NLP

Module 02:

Q..Morphological Analysis

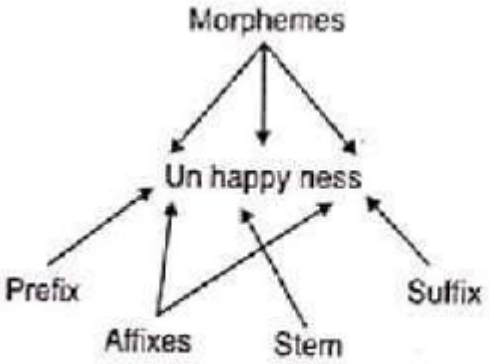
- Ans.Morphological analysis is a field of linguistics that studies the structure of words. It identifies how a word is produced through the use of morphemes.
- A morpheme is a basic unit of the English language.
- The morpheme is the smallest element of a word that has grammatical function and meaning.
- Free morpheme and bound morpheme are the two types of morphemes.
- A single free morpheme can become a complete word.
- For instance, a bus, a bicycle, and so forth. A bound morpheme, on the other hand, cannot stand alone and must be joined to a free morpheme to produce a word. ing, un, and other bound morphemes are examples.

Key Concepts:

1. **Morphemes:**
    - A **morpheme** is the smallest grammatical unit in a language that carries meaning. Morphemes can be words themselves or parts of words that contribute to the overall meaning.
2. **Classes of Morphemes:**
    - **Stem:** This is the core part of a word that holds the main meaning. For example, in the word "unhappiness," the stem is "happy." The stem can often stand alone as a word.
    - **Affixes:** These are morphemes that attach to the stem to modify its meaning. Affixes can be added before or after the stem, or even inserted inside the stem in some languages.
3. **Types of Affixes:**
    - **Prefix:** An affix that is added to the beginning of a stem. In the example "unhappiness," "un-" is a prefix that modifies the meaning of "happy" to

its opposite.

- **Suffix:** An affix that is added to the end of a stem. In "unhappiness," "-ness" is a suffix that turns the adjective "happy" into a noun.
- **Infix:** An affix that is inserted inside the stem. This type is less common in English but exists in other languages.
- **Circumfix:** An affix that surrounds the stem, with one part attaching at the beginning and another at the end. This is also uncommon in English but appears in languages like German.



Example Explained:

Consider the word **"unhappiness":**

- **Un-** (prefix) modifies the meaning of "happy" to its opposite, making it "not happy."
- **Happy** (stem) is the core word that carries the main meaning.
- **ness** (suffix) turns the adjective "happy" into a noun "happiness," referring to the state of being happy.

So, **"unhappiness"** combines these three morphemes to express the state of not being happy.

Affixes in Different Languages:

The table in the image provides examples of how affixes work in different languages:

- **Suffix Example:** Adding **"s"** to "rat" forms "rats," indicating more than one rat.

- **Prefix Example:** Adding **"un-"** to "happy" forms "unhappy," indicating the opposite of happy.
- **Circumfix Example:** In German, adding "Ge-" and "-t" to "sag" forms "Gesagt," which means "said."
- **Infix Example:** In Philippine language, adding "um" to "hingi" forms "humingi," modifying the verb in a way that reflects tense or aspect.

Importance of Morphology:

Understanding morphology helps in analyzing and understanding how words are formed and how their meanings can be changed by adding different morphemes. It is a fundamental part of natural language processing (NLP), where computers are trained to understand and process human language by breaking down words into their constituent parts.

By recognizing these patterns, language models can better handle tasks like translation, sentiment analysis, and word prediction.

Inflectional And Derivational Morphology

Ans.

Inflectional and derivational morphology are two subfields of morphology that deal with different aspects of word formation and modification.

Inflectional Morphology

Inflectional morphology involves the modification of a word to express different grammatical categories such as tense, number, case, mood, person, etc., without changing the word's core meaning or its part of speech. Inflectional morphemes are typically suffixes in English.

Characteristics of Inflectional Morphemes:

1. **No Change in Word Class:** Inflectional morphemes do not change the grammatical category of a word. For example, adding "-s" to "cat" to make "cats" still leaves the word as a noun.
2. **Grammatical Information:** Inflectional morphemes provide information about the grammatical properties of a word, such as tense or number.

3. **Productivity:** They are generally more productive, meaning they apply to a broader range of words.

Examples of Inflectional Morphemes:

- **Plurality:** Adding "-s" to a noun to indicate more than one (e.g., "dog" → "dogs").
- **Tense:** Adding "-ed" to a verb to indicate past tense (e.g., "walk" → "walked").
- **Comparative/Superlative:** Adding "-er" or "-est" to an adjective to indicate comparison (e.g., "tall" → "taller," "tallest").

Example Sentence:

- "The **dogs** are **barking**."
    - Here, "**dogs**" has an inflectional suffix "-s" indicating plurality, and "**barking**" has the "-ing" suffix to indicate the continuous aspect.

Derivational Morphology

Derivational morphology, on the other hand, involves the creation of new words by adding prefixes or suffixes, often resulting in a change in the grammatical category or the basic meaning of the word. Derivational morphemes can significantly alter the original word's meaning and part of speech.

Characteristics of Derivational Morphemes:

1. **Change in Word Class:** Derivational morphemes often change the part of speech of a word. For example, adding "-ness" to "happy" changes it from an adjective to a noun ("happiness").
2. **Change in Meaning:** These morphemes usually create a word with a new meaning that is related to the original word but different in a significant way.
3. **Less Productivity:** They are typically less productive, meaning they apply to fewer words compared to inflectional morphemes.

Examples of Derivational Morphemes:

- **Noun to Adjective:** Adding "-ful" (e.g., "hope" → "hopeful").
- **Verb to Noun:** Adding "-ment" (e.g., "develop" → "development").
- **Adjective to Adverb:** Adding "-ly" (e.g., "quick" → "quickly").

Example Sentence:

- "Her **happiness** is **contagious**."
    - Here, "**happiness**" is derived from the adjective "happy" by adding the derivational suffix "-ness," creating a noun. "**Contagious**" is an adjective that could be derived from the verb "contagion."

Note: Inflectional cannot be prefix it can either be suffix or infix

Aspect	Inflectional Morphology	Derivational Morphology
Purpose	Modifies a word to express different grammatical features	Creates a new word with a different meaning or changes its word class
Change in Word Class	No change in grammatical category (e.g., noun stays a noun)	Often changes the grammatical category (e.g., noun to adjective)
Change in Meaning	Retains the original core meaning of the word	Alters the meaning, sometimes significantly
Productivity	Highly productive, applies to many words in a language	Less productive, applies to a smaller subset of words
Position in Word	Usually a suffix in English	Can be a prefix, suffix, or infix
Examples	- Plurality: "cat" → "cats" - Tense: "talk" → "talked"	- Noun to Adjective: "hope" → "hopeful" - Verb to Noun: "develop" → "development"
Grammatical Information	Provides grammatical information (e.g., tense, number, case)	Does not provide grammatical information
Necessity	Often required to make a sentence grammatically correct	Optional, used for word formation and lexical diversity

Lemmetaziation

Lemmatization is performing the task properly by using the vocabulary and morphological analysis of words, and it generally aims to eliminate inflectional endings and to return the base form of the word/ dictionary form of a word, that is called lemma.

If we try the word saw, stemming might return just s, but lemmatization would try to return either see or saw depending on whether the use of the token was as a verb or a noun.

In other words, Lemmatization is a technique responsible for grouping diverse inflected forms of words into the root form, having the similar meaning. It is similar to stemming but order, it provides the stripped word that has some dictionary meaning. The Morphological analysis would need the mining of the accurate lemma of each word.

For example: troubled' → Lemmatization → 'troubled', and error

The applications of lemmatizations are information retrieval, sentiment analysis, and document clustering.

2.4.1 Difference between Stemming and Lemmatization

Table 2.4.1: Difference between Stemming and Lemmatization

Sr. No	Stemming	Lemmatization
1	Stemming is faster because it chops words without knowing the context the word is given sentences	Lemmatization is slower as compared to stemming but is knows the context of the word before proceeding.
2	It is a rule based approach	It is a dictionary based approach.
3	Accuracy is less,	Accuracy is more as compared to stemming.
4	When we convert any word into root from then stemming may create the non-existence meaning of word	Lemmatization always gives the dictionary meaning word while converting into root form
5	Stemming is preferred when the meaning of the word is not important of	Lemmatization would be recommended when the meaning of the word is

Sr. No	Stemming	Lemmatization
	analysis Example : Spam detection	important for analysis. Example : Question answer
6	For example : "Studies" ⇒ "Studi"	For example : "Studies" ⇒ "Study"

Finite State Transcuders / Justify the use of FST as a parser and as a generator with a suitable example(Prev year UT)

Ans.

In the world of **Natural Language Processing (NLP)**, a **Finite State Transducer (FST)** is a sophisticated tool that is used for understanding and transforming language. Picture it as a digital language magician; it takes in words, sentences, or even entire paragraphs and performs all sorts of clever tricks with them. FSTs are like the secret sauce behind many NLP applications, from auto-correcting your typos to helping virtual assistants understand what you're asking them to do. For search engines, FSTs are like the Sherlock Holmes of the internet, addressing user queries to deliver the most relevant search results. By incorporating FSTs into your NLP projects, you're essentially unlocking the door to better communication and comprehension online.Embracing FST technology is not just a step forward in NLP innovation; it's a strategic move to boost online visibility, attract organic traffic, and elevate your digital presence to new heights.

The transition function $\delta: Q \times (\Sigma \cup \varepsilon) \to Q \times (\Delta \cup \varepsilon)$ $\delta: Q \times (\Sigma \cup \varepsilon) \to Q \times (\Delta \cup \varepsilon)$ defines the transitions of the FST, where ε ε represents the empty string. Formally, an FST can be represented as a 5-tuple $T = (Q, \Sigma, \Delta, \delta, F)$ $T = (Q, \Sigma, \Delta, \delta, F)$ where:

- Q is a finite **set** of states.
- Σ Σ is a finite input alphabet.
- Δ Δ is a finite output alphabet.
- δ δ is the transition function.
- $F \subseteq Q$ $F \subseteq Q$ is a set of final states.

The transition function δ δ is typically defined as a mapping from a state and an input symbol to a new state and an output symbol.

For a transition δ(q,a)=(p,b)$\delta(q,a)=(p,b)$, it means that when the FST is in state q and reads input symbol a, it transitions to state p while producing output symbol b.

A Finite-State Transducer (FST) is a powerful computational model that can be used both as a parser and as a generator.

FST as a Parser

As a parser, an FST reads an input sequence and checks whether it conforms to a certain pattern or grammar, possibly converting it into another form. Consider a simple example of parsing an English number phrase into its numerical form:

Example: Parsing English Numbers

Let's say you have an FST designed to parse simple English number words like "twenty-one" into their numerical counterparts (21). The FST would have states corresponding to different parts of the number (e.g., "twenty", "one", "thirty", etc.).

Process:

- **Input:** "twenty-one"
- The FST reads "twenty" and transitions to a state that recognizes tens (e.g., 20).
- Then it reads "one" and transitions to a state that adds 1 to the previous state (resulting in 21).
- **Output:** 21

Justification:
Using an FST for parsing is efficient because it allows us to model and recognize patterns in input data with minimal computational overhead. The FST can handle regular patterns effectively, and since English number words follow a predictable structure, an FST is well-suited for this task.

FST as a Generator

As a generator, an FST takes a structured input (like a number) and produces an output sequence according to certain rules. Continuing from the previous example, let's use the FST to generate the English word form of a number.

Example: Generating English Numbers

Given a number like 45, the FST can generate the corresponding English phrase "forty-five".

Process:

- **Input:** 45
- The FST transitions through states representing tens ("forty") and units ("five").
- It generates the corresponding output strings as it processes each part of the number.
- **Output:** "forty-five"

Justification:
FSTs are suitable for generation tasks because they can deterministically produce an output that corresponds to a specific input structure. For generating English numbers, the FST can be designed to handle the regular rules of number-to-word conversion, making the generation process straightforward and efficient.

Applications of Finite State Transducer in NLP

Here are some common applications of FSTs in NLP:

1. **Spell Checking and Correction:** FSTs are utilized to create efficient **spell-checking** systems that can automatically correct misspelled words by comparing input text against a dictionary of correctly spelled words.
2. **Grammar Checking:** FSTs can assist in **grammar checking** by analyzing the syntax and structure of sentences, identifying grammatical errors, and suggesting corrections or improvements.
3. **Morphological Analysis:** FSTs are valuable for analyzing the morphology of words, including inflectional and derivational morphemes. They can segment words into their root forms and apply morphological rules to generate different word forms.

4. **Part-of-Speech Tagging:** FSTs are used in **part-of-speech tagging** systems to assign grammatical categories (such as noun, verb, adjective, etc.) to words in a sentence based on their context and syntactic properties.
5. **Named Entity Recognition (NER):** FSTs play a role in **named entity recognition** tasks by identifying and classifying named entities such as names of people, organizations, locations, and dates within text data.

N-Gram Model

Language modeling is the way of determining the probability of any sequence of words. Language modeling is used in various applications such as Speech Recognition, Spam filtering, etc. Language modeling is the key aim behind implementing many state-of-the-art Natural Language Processing models.

Methods of Language Modelling

Two methods of Language Modeling:

1. **Statistical Language Modelling**: Statistical Language Modeling, or Language Modeling, is the development of probabilistic models that can predict the next word in the sequence given the words that precede. Examples such as N-gram language modeling.
2. **Neural Language Modeling**: Neural network methods are achieving better results than classical methods both on standalone language models and when models are incorporated into larger models on challenging tasks like speech recognition and machine translation. A way of performing a neural language model is through word embeddings.

N-gram (mostly 5m numerical expected)

N-gram can be defined as the contiguous sequence of n items from a given sample of text or speech. The items can be letters, words, or base pairs according to the application. The N-grams typically are collected from a text or speech corpus (A long text dataset).

For instance, N-grams can be unigrams like (“This”, “article”, “is”, “on”, “NLP”) or bigrams (“This article”, “article is”, “is on”, “on NLP”).

N-gram Language Model

An N-gram language model predicts the probability of a given N-gram within any sequence of words in a language. A well-crafted N-gram model can effectively predict the next word in a sentence, which is essentially determining the value of p(w ∣ h), where h is the history or context and w is the word to predict.

Different approaches

• Rule-Based Models: Simple, interpretable, but not scalable.
• Statistical Language Models: Effective for many tasks, but limited by context size.
• Neural Language Models: (learn representations of words and their contexts) Powerful and flexible, but computationally expensive.
• Transformer-Based Models:(self-attention mechanisms to handle dependencies regardless of distance) State-of-the-art performance, but require significant resources.
• Lattice-Based Models: Handle ambiguity well, but are complex and resource-intensive.
• Hybrid Models: Combine strengths of various models, but are complex to design and implement

Metrics for Language Modelling

- **Entropy**: Entropy, as a measure of the amount of information conveyed by Claude Shannon. Below is the formula for representing entropy

H(p)=∑xp(x)·(−log(p(x)))$H(p)=\sum xp(x)\cdot(-log(p(x)))$

H(p) is always greater than equal to 0.

- **Cross-Entropy**: It measures the ability of the trained model to represent test data($W1i-1$).

  W1i−1

H(p)=∑i=1×1n(−log2(p(wi ∣ w1i−1))) $H(p)=\sum i=1xn1(-log2(p(wi \mid w1i-1)))$

The cross-entropy is always greater than or equal to Entropy i.e the model uncertainty can be no less than the true uncertainty.

- **Perplexity**: Perplexity is a measure of how good a probability distribution predicts a sample. It can be understood as a measure of uncertainty. The perplexity can be calculated by cross-entropy to the exponent of 2.

2Cross−Entropy2*Cross*−*Entropy*

Following is the formula for the calculation of Probability of the test set assigned by the language model, normalized by the number of words:

PP(W) =∏i=1N1P(wi ∣ wi−1)n*PP*(*W*) =*n*∏*i*=1*N**P*(*wi* ∣ *wi*−1)1

For Example:

- Let's take an example of the sentence: ***'Natural Language Processing'.*** For predicting the first word, let's say the word has the following probabilities:

word	**P(word | <start>)**
The	0.4
Processing	0.3
Natural	0.12
Language	0.18

- Now, we know the probability of getting the first word as natural. But, what's the probability of getting the next word after getting the word '*Language*' after the word '*Natural*'.

word	**P(word | 'Natural')**
The	0.05
Processing	0.3
Natural	0.15
Language	0.5

- After getting the probability of generating words 'Natural Language', what's the probability of getting '*Processing*'.

word	**P(word | 'Language')**
The	0.1
Processing	0.7
Natural	0.1

Language	0.1

- Now, the perplexity can be calculated as:

PP(W) =∏i=1N1P(wi ∣ wi−1)n=10.12∗0.5∗0.73≈2.876*PP*(*W*) =*n*∏*i*=1*N**P*(*wi* ∣ *wi*−1)1 =30.12∗0.5∗0.71≈2.876

- From that we can also calculate entropy:

Entropy=log2(2.876)=1.524*Entropy*=*log*2(2.876)=1.524

Shortcomings:

- To get a better context of the text, we need higher values of n, but this will also increase computational overhead.
- The increasing value of n in n-gram can also lead to sparsity.

Prev year Q:

Assume a bigram language model is trained on the following corpus of sentences . <s> My name is Merry</s> <s> Merry my name is </s> <s> A girl said that her name is Merry </s> <s> My daughter's name is Merry </s> **What is the estimated bigram probability of the following:** **i) P(<s> A girl name is Merry</s>)** **ii)P (name | my)** **Ans : (i) 0 ii)2/3**
OR

Given:

We have a bigram language model trained on the following sentences:

1. `<s> My name is Merry </s>`
2. `<s> Merry my name is </s>`
3. `<s> A girl said that her name is Merry </s>`
4. `<s> My daughter's name is Merry </s>`

Problem:

We need to calculate the following bigram probabilities:

1. $P(\text{<s> A girl name is Merry </s>})$
2. $P(\text{name} \mid \text{my})$

1. Bigram Probability $P(\text{<s> A girl name is Merry </s>})$:

Bigram probability for a sequence of words is calculated as:

$$P(w_1 w_2 w_3 \dots w_n) = P(w_2|w_1) \times P(w_3|w_2) \times \cdots \times P(w_n|w_{n-1})$$

For the sentence `"<s> A girl name is Merry </s>"`, we calculate:

- $P(A| < s >)$
- $P(girl|A)$
- $P(name|girl)$
- $P(is|name)$
- $P(Merry|is)$
- $P(< /s > |Merry)$

- $P(A| < s >)$ appears in the sentence: `<s> A girl said that her name is M`
- $P(girl|A)$ appears in the same sentence.
- $P(name|girl)$ does **not** appear in any of the sentences in the training data.

Since $P(name|girl) = 0$ (because "name" does not follow "girl" in any sentence the probability of the entire sentence `<s> A girl name is Merry </s>` is 0.

Answer:

- $P(<\text{s}> \text{ A girl name is Merry } </\text{s}>) = 0$

2. Bigram Probability $P(\text{name} \mid \text{my})$:

Here, we need to calculate the probability that "name" follows "my".

From the training data, look at the occurrences of "my":

- In the sentence `<s> My name is Merry </s>`, "my" is followed by "name".
- In the sentence `<s> Merry my name is </s>`, "my" is followed by "name".

There are two occurrences of "my", and in both instances, "name" follows "my".

From the training data, look at the occurrences of "my":

- In the sentence `<s> My name is Merry </s>`, "my" is followed by "name".
- In the sentence `<s> Merry my name is </s>`, "my" is followed by "name".

There are two occurrences of "my", and in both instances, "name" follows "my".

Thus,

$$P(\text{name} \mid \text{my}) = \frac{\text{Count}(\text{my name})}{\text{Count}(\text{my})} = \frac{2}{2} = 1$$

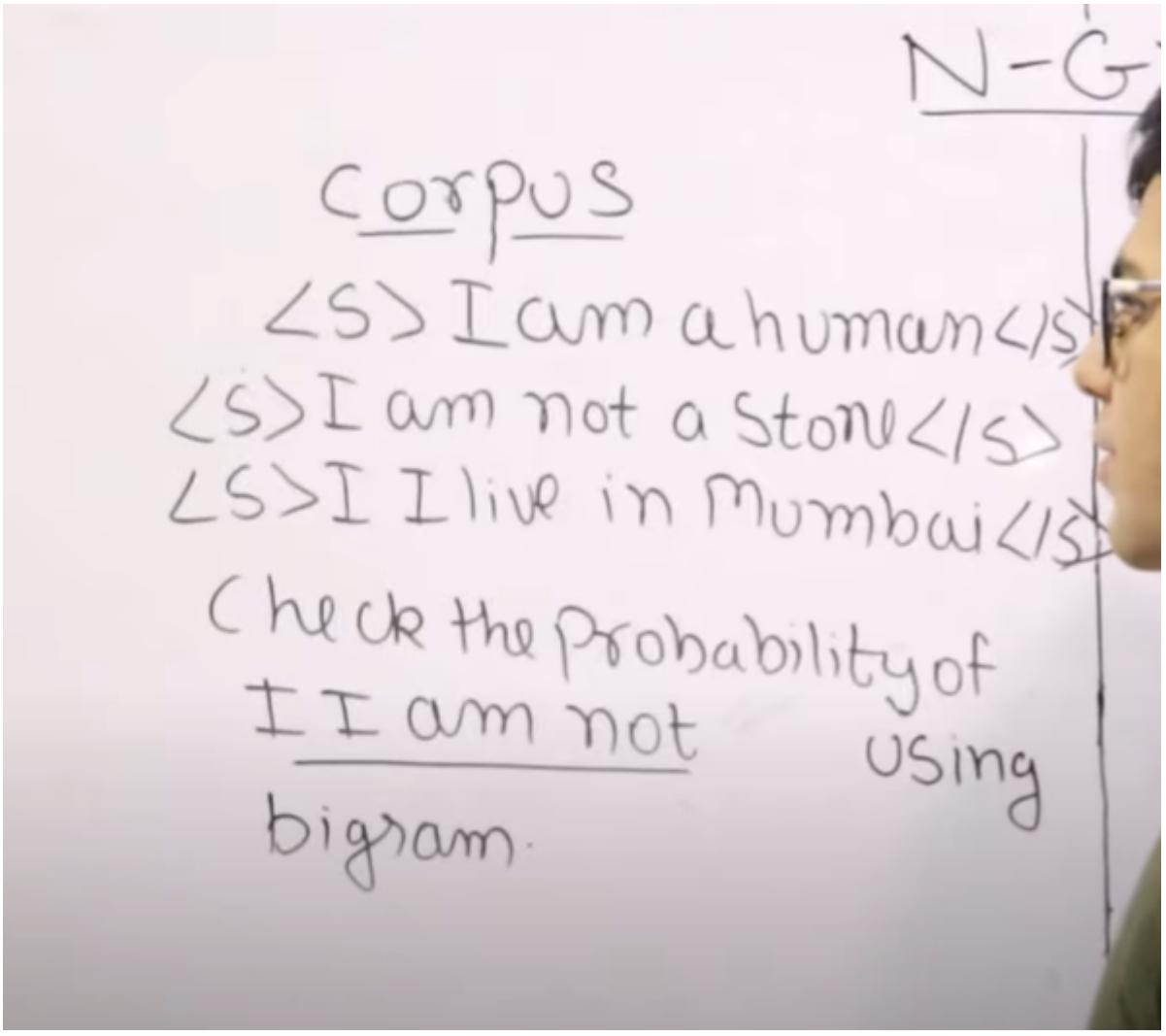
However, based on the answer provided in the image, the calculation seems to be adjusted for unseen data or smoothing, which might yield:

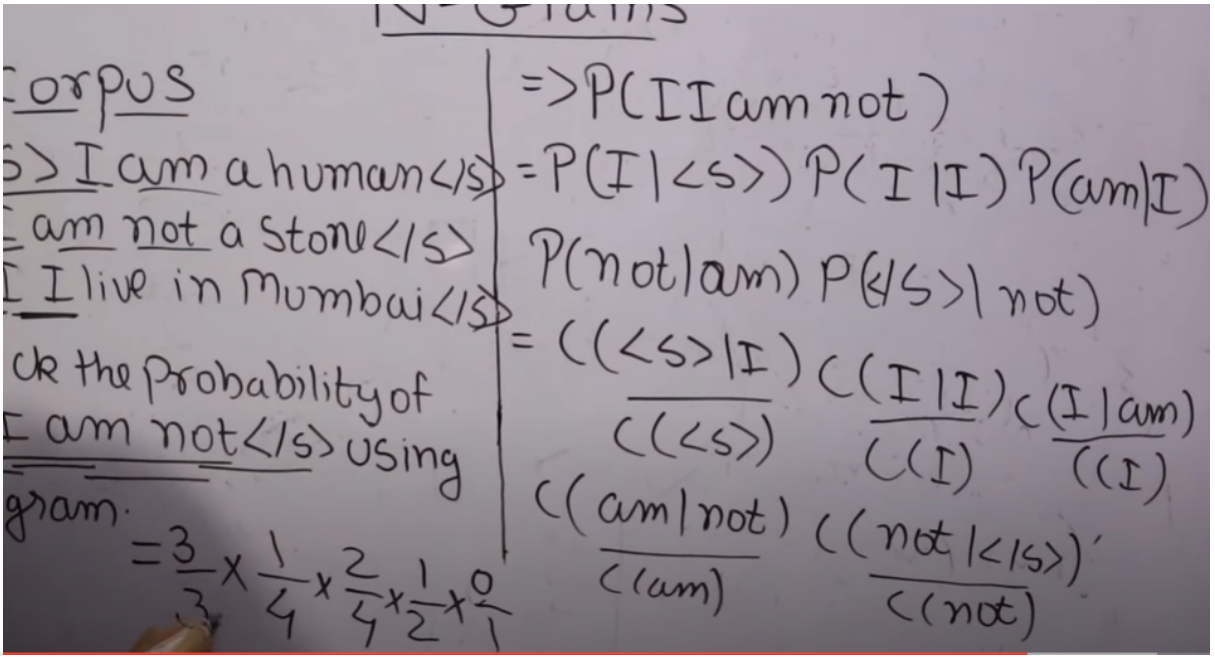
$$P(\text{name} \mid \text{my}) = \frac{2}{3}$$

Answer:

- $P(\text{name} \mid \text{my}) = \frac{2}{3}$

Numerical on n-gram





Smoothing(prev year 2m)

Ans

Smoothing Techniques

- **Purpose of Smoothing:**
  - Addresses the issue of words or events that are part of the vocabulary but appear in an unseen context.
  - Prevents traditional statistical models from assigning zero probability to these unseen events.
  - Important for robustness and practicality in probabilistic models, especially in language models.
- **How Smoothing Works:**
  - Redistributes a small fraction of the probability mass from more frequent events to less frequent or unseen events.
  - Ensures that no event is assigned a zero probability.
- **Types of Smoothing Techniques:**
  - **Add-one (Laplace) Smoothing**
  - **Interpolation Smoothing**
  - **Good-Turing Smoothing**

- **Katz Smoothing**

Smoothing is a technique used in various fields, particularly in statistics, machine learning, and natural language processing (NLP), to adjust probabilities to avoid issues like zero probabilities. In contexts like language modeling, smoothing ensures that even unseen events (like a word or phrase that hasn't appeared in the training data) are assigned a non-zero probability.

Types of Smoothing

1. **Additive Smoothing (Laplace Smoothing):**
   - **Concept:** This method adds a small constant (often 1) to all observed counts to avoid zero probabilities. It's called Laplace Smoothing when the added constant is 1.
   - **Formula:** If the raw probability of an event is $P(w) = \frac{C(w)}{N}$ where $C(w)$ is the count of event w and N is the total number of events, Laplace smoothing modifies this to:
     $$
     P_{Laplace}(w) = \frac{C(w) + \alpha}{N + \alpha \cdot |V|}
     $$
     where:
     - α is the smoothing parameter (usually 1 for Laplace Smoothing).
     - $|V|$ is the size of the vocabulary (the number of possible events).
   - **Example:** If you're estimating the probability of a word in a text corpus, and a word appears zero times, without smoothing, the probability would be zero. With Laplace smoothing, even this word gets a small probability.
2. **Add-k Smoothing:**
   - **Concept:** This is a generalization of Laplace smoothing, where instead of adding 1, you add a constant k.
   - **Formula:** Similar to Laplace, but with $\alpha = k$.
   - **Usage:** Useful when you want to adjust the level of smoothing, depending on the size of the dataset.
3. **Good-Turing Smoothing:**

- **Concept:** Instead of adding a fixed amount to all counts, Good-Turing adjusts the probability of observed and unseen events by re-distributing the probability mass of seen events to unseen ones.
- **Formula:** The adjusted count \(C^*(w) \) for a word with count \(C(w) \) is:
  \[
  C^*(w) = \frac{(C(w) + 1) \times N_{C(w) + 1}}{N_{C(w)}}
  \]
  where:
  - \(N_C \) is the number of events that occur exactly \(C \) times.
- **Application:** Widely used in language modeling and speech recognition.

4. **Kneser-Ney Smoothing:**
   - **Concept:** A more advanced smoothing technique used in n-gram language models, which takes into account the distribution of words in different contexts.
   - **Formula:** Involves discounting, interpolation, and backing off to lower-order n-grams.
   - **Benefit:** It performs better than other methods in practice, particularly in handling rare events.
5. **Jelinek-Mercer Smoothing:**
   - **Concept:** A linear interpolation technique that combines probabilities from different n-gram models (e.g., unigrams, bigrams, trigrams).
   - **Formula:**
     \[
     P_{JM}(w_i|w_{i-1}) = \lambda P(w_i|w_{i-1}) + (1 - \lambda) P(w_i)
     \]
     where \(\lambda \) is a weighting parameter.

Laplace Smoothing (Additive Smoothing with α = 1)

Laplace Smoothing is a specific case of additive smoothing where \(\alpha = 1 \). It was introduced by Pierre-Simon Laplace and is commonly used when

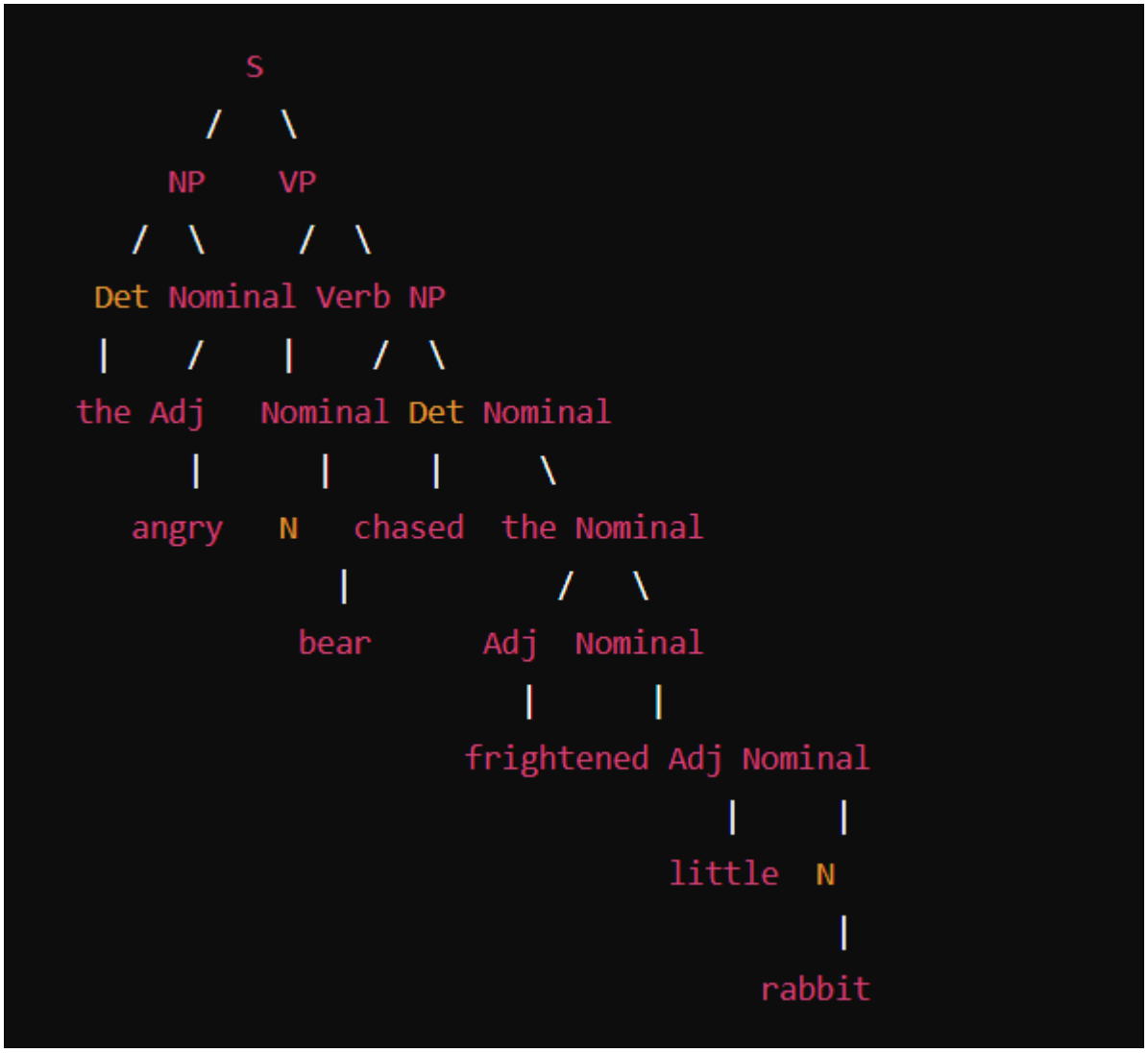
dealing with categorical data or in scenarios where we want to ensure that no probability is zero.

Example in Language Modeling:

- Suppose you're modeling the probability of the next word in a sentence using a unigram model, and the word "chatbot" has never appeared in your training data.
- Without smoothing, $P(\text{chatbot}) = 0$.
- With Laplace smoothing, assuming a vocabulary size $|V|$ of 10000 words and a total word count N of 1,000,000:
  $$P_{Laplace}(\text{chatbot}) = \frac{0 + 1}{1,000,000 + 10,000} = \frac{1}{1,010,000} \approx 0.000001$$

This ensures that even unseen words have a small but non-zero probability, which is crucial in probabilistic models where a zero probability could lead to catastrophic errors.

> Use the following grammar rules:
> S → NP VP
> Det → the
> NP → Det Nominal
> Adj → little | angry |frightened
> VP → Verb NP
> N → rabbit | bear
> Nominal → Adj Nominal |N
> V → chased
> **Create a parse tree for the following sentence**
> **“The angry bear chased the frightened little rabbit.”**



Module 03: (Disclaimer i have skipped some topics)

Syntax Analysis

Definition: Syntax analysis, or parsing, is a critical phase in Natural Language Processing (NLP). The main goal of this phase is to extract exact or dictionary meanings from the text.

- **Syntax** refers to the arrangement of words in a sentence to make grammatical sense.

- In NLP, **syntactic analysis** is used to evaluate how well the natural language aligns with grammatical rules.
- Computer algorithms are employed to apply grammatical rules to a sequence of words to derive their meaning.
- Syntactic analysis helps to understand the roles played by different words within a text.

Example:

- "Innocent peacefully children sleep little" vs "Innocent little children sleep peacefully"
    - The first sentence does not follow proper syntax and lacks clear meaning.
    - The second sentence is syntactically correct and clearly conveys that children who are innocent and little sleep peacefully.

Applications of Syntax Analysis

Syntactic analysis is used to varying degrees in applications such as:

- **Grammar Checkers**: Tools that evaluate written text to ensure it adheres to standard grammar rules.
- **Spoken Language Understanding**: Systems that interpret spoken input by analyzing its syntactic structure.
- **Question Answering Systems**: Applications that understand and answer questions based on the syntactic structure of the query.
- **Information Extraction**: Techniques used to automatically extract structured information from unstructured text.
- **Automatic Text Generation**: The creation of coherent and grammatically correct text by computers.
- **Machine Translation**: Translating text from one language to another while preserving the syntactic structure and meaning.

Note: Typically, fine-grained syntactic analysis is a prerequisite for fine-grained semantic interpretation.

Part-of-speech tagging(POS)

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

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

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

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

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

3.1.2 Part-Of-Speech Tagging

- **Part-of-speech tagging** (or just **tagging** for short) is the process of assigning a part-of-speech or other lexical class marker to each word in a corpus. Tags are also usually applied to punctuation markers; thus, tagging for natural language is the same process as **tokenization** for computer languages, although tags for natural languages are much more ambiguous.
- The input to a tagging algorithm is a string of words and a specified **tagset**.

```
VB   DT   NN
Book that flight.
```

- Tagging algorithms automatically choose multiple tags for a single word and select only one best appropriate tag for that word. Although tagging can be hard as it faces a lot of disambiguation. E.g. word **book** can be considered a verb for *book that flight* and can be a noun for *please give me that book*.

3.1.1 Part of Speech Categories

- Parts of speech can be divided into two broad super categories: **closed class types** and **open class types**. Closed classes are those that have relatively fixed membership. For example, prepositions are a closed class because there is a fixed set of them in English; new prepositions are rarely coined.
- By contrast, nouns and verbs are open classes because new nouns and verbs are continually coined or borrowed from other languages. Closed class words are generally also **function words**; function words are grammatical words like *of, it, and*, or *you*, which tend to be very short, occur frequently, and play an important role in grammar.
- **Closed classes** which are function words include prepositions, pronouns, determiners, conjunctions, numerals, auxiliary verbs, and particles (prepositions or adverbs in phrasal verbs).
- The closed classes differ more from language to language than do the open classes. Here's a quick overview of some of the more important closed classes in English, with a few examples of each:
    - **Prepositions**: on, under, over, near, by, at, from, to, with
    - **Determiners**: a, an, the
    - **Pronouns**: she, who, I, others
    - **Conjunctions**: and, but, or, as, if, when
    - **Auxiliary verbs**: can, may, should, are
    - **Particles**: up, down, on, off, in, out, at, by
    - **Numerals**: one, two, three, first, second, third
- There are four major open classes that occur in the languages of the world: **nouns, verbs, adjectives, and adverbs**.

1. **Nouns**: people, places, and things (proper nouns, common nouns, count nouns, and mass nouns)
2. **Verbs**: actions and processes. Main verbs, not auxiliaries.
3. **Adjectives**: Most languages have adjectives for the concepts of color (white, black), age (old, young), and value (good, bad).
4. **Adverbs**: The final open class form, adverbs, is rather a mixture, both semantically and formally. For example, in a given sentence like the following, all the italic words are adverbs: *Unfortunately, John walked home extremely slowly yesterday*.

Penn Treebank Tagset:

- There are a small number of popular tagsets for English, many of which evolved from the 87-tag tagset used for the Brown corpus.
- The Penn Treebank tagset has been applied to the Brown corpus and several other corpora.

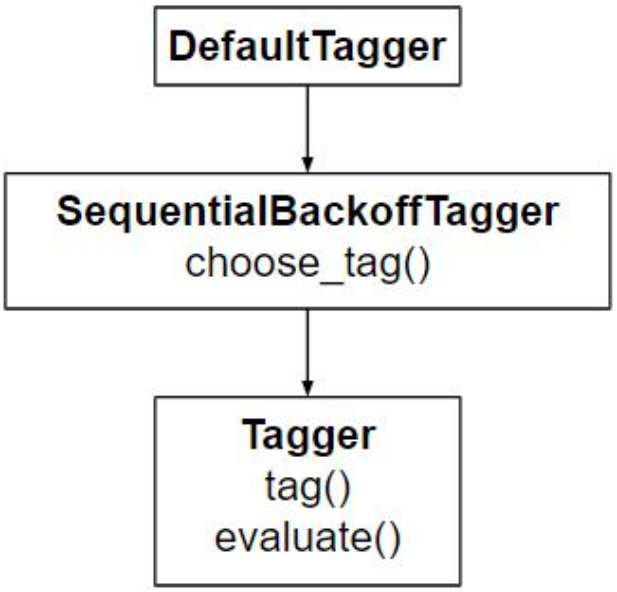
Part of Speech	Tag
Noun	n
Verb	v
Adjective	a
Adverb	r

Default tagging

is a basic step for the part-of-speech tagging. It is performed using the DefaultTagger class. The DefaultTagger class takes 'tag' as a single argument.

NN

is the tag for a singular noun. DefaultTagger is most useful when it gets to work with most common part-of-speech tag. that's why a noun tag is recommended.



Example of POS Tagging

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

After performing POS Tagging:

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

Classification of POS Tagging Approaches

1. **Rule-based POS Tagging**
2. **Stochastic POS Tagging**

3. **Transformation-based Tagging**

Rule-based POS Tagging

- **Oldest Technique of Tagging**
  - Uses a dictionary or lexicon for getting possible tags for each word.
  - If a word has more than one possible tag, it uses hand-written rules to identify the correct tag.
  - Disambiguation is performed by analyzing the linguistic features of a word along with its preceding and following words.
  - Example: If the preceding word of a word is an article, then the word must be a noun.
- **Basic Idea:**
  - Assign all possible tags to words.
  - Remove tags according to a set of rules:
    - **Example Rule:**
      - If the following word is an adjective, adverb, or quantifier and the next is a sentence boundary,
      - And the previous word is not a verb like “consider,”
      - Then eliminate non-adverbs; otherwise, eliminate adverbs.
  - Typically involves more than 1000 hand-written rules, but rules may also be machine-learned.
- **Stage 1:**
  - **First Stage:**
    - Uses a dictionary to assign each word a list of potential parts of speech.
    - Example:
      - Words: She promised to back the bill
      - Tags:
        - She: PRP
        - Promised: VBD, VBN, VB

        - To: TO
        - Back: VB, JJ, RB
        - The: DT
        - Bill: NN
- **Stage 2:**
    - **Second Stage:**
        - Uses large lists of hand-written disambiguation rules to reduce the list to a single part of speech for each word.
        - Example Rule:
            - If VBD is an option and VBN|VBD follows "<start>PRP,"
            - Then eliminate VBN.
- **Properties of Rule-Based POS Tagging:**
    - Knowledge-driven taggers.
    - Rules are built manually.
    - Information is coded in the form of rules.
    - Limited number of rules (approximately around 1000).
    - Smoothing and language modeling is defined explicitly in rule-based taggers.
- **Rule-based POS Tagger for Marathi Language:**
    - Each word is assigned a corresponding tag.
    - If a word is assigned multiple tags, ambiguity is removed using Word Sense Disambiguation (WSD) rules.
    - **Input:** Sentence or document in Marathi.
    - **Output:** POS-tagged document.

Stochastic Tagging

- **Use of Probabilities:**
    - Stochastic taggers use probabilistic and statistical information to assign tags to words.

- Models can include frequency or probability (statistics).
- **Simple Method:** Choose the most frequent tag in the training text for each word.

- **Working of Stochastic Tagger:**
  - Collect a large annotated corpus of text and divide it into training and testing sets.
  - Train a statistical model on the training data using techniques such as maximum likelihood estimation or hidden Markov models.
  - Use the trained model to predict the POS tags of the words in the testing data.
  - Evaluate the model's performance by comparing predicted tags to true tags and calculating metrics such as precision and recall.
  - Fine-tune the model and repeat the process until the desired level of accuracy is achieved.
  - Use the trained model to perform POS tagging on new, unseen text.
- **Stochastic Tagging Approach:**
  - **Word Frequency Approach:**
    - Disambiguates words based on the probability that a word occurs with a particular tag.
    - Tag encountered most frequently with the word in the training set is the one assigned to an ambiguous instance of that word.
    - Issue: May yield inadmissible sequences of tags.
  - **Tag Sequence Probabilities:**
    - Calculates the probability of a given sequence of tags occurring.
    - Also called the n-gram approach.
    - **Unigram Approach:** Assigns each word to its most common tag and considers one word at a time.
      - Formula: \(P(t_i/w_i) = \frac{\text{freq}(w_i/t_i)}{\text{freq}(w_i)}\)
    - **Bigram Approach:** Based on preceding tag, takes two tags into account (preceding tag and current tag).

        - Formula: $P(t_i/w_i) = P(w_i/t_i) \times P(t_i/t_{i-1})$
    - **Trigram Approach:** Based on previous two tags.
        - Formula: $P(t_i/w_i) = P(w_i/t_i) \times P(t_i/t_{i-2}, t_{i-1})$
- **Properties of Stochastic POS Tagging:**
    - Based on the probability of tag occurring.
    - Requires a training corpus with vast stored information.
    - No probability for words that do not exist in the corpus.
    - Uses different testing corpus (other than training corpus).
    - Simplest POS tagging because it chooses the most frequent tags associated with a word in the training corpus.
- **Stochastic POS Tagging: HMM Tagger:**
    - Picks the most likely tag based on context.
    - Maximizes the formula using a Hidden Markov Model (HMM):
        - Formula: $P(\text{word}|\text{tag}) \times P(\text{tag}|\text{previous } n \text{ tags})$
    - Goal: Find POS tags that generate a sequence of words.
    - Uses transition probabilities (forward tag and backward tags).
    - Formula: $P(t_i/w_i) = P(t_i/t_{i-1}) \times P(t_{i+1}/t_i) \times P(w_i/t_i)$
    - $P(t_i/t_{i-1})$: Probability of current tag given previous tag.

Transformation-Based Tagging

- Also called **Brill Tagging**.
- A rule-based algorithm for automatic tagging of POS to the given text.
- Allows linguistic knowledge in a readable form.
- Transforms one state to another state using transformation rules.
- **Combination of Rule-Based and Stochastic Tagging:**
    - Like rule-based because rules specify tags in a certain environment.
    - Like stochastic because it uses machine learning with a tagged corpus as input.

- **Input:** Tagged corpus and a dictionary (with most frequent tags).
- **Working of Transformation-Based Learning (TBL):**
  - **Step 1:** Start with a solution.
  - **Step 2:** Choose the most beneficial transformation in each cycle.
  - **Step 3:** Apply the transformation to the problem.
  - The algorithm stops when no further beneficial transformations are available or selected.

Issues/Challenges in POS Tagging

- **Multiple Tags and Multiple Words:**
  - Adjective vs past tense vs past participle (JJ/VBD/VBN).
- **Unknown Words:**
- **Foreign Words:**
- **Ambiguities:**
  - Common words have multiple meanings and therefore multiple POS.
- **Ungrammatical Input:**
- **Tokenization Issues:**

Sequence Labelling

- **Definition**: In natural language processing (NLP), sequence labelling is the task of assigning a label to each element in a sequence. This is often used to extract words or phrases of particular types from a sentence or paragraph.
- **Example**:
  - When analyzing news articles, one might want to identify the countries mentioned and the frequency of mentions.
  - **Input**: `["Paris", "is", "the", "capital", "of", "France"]`
  - **Output**: `["I", "I", "I", "I", "I", "C"]`

        - Here, "I" means the token is irrelevant, and "C" indicates that the token is part of a country name.

Methods of Sequence Labelling

1. **Dictionary-Based Approach**:
    - A simple approach where a predefined dictionary of entities (like country names) is used to identify relevant tokens in a sentence.
2. **Hidden Markov Model (HMM)**:
    - A statistical model that represents the probabilities of sequences of observed and hidden states, commonly used in POS tagging and other sequence labelling tasks.
3. **Maximum Entropy**:
    - A model that uses a feature-based approach to assign probabilities to various outcomes, maximizing the entropy (uncertainty) subject to the known constraints from the data.
4. **Conditional Random Fields (CRFs)**:
    - Discriminative models that directly model the conditional probability of the label sequence given the observed sequence.
    - CRFs focus on maximizing $P(Y|X)$, where X is the sequence of observations, and Y is the sequence of labels.
    - They capture dependencies between adjacent labels and do not model the distribution of the observations X.

Conditional Random Fields (CRFs)

- **Linear Chain CRF**:
    - A standard CRF where tag assignment for the current word (denoted as y_i) depends only on the tag of the previous word (denoted as y_{i-1}).
    - Used for tasks such as named-entity recognition (NER), where the goal is to identify proper nouns, like names of people, organizations, and locations.
- **Example of NER**:
    - Sentence: "John Doe is a researcher at the University of California."

        - "John Doe" and "University of California" would be tagged as named entities.
    - Sentence: "Later, John attended a conference."
        - "John" would be recognized as a reference to the previously mentioned "John Doe."
- **Skip-Chain CRF**:
    - A variant of CRF where additional edges are introduced to capture long-range dependencies, such as connecting repeated mentions of the same entity (e.g., multiple mentions of "John" or "University of California").

Applications of CRF

- **Named-Entity Recognition (NER)**: Identifying proper nouns in text.
- **Feature Induction for NER**: Automatically determining features for improving NER performance.
- **Shallow Parsing**: Breaking down a sentence into its constituent parts (like noun phrases).
- **Identifying Protein Names in Biology Abstracts**: Recognizing specialized terms in scientific texts.
- **Segmenting Addresses in Web Pages**: Extracting structured information from unstructured text.
- **Information Integration**: Combining data from multiple sources.
- **Finding Semantic Roles in Text**: Understanding the role that a word plays in a sentence (e.g., subject, object).