**
   - **Pattern:** Matches any one of a set of characters.
   - **Example:** `[aeiou]` matches any vowel in "Hello."

3. **Quantifiers:**
    - **Pattern:** Specifies the number of occurrences of the preceding element.
    - **Example:** `ab{2,4}` matches "abb," "abbb," and "abbbb."
4. **Wildcard (dot):**
    - **Pattern:** Matches any single character except a newline.
    - **Example:** `c.t` matches "cat," "cut," etc. in "The cat is cute."
5. **Anchors:**
    - **Pattern:** Specifies the position in the string where a match must occur.
    - **Example:** `^Start` matches "Start of the line."
6. **Alternation:**
    - **Pattern:** Matches one of several patterns.
    - **Example:** `cat|dog` matches "cat" or "dog" in "I have a cat and a dog."
7. **Grouping:**
    - **Pattern:** Groups multiple tokens together.
    - **Example:** `(ab)+` matches "ab," "abab," etc. in "abcabab."
8. **Escape Characters:**
    - **Pattern:** Escapes a special character, treating it as a literal.
    - **Example:** `\\d` matches any digit in "The price is $10."
9. **Word Boundaries:**
    - **Pattern:** Matches the position between a word character (as defined by `\\w`) and a non-word character.
    - **Example:** `\\bword\\b` matches "word" but not "subword" in "This is the word."
10. **Character Range:**
    - **Pattern:** Specifies a range of characters.
    - **Example:** `[0-9]` matches any digit in "The year is 2022."

What is language model? Write a note on N-Gram language model. <V.V.V.IMP>

- A language model is a computational model designed to understand, generate, and predict human language.
- It learns the statistical and structural patterns of a given language by analyzing large amounts of text data.
- The fundamental goal of a language model is to capture the inherent rules, dependencies, and semantics present in natural language.
- These models play a crucial role in various natural language processing tasks, including text generation, machine translation, sentiment analysis, and more.

N-Gram Language Model:

The N-Gram language model is a statistical language model that calculates the probability of a word based on the previous N-1 words in a sequence. It is a simplified approach to language modeling that provides a foundation for understanding the sequential nature of language. The term "N-Gram" refers to a contiguous sequence of N items, which, in the case of language modeling, are typically words.

Key Features of N-Gram Language Model:

1. **Order of N:**
   - The order of the N-Gram model, denoted as "N," determines how many previous words are considered to predict the next word. For example, in a bigram (2-gram) model, the probability of a word is conditioned on the preceding word.
2. **Markov Assumption:**
   - N-Gram models rely on the Markov assumption, which assumes that the probability of the next word depends only on the previous N-1 words and is independent of the words that came before. This simplifies the modeling process.
3. **Probability Calculation:**
   - The probability of a word sequence is calculated using the chain rule of probability. For a trigram (3-gram) model, the probability of a sequence of words \(W\) is calculated as:
     \[P(W) = P(w_1) \cdot P(w_2 | w_1) \cdot P(w_3 | w_1, w_2) \cdot \ldots \]
4. **Parameter Estimation:**
   - Parameters, or probabilities, in N-Gram models are estimated from a training corpus by counting the occurrences of N-grams. Smoothing techniques may be applied to handle unseen N-grams and avoid zero probabilities.
5. **Language Modeling Applications:**
   - N-Gram language models are widely used for tasks such as text generation, spell checking, and machine translation. They serve as the basis for more advanced language models, including neural language models.

Example:

Consider the bigram "2-gram" model:

- **Training Corpus:**
  - "The cat is on the mat. The dog is on the rug."
- **Probability Calculation:**

$$P(\text{"cat"} \mid \text{"The"}) = \frac{\text{Count("The cat")}}{\text{Count("The")}}$$
$$P(\text{"is"} \mid \text{"cat"}) = \frac{\text{Count("cat is")}}{\text{Count("cat")}}$$

- **Prediction:**
  - Given the context "The cat," the model predicts the next word based on the probability distribution of words that follow "The cat" in the training corpus.

Advantages of n-gram models:

- **Captures local context:** By considering sequences of words, n-grams can capture the meaning and context of language that single words often cannot. This makes them effective for tasks like language modeling, speech recognition, and machine translation.
- **Simple and efficient:** The concept of n-grams is straightforward and the algorithms involved are relatively simple to implement. This makes them computationally efficient and attractive for resource-constrained applications.
- **Effective for smaller datasets:** When dealing with limited training data, n-grams can still be quite effective in capturing statistical relationships between words. This can be beneficial for specific tasks where larger datasets are unavailable.
- **Interpretability:** Compared to more complex models, n-grams offer a certain degree of interpretability. Analyzing the n-grams present in a text can provide insights into the topic and language use.

Disadvantages of n-gram models:

- **Data sparsity:** As the n-gram order increases, the number of possible n-grams also grows exponentially. This can lead to data sparsity, where many n-grams rarely or never appear in the training data. This can result in unreliable predictions for unseen n-grams.
- **Limited long-range dependencies:** N-grams focus on local context and typically have difficulty capturing long-range dependencies between words in a sentence. This can limit their effectiveness for tasks requiring understanding complex sentence structures.
- **Overfitting:** N-gram models trained on small datasets are prone to overfitting, where they memorize the training data but fail to generalize well to unseen examples. This can lead to inaccurate predictions on new data.
- **Scalability issues:** N-gram models can become computationally expensive and cumbersome to handle with large datasets or high n-gram orders. This can limit their applicability for massive text analysis tasks.
- **Ignoring word order:** Some n-gram models treat word order within the n-gram as irrelevant, which can lead to inaccurate predictions and limit their ability to capture complex grammatical structures.

Explain how N-Gram model is used in spelling correction.

N-Gram models are commonly used in spelling correction tasks to identify and correct misspelled words in a given text. The basic idea behind using N-Gram models for spelling correction is to leverage the probability distribution of word sequences to estimate the likelihood of a given word being spelled correctly based on its context. Here's how N-Gram models are applied in spelling correction:

1. **Training the N-Gram Model:**
    - The first step is to train an N-Gram language model on a large corpus of correctly spelled text. The model is typically trained with bigrams (2-grams) or trigrams (3-grams), where the probability of a word is conditioned on the previous one or two words.
2. **Calculating Probabilities:**
    - For each word in the training corpus, the N-Gram model calculates the probability of that word given its context (preceding words). This is done by counting the occurrences of word sequences in the training data and normalizing the counts to obtain probabilities.
3. **Building a Probability Distribution:**
    - The N-Gram model builds a probability distribution for each possible word given its context. For example, if the context is "The cat," the model calculates the probabilities of different words that often follow "The cat" in the training data.
4. **Identifying Candidates:**
    - When a user enters a potentially misspelled word, the N-Gram model can suggest correction candidates based on the context in which the word appears. The model identifies words that are probable in that context and ranks them according to their likelihood.
5. **Ranking and Selection:**
    - The correction candidates are ranked based on their probabilities, and the most probable candidate is selected as the suggested correction. This ranking helps prioritize corrections that are more likely to be correct in the given context.
6. **Smoothing Techniques:**
    - To handle unseen or infrequent N-Grams, smoothing techniques are often applied. Smoothing methods help prevent zero probabilities for word sequences not present in the training data, improving the model's generalization.

Example:

Consider the following sentence with a potential misspelling:

- Original Sentence: "The cat sat on the mta."

Assuming a bigram model, the correction process might proceed as follows:

- For the bigram "on the," the model suggests probable words like "mat".
- The model calculates probabilities for each candidate based on the context "on the" and selects the most likely correction.

The corrected sentence might be: "The cat sat on the mat."

By leveraging the contextual information provided by N-Gram models, spelling correction systems can offer accurate suggestions for correcting misspelled words, especially in the context of adjacent words.
