1. What is Natural Language Processing (NLP)?
Answer: Natural Language Processing (NLP) is a field of Artificial Intelligence (AI) concerned with the interactions between computers and human (natural) languages. It focuses on enabling computers to understand, interpret, and generate human language in a way that is both meaningful and useful.
2. Mention any two real-world applications of NLP.
Answer:

- **Sentiment Analysis:** Determining the emotional tone or attitude expressed in text, used for market research, brand monitoring, etc.
- **Chatbots and Conversational AI:** Building interactive agents that can engage in conversations with humans, provide customer service, answer questions, and more.

3. Define empirical laws in the context of NLP.
Zipf's Law: When words are ranked according to their frequencies in a large enough collection of texts and then the frequency is plotted against the rank, the result is a logarithmic curve.
Heap's law states that the number of unique words V in a collection with N words is approximately Square root of N.
4. What are the key steps involved in text processing?
Keys Steps involved in text processing are as follows:
Text Cleaning In this step, we will perform fundamental actions to clean the text. These actions involve transforming all the text to lowercase, eliminating characters that do not qualify as words or whitespace, as well as removing any numerical digits present.
Tokenization Tokenization is the process of breaking down large blocks of text such as paragraphs and sentences into smaller, more manageable units.
Stopword Removal Stopwords refer to the most commonly occurring words in any natural language. One of the advantages of removing stopwords is that it can reduce the size of the dataset, which in turn reduces the training time required for natural language processing models.
Stemming/Lemmatization Stemming is a process that stems or removes last few characters from a word, often leading to incorrect meanings and spelling.
Lemmatization considers the context and converts the word to its meaningful base form, which is called Lemma.
5. Explain different types of ambiguity in NLP with example.
Ambiguity in NLP arises when the same word or sentence can be interpreted in multiple ways. The sources detail different types of ambiguity along with examples:

- **Lexical Ambiguity**: This occurs when a single word has multiple possible meanings.
    - For example, the word "will" can have different interpretations. In the sentence "will will will will's will" the word "will" is used five times with different meanings.
    - Identifying whether "rose" refers to 'r o s e' or 'r o s e s' is also an instance of lexical ambiguity.
    - Another example is the word "duck" which can be either a noun or a verb.
    - Words like "make" can mean "to create" or "to cook".

- **Structural Ambiguity**: This arises when the same sentence can be interpreted in multiple ways due to differing syntactic structures.
    - For example, in the sentence "The man saw the boy with the binoculars," it is unclear whether the man or the boy is holding the binoculars.
    - In the sentence "what are police looking in to," the ambiguity lies in whether the police are looking into the hole or the matter.
    - "Stolen painting found by tree" can be interpreted as the tree finding the painting, while the actual meaning is the painting was found near the tree.
- **Syntactic Category Ambiguity**: A word's role in a sentence can be ambiguous.
    - The word "duck" can be a noun or a verb.
    - "Her" can be a possessive or dative.
- **Semantic Ambiguity**: This involves different interpretations at the meaning level.
    - Words can have several meanings, referred to as different senses of the word. For example, the word "bass" can refer to fish or music.
- **Anaphoric Ambiguity**: Ambiguity in determining what a word or phrase refers back to.
- **Referential Ambiguity**: Noun phrases may have a confusing resemblance to expressions that are not referring expressions.
- **Segmentation Ambiguity**: This refers to the uncertainty in how a sentence should be segmented.
    - For example, the sentence "New York-New Heaven Railroad" can be segmented in two ways: "New York - New Heaven" or "New York, New Heaven".
- **New word and new senses**: The continuous introduction of new words and senses creates challenges for NLP.
    - For example, words like Google, Skype, and Photoshop are now used as verbs. The word "sick" can be used to mean something is good. Ambiguity is pervasive in natural language, making NLP challenging. Successfully resolving ambiguity often requires using knowledge and inference capabilities.

6. **Explain the concept of tokenization in text processing.**
Tokenization is a term that describes breaking a document or body of text into small units called tokens. You can define tokens by certain character sequences, punctuation, or other definitions depending on the type of tokenization. Doing so makes it easier for a machine to process the text.
7. **What is the difference between stemming and lemmatization?**

Aspect	**Lemmatization**	**Stemming**
Definition	Converts words to their base or dictionary form (lemma).	Reduces words to their root form (stem), which may not be a valid word.
Complexity	Higher complexity, context-aware.	Lower complexity, context-agnostic.

Aspect	**Lemmatization**	**Stemming**
Algorithms	Uses dictionaries and morphological analysis.	Uses rule-based algorithms like Porter, Snowball, and Lancaster Stemmers.
Accuracy	Produces more accurate and meaningful words.	Less accurate, may produce non-meaningful stems.
Output Example	"Running" → "run", "Better" → "good".	"Running" → "run" or "runn", "Better" → "bett".
Speed	Slower due to more complex processing.	Faster due to simpler rules.
Use in Search Engines	Better search results through understanding context.	Useful for quick search indexing
Text Analysis	Essential for tasks needing accurate word forms (e.g., sentiment analysis, topic modeling)	Used for initial stages of preprocessing to reduce word variability
Machine Translation	Helps in producing grammatically correct translations	Less common due to potential inaccuracy
Information Retrieval	Suitable for detailed and precise analysis	Useful for reducing data dimensionality

8. **Define edit distance with an example.**

Edit distance, also known as **Levenshtein distance**, is a metric used to quantify the similarity between two strings. It measures the minimum number of single-character edits (insertions, deletions, or substitutions) required to transform one string into the other.

The three basic types of single-character edits are:

1. **Insertion**: Adding a single character to one of the strings.
2. **Deletion**: Removing a single character from one of the strings.
3. **Substitution**: Replacing a character in one of the strings with another character.

To transform *Saturday* into *Sunday*, we need the minimum number of single-character edits (insertions, deletions, or substitutions).

1. The first character (*S*) is the same, so no change is needed.
2. Replace *a* with *u*.
3. Replace *t* with *n*.
4. Replace *u* with *d*.
5. Replace *r* with *a*.
6. Delete *d* since *Sunday* is shorter.

This results in **five edits**—four substitutions and one deletion—giving an **edit distance of 5**.

9. **What is Regular Expression.Write Regular Expression for the following: i. To extract USN numbers ii. To extract the email id iii To extract the Phone numbers**

A regular expression (regex) is a language for specifying text search strings and is a powerful tool for text processing in computer science. It is an algebraic notation for characterizing a set of strings and is used in every computer language, text processing tools, and editors. Regular expressions are useful when there is a pattern to search for and a corpus of texts to search through.

Email IDs: A general pattern is **^[^@]+@[^@]+\.[^@]+$**.

Phone Numbers: A basic pattern for phone numbers could be **^\+?[1-9][0-9]{10,12}$.**

USN: NNM\d{2}RI\d{3}

9. How is weighted edit distance different from standard edit distance?

Standard edit distance assigns an equal cost (usually 1) to each edit operation (insertion, deletion, substitution). Weighted edit distance allows assigning different costs to different operations. This is useful when certain types of errors are more probable or have different importance in a specific application. For instance, in spell correction, substituting vowels might have a lower cost than substituting consonants, reflecting typical typing errors.

8. Explain the purpose of N-gram language models

N-gram language models are probabilistic models that predict the likelihood of a word sequence. They are based on the idea that the probability of a word depends on the preceding N-1 words. Their purpose is to capture statistical patterns in language and estimate the probability of sentences, which is crucial for tasks like text generation, speech recognition, machine translation, and spelling correction.

9. Write the formula for calculating bigram probabilities.

In a bigram model, the probability of a word w_i given its preceding word w_{i-1} is calculated using the maximum likelihood estimate (MLE):

$$P(w_i \mid w_{i-1}) = \frac{\text{count}(w_{i-1}, w_i)}{\text{count}(w_{i-1})}$$

Here:

- count (w_{i-1}, w_i) is the number of times the bigram (w_{i-1}, w_i) occurs in the corpus.
- count(w_{i-1}) is the total number of times the word w_{i-1} appears (serving as the first word in any bigram).

This formula gives the conditional probability that word w_i follows word w_{i-1} based on the observed frequencies in your training data

10. What are the limitations of N-gram models?

The following are the limitations of N-gram models

i. The probability estimates depend largely on the occurrence of words in the training corpus, so this is not desirable in circumstances where the test corpus looks vastly different. For example, an n-gram model trained on works of Shakespeare would not generate good predictions for the works of another playwright or poem.

ii. N-grams can have too many parameters as mentioned before. This means in order to use a better performing n-gram, which usually has a higher n, we must choose one with more parameters, especially when using larger corpora for better probability estimates. This often requires better feature selection approaches.

iii. N-gram models struggle to capture longer-distance context clues. It has been shown that after 6-grams, the gain in performance is limited. This is unfavorable in tasks where that is a particularly desirable feature and necessity.

11. What are Corpora? How it is useful in the NLP?

Corpora are computer-readable collections of text or speech. They are essential in NLP for various tasks. Corpora are used for **training language models**, creating **training and test sets**, **semantic analysis**, **machine learning**, **part-of-speech tagging**, **named entity recognition**, **coreference resolution**, **discourse analysis**, **machine translation**, and **question answering**.

12. Define perplexity in language models.

Perplexity is a metric used to evaluate how well a language model predicts a sequence of words. In simple terms, it measures the model's "surprise" when encountering a test set: a lower perplexity indicates tc vhat the model is better at predicting the test data. Here are the key points: More formally, given a test set $W = w_1, w_2, \ldots, w_N,$ the perplexity is defined as:

$$\text{Perplexity}(W) = P(W)^{-1/N} = \exp\left(-\frac{1}{N}\sum_{i=1}^{N}\log P(w_i \mid w_1^{i-1})\right)$$

12.

What is the role of smoothing in language models?

Smoothing techniques help in the following ways:

1.Avoiding zero probabilities: Smoothing methods ensure that no n-gram has a probability of zero. This is important because, in real-world language data, it's unlikely that every possible n-gram will be observed. Smoothing assigns a non-zero probability to unseen or infrequent n-grams, allowing the model to assign some likelihood to these unseen events.

2.Reducing overfitting: Smoothing helps prevent overfitting of the language model to the training data. Overfitting occurs when a model becomes overly specific to the training data and fails to generalize well to unseen data. By smoothing the probabilities, the model assigns some probability mass to unseen events, reducing the risk of overfitting and improving the model's generalization ability.

3.Handling data sparsity: Language models often encounter rare or infrequent n-grams that have limited or no training examples. Smoothing techniques estimate probabilities for these unseen events by redistributing probability mass from observed events. This helps in more accurately modeling the likelihood of unseen or infrequent n-grams.

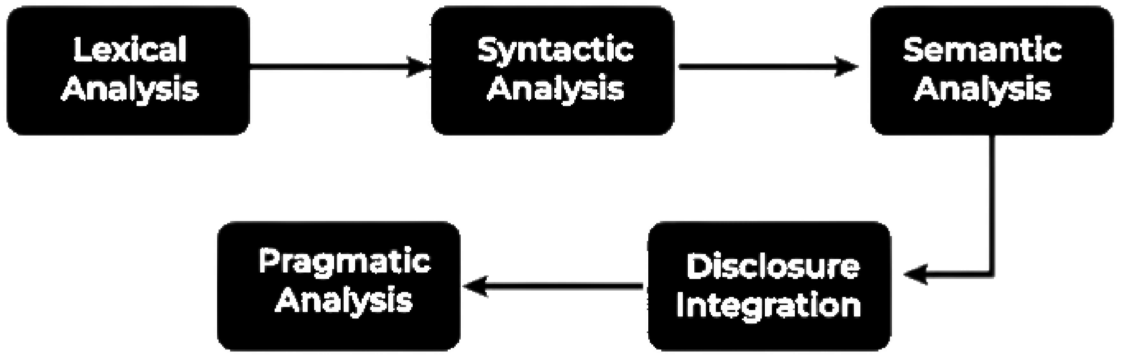
4.Improving interpolation and backoff: Smoothing methods are crucial for interpolation and backoff techniques in language modeling. Interpolation combines probability estimates from different n-gram orders, and smoothing ensures that even if a higher-order n-gram has insufficient counts, lower-order n-grams can still contribute to the probability estimation. Similarly, in backoff, smoothing allows the model to back off to lower-order n-grams when higher-order n-grams have limited or no data.

13.Briefly explain Laplace smoothing.

Laplace Smoothing works by adding 1 to the count of every possible n-gram, including those that were not observed in the training data. This adjustment ensures that no n-gram has a zero probability, which would indicate that it is impossible according to the model. By doing so, Laplace Smoothing distributes some probability mass to these unseen n-grams, making the model more adaptable to new data.
Here's a step-by-step breakdown of how Laplace Smoothing is applied:

1. **Count the N-grams**: First, count the occurrences of all n-grams in the training data.
2. **Add 1 to All Counts**: Add 1 to the count of each n-gram, including those with zero counts.
3. **Adjust the Denominator**: Add the size of the vocabulary V to the denominator, accounting for the total number of possible n-grams.

14. Explain the neat diagram different steps in the NLP.



NLP transforms raw text into a structured format for analysis and application development, involving several key steps. These steps form a pipeline
The general steps in NLP are
Lexical Analysis: This involves identifying and analyzing the structure of words, dividing text into paragraphs, sentences, and words. A lexicon is a collection of words and phrases in a language
Syntactic Analysis (Parsing): This analyzes words in a sentence for grammar and arranges them to show the relationships between the words
. For example, "The school goes to boy" would be rejected by an English syntactic analyzer
Text Normalization: This involves converting text to a standard format, including lowercasing, removing punctuation, and handling special characters
Tokenization: This breaks down text into individual words or phrases, known as tokens. This is often the first step in text processing.

15. How does back-off smoothing work in language models?
Backoff smoothing, also known as Katz smoothing, is a smoothing technique that uses lower-order models to estimate the probabilities of higher-order models. For example, a bigram model is a language model that uses two words to predict the next word, while a unigram model uses only one word. Backoff smoothing works by using the bigram model if there is enough evidence for it, and falling back to the unigram model otherwise. This way, it avoids assigning zero probabilities to unseen bigrams, and it also discounts the probabilities

of seen bigrams to account for the backoff. Backoff smoothing is more flexible and realistic than additive smoothing, but it requires more parameters and computation.

15. Differentiate between unigrams and bigrams with examples.

Unigrams and **bigrams** are types of n-grams, where "n" represents the number of tokens in each group. The main differences are:

Unigrams: Unigrams are individual tokens (typically words) taken separately from a text. Unigrams are used to capture the frequency of individual words. However, they do not consider the context or order in which words appear.

Bigrams: Bigrams are pairs of consecutive tokens from a text. Bigrams capture relationships between adjacent words, providing context that can be useful in tasks like language modeling, sentiment analysis, and machine translation.

Consider the sentence:

"Smart algorithms solve complex problems."

Unigrams:

Each individual word is treated separately. The unigrams are:

"Smart", "algorithms", "solve", "complex", "problems"

Bigrams:

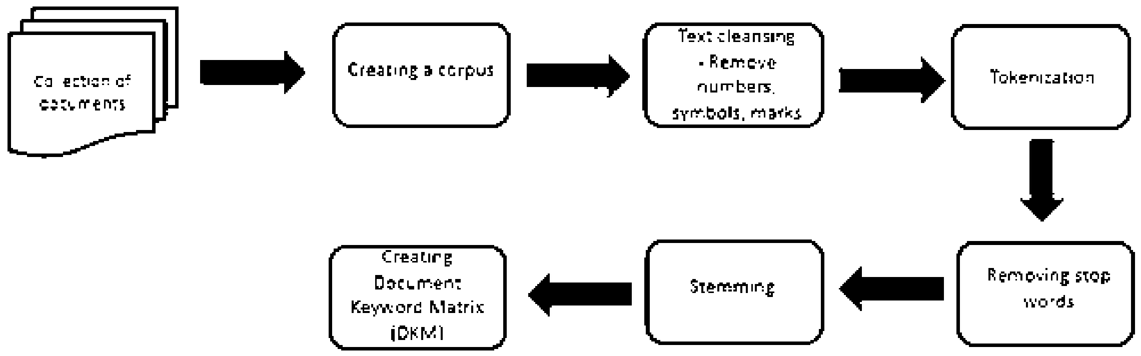
Pairs of consecutive words are grouped together. The bigrams are:

"Smart algorithms", "algorithms solve", "solve complex", "complex problems"

16. Why is text preprocessing important?

Data quality significantly influences the performance of a machine-learning model. Inadequate or low-quality data can lead to lower accuracy and effectiveness of the model. In general, text data derived from natural language is unstructured and noisy. So text preprocessing is a critical step to transform messy, unstructured text data into a form that can be effectively used to train machine learning models, leading to better results and insights.

17. Describe the process of text pre-processing with suitable examples.


Collection of documents
Creating a corpus
Text cleansing - Remove numbers, symbols, marks
Tokenization
Removing stop words
Stemming
Creating Document Keyword Matrix (DKM)

Text preprocessing typically involves the following steps:

- Lowercasing
- Removing Punctuation & Special Characters
- Stop-Words Removal

- Removal of URLs
- Removal of HTML Tags
- Stemming & Lemmatization
- Tokenization
- Text Normalization

Some or all of these text preprocessing techniques are commonly used by NLP systems. The order in which these techniques are applied may vary depending on the needs

Lowercasing is a text preprocessing step where all letters in the text are converted to lowercase. This step is implemented so that the algorithm does not treat the same words differently in different situations.

Removing Punctuation & Special Characters

Punctuation removal is a text preprocessing step where you remove all punctuation marks (such as periods, commas, exclamation marks, emojis etc.) from the text to simplify it and focus on the words themselves.

Stop-Words Removal

Stopwords are words that don't contribute to the meaning of a sentence. So they can be removed without causing any change in the meaning of the sentence.

Removal of URLs

This preprocessing step is to remove any URLs present in the data.

Removal of HTML Tags

Removal of HTML Tags is a text preprocessing step used to clean text data from HTML documents. When working with text data obtained from web pages or other HTML-formatted sources, the text may contain HTML tags, which are not desirable for text analysis or machine learning models.

18. Discuss Zipf's Law and Heaps' Law with examples.

Zipf's Law states that in a large corpus of natural language, the frequency *f* of any word is inversely proportional to its rank rrr in the frequency table. In other words, if you sort all words by their frequency, the second most frequent word appears approximately half as often as the most frequent, the third word about one-third as often, and so on. Mathematically, the law can be written as:

$$f(r) \propto \frac{1}{r^s}$$

where s is typically close to 1 for natural language. If the most frequent word appears k times, then the r-th ranked word will appear about k/r times.

Example:

Consider a corpus such as a collection of news articles. In many English texts, the word "the" is the most frequent. Suppose "the" occurs 10,000 times; then Zipf's Law suggests that the second most common word might occur roughly 5,000 times, the third about 3,300 times, and so on. Although real data rarely follow the law perfectly (especially at the very high and low frequency ends), the overall pattern is striking. This regularity has been observed across languages and types of text .

Heaps' Law (sometimes called Herdan's Law) describes how the number of distinct words (vocabulary size VVV) grows as a function of the total number of words NNN in a text corpus. The law is typically expressed as:

$$V(N) = K \cdot N^{\beta}$$

where K is a constant and β\betaβ is usually between 0.4 and 0.6 for natural language. The sub-linear exponent $\beta<1$ means that as the corpus grows larger, the rate at which new (unique) words appear slows down.

Example:

Imagine you have a corpus of 1 million words and you observe a vocabulary of 50,000 unique words. If you increase your corpus size to 4 million words, Heaps' Law predicts that the vocabulary might grow roughly by a factor of 4^{β}. With β around 0.5, the vocabulary size would be expected to be about:

$V(4 \text{ million}) \approx 50{,}000 \times (4)^{0.5} = 50{,}000 \times 2 = 100{,}000$

This sub-linear growth shows that while more text brings in new words, there is considerable repetition even in very large corpora .

In many NLP tasks, you will see that a few words (e.g., "the," "of," "and") dominate text (Zipf's Law), while the overall number of unique words grows more slowly than the total number of words (Heaps' Law). This explains why even massive corpora have a relatively contained vocabulary compared to the total number of tokens.

19. Explain different smoothing techniques

Common smoothening mechanisms:

Laplace (add-one) smoothing: This is a simple and intuitive smoothing method where a constant (usually 1) is added to the count of each n-gram. It ensures that no probability is assigned as zero, but it can result in over-smoothing.

Good-Turing smoothing: Good-Turing smoothing estimates the probability of unseen or infrequent n-grams based on the counts of observed n-grams. It uses the observed frequency distribution to estimate the probability mass for unseen events.

Kneser-Ney smoothing: Kneser-Ney smoothing, which we discussed earlier, is a more advanced and widely used technique. It incorporates modified Kneser-Ney discounting to estimate probabilities for unseen or infrequent n-grams by considering the frequency of their continuations.

Absolute discounting: Absolute discounting applies a discounting factor to the count of each n-gram, redistributing the discounted probability mass to unseen or infrequent n-grams. The discounting factor is typically based on the frequency of each n-gram.

20. Explain the role of N-gram models in language modelling. What are their advantages and disadvantages?

N-gram models serve as statistical language models that predict the probability of a word based on a fixed-size history. They are fundamental in tasks like speech recognition, machine translation, and text generation, where predicting the next word in a sequence is critical.

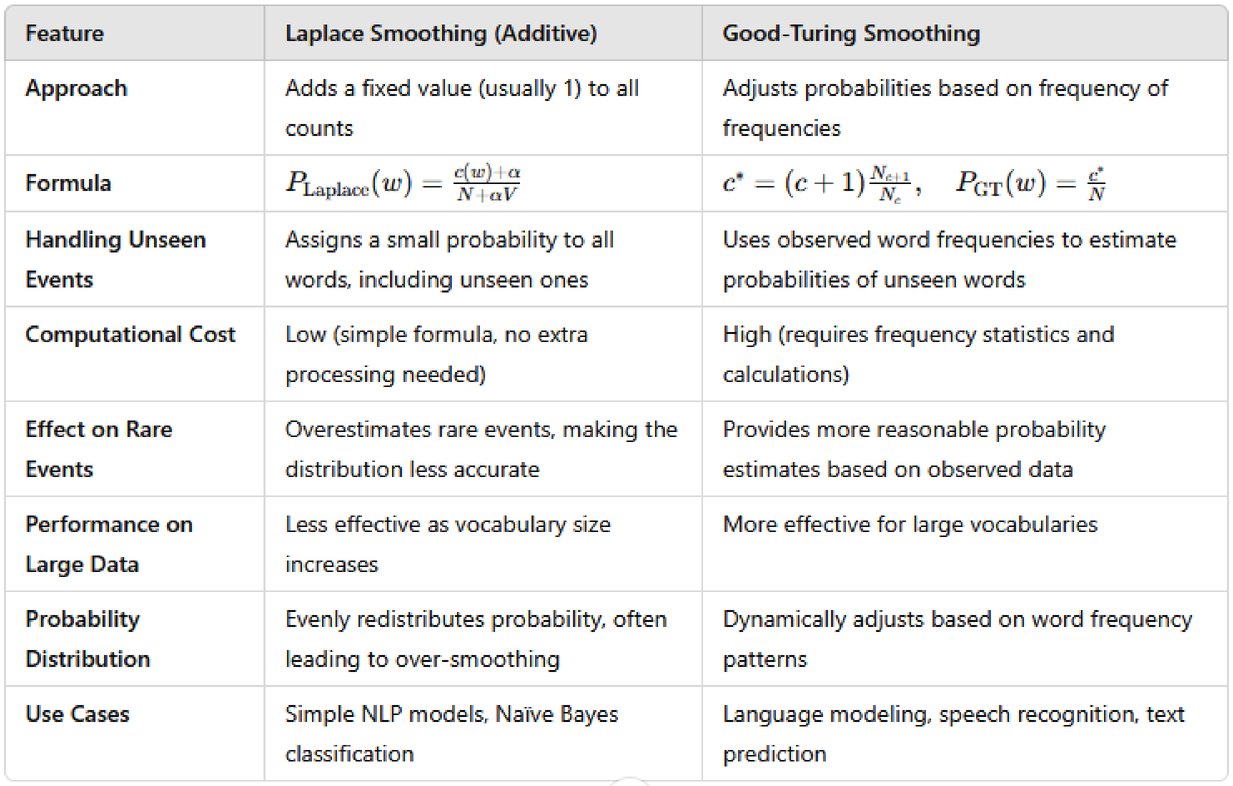
Advantages:

Their simplicity, interpretability, and computational efficiency (for smaller values of n) make them useful baseline models and practical solutions for many applications.

Disadvantages:

They suffer from data sparsity and the inability to capture long-range dependencies. As the value of nn increases, models become computationally expensive and prone to overfitting, which necessitates careful smoothing and optimization.

21. Compare and contrast Laplace and Good-Turing smoothing methods.



Feature	Laplace Smoothing (Additive)	Good-Turing Smoothing
Approach	Adds a fixed value (usually 1) to all counts	Adjusts probabilities based on frequency of frequencies
Formula	$P_{\text{Laplace}}(w) = \frac{c(w)+\alpha}{N+\alpha V}$	$c^* = (c+1)\frac{N_{c+1}}{N_c}, \quad P_{\text{GT}}(w) = \frac{c^*}{N}$
Handling Unseen Events	Assigns a small probability to all words, including unseen ones	Uses observed word frequencies to estimate probabilities of unseen words
Computational Cost	Low (simple formula, no extra processing needed)	High (requires frequency statistics and calculations)
Effect on Rare Events	Overestimates rare events, making the distribution less accurate	Provides more reasonable probability estimates based on observed data
Performance on Large Data	Less effective as vocabulary size increases	More effective for large vocabularies
Probability Distribution	Evenly redistributes probability, often leading to over-smoothing	Dynamically adjusts based on word frequency patterns
Use Cases	Simple NLP models, Naïve Bayes classification	Language modeling, speech recognition, text prediction

21. Define Stemming and Lemmatization What are the differences between Lemmatization and Stemming?

Definition: Stemming involves reducing words to their root or base form. It aims to simplify words to their core, removing prefixes or suffixes.

Lemmatization is similar to stemming but more sophisticated. It involves reducing words to their base or dictionary form (lemma) to ensure a valid word.

Differences between Stemming and Lemmatization:

Output Validity:

Stemming: May result in non-valid words, as it focuses on removing prefixes and suffixes to obtain a common root.

Lemmatization: Ensures the output is a valid word, considering the context and part of speech.

Context and Part of Speech:

Stemming: Generally doesn't consider context or part of speech. It applies rules to trim words.

Lemmatization: Takes into account context and the grammatical role of the word in a sentence.

Use Cases:

Stemming: Often used for information retrieval or search engines where computational efficiency is crucial.

Lemmatization: Preferred in applications where maintaining the validity of words and understanding context is essential, such as question answering systems or chatbots.

Choose Stemming: When you want simplicity and speed, and non-valid words are acceptable.

Choose Lemmatization: When maintaining word validity and understanding context is critical.

Examples of Stemming and Lemmatization

Example 1:

Original: “Dancing, dancer, danced.”

Stemming Result: [“danc”, “dancer”, “danc”]

Lemmatization Result: [“dance”, “dancer”, “dance”]

Example 2:

Original: “Organization, organizational, organized.”

Stemming Result: [“organ”, “organ”, “organ”]

Lemmatization Result: [“organization”, “organizational”, “organize”]

Example 3:

Original: “Happiness, happier, happiest.”

Stemming Result: [“happi”, “happier”, “happiest”]

Lemmatization Result: [“happiness”, “happy”, “happy”]

22. Define advanced smoothing models in language modelling.

Advanced smoothing models are techniques used in language modeling to better estimate the probability of sequences of words, especially when dealing with unseen n-grams in a training corpus. Advanced Smoothing Models:

- Good-Turing Smoothing: This method aims to determine the probability for words that have not appeared in the training data.
- Kneser-Ney Smoothing: Aims to improve upon regular unigram correction. It gives higher weight to words that occur more often in the corpus.
- Interpolation: Advanced smoothing can involve interpolation methods. In interpolation, the value of lambda depends on the context. To choose the lambdas, a held-out corpus is used to determine which lambda values yield the highest probability.
- Unigram Prior Smoothing: Instead of adding a uniform weight to each n-gram, higher weights are assigned to words that occur more frequently in the corpus.
- Add-k Smoothing: Instead of adding 1 to each count (as in add-one smoothing), a fractional count k is added. The value of k can be optimized using a devset.

23 What is computational morphology?

In Morphology, we study the internal structure of words and how words are built up from smaller meaningful units called Morphemes.

24. Differentiate between inflectional and derivational morphology.

Inflectional morphology and **derivational morphology** are two main types of morphology that deal with how words change form, but they do so in different ways.

Inflectional Morphology:

- **Function**: Inflectional morphology modifies a word to express different **grammatical categories** such as tense, number, gender, or case. These modifications are made **without changing the word's core meaning or part of speech**.
- **Core Meaning**: The addition of inflectional morphemes changes the grammatical function of a word but not its basic meaning. For example, "run" and "ran" have the same core meaning, but "ran" indicates past tense.
- **Word Class**: Inflectional morphemes do not change the part of speech of the word. For instance, adding "-s" to "cat" to make "cats" keeps the word as a noun.
- **Examples**:
  - walk → walked (past tense)
  - cat → cats (plural)
  - bring → brought, brings
- **Grammatical functions** include subject-verb agreement or tense.
- **Obligatory**: Often obligatory in forming grammatically correct sentences, as it provides essential grammatical information.
- **Productivity**: More productive, applies regularly across many words within the same grammatical category.
- Important for NLP tasks like **lemmatization**, **part-of-speech tagging**, and **syntactic parsing**.
- **Limited number of inflectional affixes**. English has eight inflectional suffixes that serve a variety of grammatical functions.

Derivational Morphology:

- **Function**: Derivational morphology creates **new words** by adding affixes (prefixes, suffixes) to a base or root word. It often changes the meaning or part of speech of the original word.
- **Word Class**: Often changes the word class (e.g., verb to noun, adjective to adverb).
- **Semantic Impact**: Can significantly alter the meaning of the original word, sometimes creating a completely new concept.
- **Examples**:
  - happy → happiness (adjective to noun)
  - teach → teacher (verb to noun)
  - logic → logical

    - kind → unkind (adjective to adjective with opposite meaning)
- **Optional**: Generally optional; a base word can exist without derivation.
- **Less productive**: Not all words can take the same derivational affixes.
- Understanding derivational morphology is essential in NLP for tasks such as **part-of-speech tagging**, **named entity recognition**, **machine translation**, and **information retrieval**. Derivational morphology is also helpful in sentiment analysis.

25. What are finite-state methods in morphology?

Finite-state methods are a popular approach in computational morphology for processing and analyzing the structure of words. They use finite-state machines to model morphological processes.

Here's a breakdown of how finite-state methods are applied in morphology:

- **Finite State Automata (FSA)**: FSAs can recognize words by determining whether a word is singular or plural. FSAs have a finite number of states and transitions between them.
- **Finite State Transducers (FSTs)**: FSTs are an extension of FSAs that generate output strings based on input. FSTs are commonly used for tasks like morphological analysis, machine translation, and speech processing.
    - FSTs can **map a root form to its inflected forms**. For example, an FST could map "walk" to "walked", "walking", and "walks".
- **How FSTs Work in Morphology**:
    - FSTs model the **combination of morphemes** and the **changes that occur at morpheme boundaries**.
    - They use a directed graph where nodes are states and edges between the nodes are transitions.
    - When combining morphemes, certain changes happen at the boundary, and FSTs capture this regular phenomenon.
    - For instance, an FST can capture the pluralization of nouns by adding an "s".
    - FSTs can also be used to convert an NFA (nondeterministic finite automaton) to a DFA (deterministic finite automaton) and further minimize the number of states.
- **Two-Level Morphology**: This approach uses an intermediate level between the lexical form (e.g., "cat + noun + plural") and the surface form (e.g., "cats"). Instead of going directly from the lexical to the surface level, it goes to the intermediate level first.
    - The transition from the intermediate to the surface level depends on the ending characters of the stem and the starting characters of the affix.
    - Context-sensitive rules can be applied to convert the intermediate form to the surface form.
- **Morphophonology**: FSTs are used to handle morphophonology, which is the area of linguistics that deals with the relations and interaction of morphology with phonology. FSTs can model spelling variations.
    - For example, the plural affix "-s" can be pronounced differently depending on the stem it attaches to (e.g., /z/ in "flags," /ɪz/ in "glasses," and /s/ in "cats").

- **Advantages**:
  - FST implementations are relatively straightforward and efficient.
  - FSTs can simultaneously model morphological generation and analysis.
- **Applications of FSTs**:
  - **Morphological analysis**
  - **Machine translation**
  - **Speech processing**
- **Limitations and Alternatives**:
  - **Difficult Morphology**: FSTs, which rely on an item and arrangement (I&A) model of word structure, may face challenges with languages that exhibit non-isomorphism or non-contiguity.
  - **Paradigm-Based Approaches**: An alternative to FSM is a paradigm-based approach, where word structure is computed by the stem's place in a cell in a paradigm.
- **Popular Tools**: OPENFST is a popular tool for doing morphological analysis for a given language.

26. Explain the concept of finite-state transducers with an example.

A **finite-state transducer (FST)** is a type of finite-state machine that **generates output strings based on input**. It is an extension of a finite-state automaton (FSA). FSTs are used in natural language processing (NLP) for morphological analysis, speech recognition, and machine translation.

Key concepts related to FSTs:

- **States and Transitions**: An FST, similar to an FSA, has a finite number of states and transitions between them. Each transition is labeled with an input symbol and an output symbol (or an empty symbol, ε). The FST moves from one state to another based on the input, while producing corresponding output.
- **Input and Output**: For each input symbol, the FST generates an output symbol. The output can be a transformed version of the input.
- **Types of FSTs**:
  - **Deterministic FST (DFST)**: For each state and input symbol, there is exactly one transition. The output is uniquely determined by the input.
  - **Non-Deterministic FST (NFST)**: A given input symbol may lead to multiple possible transitions, allowing multiple possible outputs.
- **Applications in NLP**:
  - **Morphological Analysis**: FSTs can map a root form to its inflected forms. For example, an FST could map "walk" to "walked", "walking", "walks", etc.
  - **Machine Translation**: FSTs can model translation rules where input words or phrases are translated into another language.
  - **Speech Processing**: FSTs can map phonetic representations to text or vice versa, enabling tasks like speech recognition or text-to-speech.

Example in Morphological Analysis: Consider a simple FST for English plural formation:

- **Input**: "cat"
- **States**: q0 (start), q1 (final)
- **Transitions**: (q0, "cat", q1, "cats")

In this case, the input "cat" transitions the FST from q0 to q1 and produces the output "cats".

27. What is Part-of-Speech (POS) tagging?
Part-of-speech (POS) tagging is the process of assigning a **grammatical category** to each word in a text, like nouns, verbs, adjectives, etc.. It involves labeling each word in a sentence with its appropriate part of speech. POS Tagging helps to understand the context of words in a sentence and also disambiguate words that can have multiple parts of speech. It is an essential initial step for higher-level NLP tasks.
28. List any two applications of POS tagging.
Two applications of Part-of-Speech (POS) tagging are:

- **Syntactic parsing**: POS tags of words in a sentence are needed to determine the correct word combinations.
- **Named-entity recognition**: POS tagging helps in identifying entities and the relationships between them. Named Entity Recognition (NER) is used in applications like information retrieval and question answering systems.

29. With the help of example, explain the process of POS Tagging
Part-of-speech (POS) tagging is an NLP task that involves **assigning a grammatical tag** (like noun or verb) **to each word in a text**. POS tagging is a **disambiguation task** because words can have more than one possible part-of-speech, so the proper tag must be chosen based on the context. Models like **Hidden Markov Models (HMMs)**, **Conditional Random Fields (CRF)**, and **neural networks** are used, and the accuracy is measured by comparing the tags to human-annotated "gold labels". For example, in the sentence "Janet will back the bill", the words are tagged as: Janet/NNP, will/MD, back/VB, the/DT, bill/NN.

29. Explain different approaches used for POS tagging.
Different approaches can be used to tackle the problem of Part-of-Speech (POS) tagging. POS tagging involves identifying the category of each word in a sentence or document. The correct tag must be determined for a particular instance of a word based on the context.
Here's an explanation of different approaches for POS Tagging, drawing from the sources:

- **Rule-based Approach:** This involves linguists defining patterns or "if-then-else" rules to determine the appropriate tag based on context.
- **Statistical Tagging:** This approach uses a training corpus, which is a text where each word is tagged with its correct POS category, to learn the actual tag for each individual word. Different models can be used.
- **Probabilistic Tagging Models:** These models involve some data observations and a certain class. In the case of POS tagging, the words are the data, and the tags are the classes that need to be assigned to different words.
- **Hidden Markov Model (HMM):** The HMM is a generative approach used for sequence labeling, including POS tagging. HMMs determine the most probable tag sequence for a given sentence. A disadvantage of HMM is the handling of unknown words.
- **Maximum Entropy Markov Model (MEMM):** MEMM is a discriminative model.
- **Conditional Random Fields (CRF):** CRFs are discriminative models used for sequence labeling tasks. CRFs can incorporate a lot of features.

- **Neural Approaches:** Neural approaches to NER mainly follow from the pioneering results of Collobert et al. (2011), who applied a CRF on top of a convolutional net. BiLSTMs with word and character-based embeddings as input followed shortly and became a standard neural algorithm for NER (Huang et al. 2015, Ma and Hovy 2016, Lample et al. 2016) followed by the more recent use of Transformers and BERT.

Key considerations in POS Tagging:

- **Tagsets**: Defining the tagset, which is the set of all possible categories among which you have to choose. Tagsets can range from coarse (noun, verb, adjective, adverb) to fine-grained. One popular tagset is the UPenn Treebank tagset.
- **Ambiguity**: Dealing with words that can have multiple parts of speech. Disambiguation involves resolving the ambiguities and finding the unique part of speech tag for each of the words.
- **Evaluation**: Comparing new approaches against a simple baseline.

POS tagging is a sequence labeling task. It assigns a label to each word in an input sequence. The accuracy of POS tagging algorithms can be extremely high.