Introduction to NLP:

NLP is a subset of AI deals with the interaction between computers and human (natural) languages. It helps machines understand, interpret, and generate human language. NLP applications include machine translation, sentiment analysis, text summarization, and chatbots. NLP techniques that involves analyzing the structure, meaning, and context of the data.

Uses of NLP:

1. **Language Translation**
    - Example: Google Translate, Microsoft Translator
    - Converts text or speech from one language to another.
2. **Chatbots and Virtual Assistants**
    - Example: Siri, Alexa, ChatGPT
    - Helps in human-like conversations and answering questions.
3. **Text Summarization**
    - Automatically shortens long documents while keeping the main idea.
4. **Sentiment Analysis**
    - Used in social media, reviews, etc., to understand emotions (positive, negative, neutral).
5. **Speech Recognition**
    - Converts spoken language into text.
    - Example: Voice typing, voice search.
6. **Spam Detection**
    - Filters out spam emails by understanding the content.
7. **Search Engines**
    - Improves search accuracy by understanding the user's query.
8. **Text Classification**
    - Categorizes documents, emails, or news articles automatically.

Grammar-based LM:

Grammar-based language modelling is a method in NLP where rules of grammar (like subject-verb-object, tenses, etc.) are used to build or analyze sentences. Instead of just predicting the next word based on probability it uses the structure of the language how sentences are formed to understand or generate text.

A language model is a machine learning model LM that predicts upcoming words. More formally, a language model assigns a probability to each possible next word. Language models can also assign a probability to an entire sentence.

How Grammar-Based LMs Work:

1. Define Grammar Rules:

The process begins with defining a set of grammar rules that specify how words and phrases can be combined to form valid sentences.

- Sentence → Noun Phrase + Verb Phrase
- Noun Phrase → Article + Noun
- Verb Phrase → Verb + Noun Phrase
- Article → "the", "a"
- Noun → "boy", "girl", "apple"
- Verb → "eats", "sees"

2. Parse Input Text:

The input text is parsed using the defined grammar rules. This involves identifying the different parts of the sentence and their relationships.

Ex: "The boy / eats / the apple"

3. Apply Rules for Generation:

The rules are then applied to generate new text that sticks to the grammar or to interpret the meaning of existing text.

- “ A girls sees the boy”
- “ The boy sees a girl”
- “ The girl eats the apple”

4. Evaluate Output:

The generated text is evaluated based on its grammaticality and coherence.

- "The boy eats an apple" → grammatically correct and clear meaning.
- "Eats boy the apple" → wrong word order, breaks grammar rules.

Types of Grammar Models

- Context-Free Grammar (CFG) – Most common; rules don't depend on context.
- Dependency Grammar – Focuses on relationships between words (like verb-object).
- Phrase Structure Grammar – Breaks sentences into phrases (noun phrase, verb phrase).
- Feature-based Grammar – Adds grammatical features like number, gender, tense.

Uses of Grammar-Based Models

- Syntax Checking – Ensures sentence is grammatically correct.
- Machine Translation – Helps translate sentences with correct structure.
- Speech Recognition – Understands spoken words in correct order.
- Text Generation – Generates well-formed, grammatically correct sentences.

Advantages:

- Provides high grammatical accuracy.
- Ensures structured and logical sentence formation.
- Easy to explain and debug (rules are visible and editable).
- Good for low-resource languages (when training data is limited).

Disadvantages:

- Rule creation is time-consuming and requires grammatical knowledge.
- Not suitable for slang or informal text (like social media).
- Doesn't adapt to new language patterns automatically.
- Not scalable for very large datasets or dynamic language use.

Statistical LM:

Statistical Language Modelling, is the development of probabilistic models that can predict the next word in the sequence given the words that precede it.

A statistical language model learns the probability of word occurrence based on examples of text. Simpler models may look at a context of a short sequence of words, whereas larger models may work at the level of sentences or paragraphs. Most commonly, language models operate at the level of words.

Types of statistical LM

1. N-gram:

This is one of the simplest approaches to language modelling. Here, a probability distribution for a sequence of ‘n’ is created, where ‘n’ can be any number and defines the size of the gram. There are different types of N-Gram models such as unigrams, bigrams, trigrams, etc.

Unigram:

A unigram model is the simplest type of statistical language model. It treats each word in a sentence as independent of the words before or after it. This means the model doesn't consider the order of words it only looks at the individual words and how often they appear in a large text.

Example:
Sentence: "I like ice cream"
Unigrams: "I", "like", "ice", "cream"

Bigram:

A bigram model looks at two words at a time to understand how likely a word is to follow another. It takes word order into account, unlike the unigram model. It calculates the probability of a word based on the word that came before it.

Example:
Sentence: "I like to eat "
Bigrams: "I lke", "like to", "to eat"

Trigram:

A trigram model takes it a step further by looking at three words at a time. It predicts the next word based on the previous two words. This gives it a better understanding of sentence structure and context than unigram and bigram models.

Example

Sentence: "She is reading books"
Trigrams: "She is reading", "is reading books"

2. Continuous space:

In this type of statistical model, words are arranged as a non-linear combination of weights in a neural network. The process of assigning weight to a word is known as word embedding. This type of model proves helpful in scenarios where the data set of words continues to become large and include unique words.

Regular Expressions:

- Regular expression is also called as Regex.
- Regular expression is a sequence of characters that define a search pattern.
- It is used for pattern matching or string matching.
- Especially it is used for email validation, phone number validation. So if we want password contains a lowercase alphabets and uppercase alphabets and one special character whatever will those we can added using regular expression.

Certain rules for Regex:

[abc] – it is basically means anything a, b or c

[^ abc] – it basically any character except a,b,c

[a - z] – it basically any character a-z

[A - Z] – it is basically any character A-Z

[0 - 9] – it basically any digit 0 to 9

[a-z, A-Z] – it basically a to z, A to Z

Quantifiers:

[ ]? – what are written inside or rule inside it will occur zero or one time.

[ ]+ - what are written inside it will one time or more times.

[ ]* - what are written inside it will zero or more times.

[ ]{n} – what are written inside it will occur n times.

[ ]{n, } – what are written inside it will occur n or more times.

[ ] {y,z} – what are written it will occur at least y times and less than z times.

^ caret – it tells computer that must start at beginning of the string or line.

$ dollar – it tells computer that must occur at end of the string or line.

\ backslash – it is used actual '+','-', '.'etc characters add a backslash before the character this tell computer to treat that following character as a search character.

Regex meta characters:

\s: matches any whitespace character such as space and tab.

\S: matches any non whitespace characters.

\d: matches any digital characters.

\D; matches any non digital characters.

\w: matches any word characters.

\W; matches any non word characters.

Examples:

1. **Mobile number start with 8 or 9 and total digit =10.**

   [8 9] [0 - 9]{9}

2. **First character uppercase, contains lowercase, only one digit allowed in between.**
   [A - Z] [a - z]+ [0 – 9] [a – z, A - Z] +
3. **Email ID – Aditya123@gmail.com**

   First email can be divide into three parts
   Part1 – Aditya123
   Part2 – gmail
   Part3 – com, in

   [A -Z, a – z,0 – 9, _ \-] + [@] [a – z] + [\.] [a – z]{2,3}

Finite State Automate:

A Finite State Automaton (FSA), also known as a Finite State Machine (FSM), is a computational model used in computer science to represent and control execution flow. It consists of a finite number of states, transitions between those states, and actions.
Finite automata come in deterministic (DFA) and non-deterministic (NFA), They are widely used in text processing, compilers, and network protocols.

How it works

- Starts in one initial state (called the start state).
- It reads input symbols one at a time (like letters or numbers).
- Based on the current state and the input symbol, it moves to another state using a rule (called a transition).
- After reading all the input, if it ends in a final state (accepting state), the input is said to be accepted.
- Otherwise, the input is rejected.

Formal Definition:

A **Finite State Automaton** is defined as a 5-tuple:

$$FSA = (Q,\Sigma ,q0,F, \delta)$$

Where:

- **Q**: A finite set of states
- **Σ (Sigma)**: A finite set of input symbol
- $\mathbf{q_0}$: The initial state
- **F**: A set of final states ($F \subseteq Q$)
- **δ (delta)**: A transition function $\delta: Q \times \Sigma \rightarrow Q$

Types of Finite Automata

There are two types of finite automata:

- Deterministic Finite Automata (DFA)
- Non-Deterministic Finite Automata (NFA)

Deterministic Finite Automata (DFA):

A DFA is represented as {Q, Σ, q, F, δ}. In DFA, for each input symbol, the machine transitions to one and only one state. DFA does not allow any null

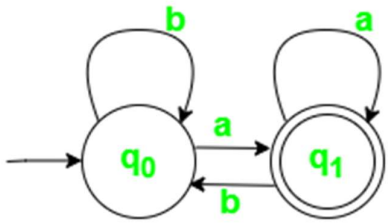
transitions, meaning every state must have a transition defined for every input symbol.

- In DFA given the current state we know that next state will be.
- It has only one unique next state.
- It has no choice.
- It is a simple and easy to design.

Formal Definition:

DFA consists of 5 tuples: {Q, Σ, q, F, δ}.

- **Q:** set of all states.
- **Σ:** set of input symbols.
- **q:** Initial state.
- **F:** set of final state (F ⊆ Q)
- **δ:** Transition Function, defined as δ: Q X Σ --> Q.



Transition table

State\	a	b
q0	q1	q0
q1	q1	q0

Non-Deterministic Finite Automata (NFA):

A Nondeterministic Finite Automaton (NFA) is a type of finite state machine used in computational theory and natural language processing. Unlike a DFA, an NFA allows multiple possible transitions for a given state and input symbol, including transitions without any input (called ε-transitions).

- It can transition to multiple states for the same input.
- It allows null (ϵ) moves, where the machine can change states without consuming any input. The next state may be chosen random.

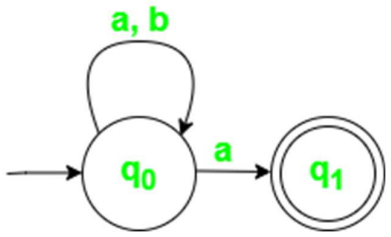
Formal Definition:

An NFA is a 5-tuple:

$$NFA= \{Q, \Sigma, q, F, \delta\}$$

Where:

- **Q**: A finite set of states.
- **Σ (Sigma)**: A finite set of input symbols (alphabet)
- **q**: The initial state.
- **F**: A set of accepting (final) states ($F \subseteq Q$)
- **δ (delta)**: A transition function: $\delta: Q \times (\Sigma \cup \{\varepsilon\}) \rightarrow 2Q$



Transition table

State	a	b
q0	{q0,q1}	q0
q1	φ	φ

English morphology:

Morphology is the study of words morpheme are the minimal units of words that have a meaning and cannot be sub divided further. There are mainly 2 types free and bound. Free morpheme occur alone and bound morpheme must occur with another morpheme.

For ex: free morpheme is “bad” and on ex of bound morpheme is “ly” it is bound although it has meaning it cannot stand alone. It must be attached to another morpheme to produce a word.

Free morpheme: bad

Bound morpheme: ly

Word: badly

When we talk about words, there are two groups lexical or content and function words or grammatical.

Lexical or content: There are open class words and include nouns, verbs, adjectives and adverbs. New word can regularly be added to this group.

Function word or grammatical: There are closed word are conjunctions, prepositions, article and pronouns and new words cannot be (or) rarely added to this class.

Affixes are often the bound morpheme, this group include prefix, suffix, infixes and circumfix.

Prefixes are added to the beginning of another morphemes

Suffixes are added to the end. Following are ex of each of there

Prefix: re- added to produce redo.

Suffix: -or added to edit produce editor.

Infix: -um- added to fikas produce fumikas in Bontoc.

Circumfix: ge--- t is added to lieb to produce geliebt in German.

There are two categories of affixes: derivational and inflectional the main difference between the two is that derivational affix are added to morpheme to form new words that may or may not be the some part of speech and inflectional affixal are added to the end of an existing word for purely grammatical reasons.

English morphemes

A: Free

1.open class.

2. closed class.

B: Bound

1. Affix
    a. Derivational
    b. Inflectional.
2. Root

Transducers or lexicon and rules:

A lexical transducer is a special type of finite-state machine that is used in language processing to convert inflected or surface forms of words into their basic or lexical forms, and vice versa. It was first introduced by Korhonen, Kaplan, and Zaenen in 1992. In simple terms, it acts like a translator between how a word appears in a sentence (surface form) and its base form with grammatical tags (lexical form). This is useful in morphological analysis and generation.

Lexicon:

The lexicon is a list of base words (root forms) along with information about their meanings and grammatical behaviour. Each word in the lexicon has a canonical form and a set of tags that describe its grammatical features like tense, number, person, etc.

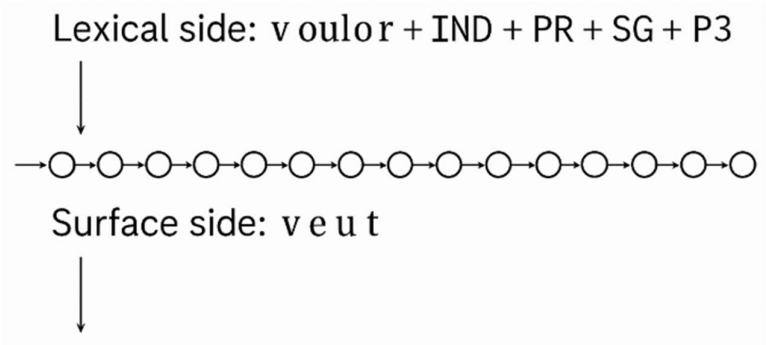
Rules:

Rules define how words change from their base (lexical) form into their surface form. These rules are often morphological, like adding suffixes for tense or changing spellings for conjugation.

Example (French):

For example, the French surface word **"veut"** (which means "wants") can be analysed using a lexical transducer and matched to its lexical form: **vouloir + IND + PR + SG + P3**

- **vouloir** = verb root (to want)
- **IND** = indicative mood
- **PR** = present tense
- **SG** = singular
- **P3** = third person



- Circles represent the states and arcs represents the pair of symbols: a lexical symbol and a surface symbol Sometimes the lexical and surface symbols are the same (e.g., **v:v**), Sometimes they differ (e.g., **o:e**).
- Finite state transducers are bidirectional. The same transducer can be used for analysis (veut -> voulor +IND + PR+SG+P3) as well as generation (voulor +IND + PR+SG+P3 - > veut).
- Analysis and generation different only with the respect to the choices of the input side (surface or lexical).

Tokenization:

Tokenization is a fundamental step in Natural Language Processing (NLP). It involves dividing a Textual input into smaller units known as tokens. These tokens can be in the form of words, characters, sub-words, or sentences. used to convert unstructured text into a structured format that machines can easily analyze and understand.

- Involves dividing a string or text into a list of smaller units known as tokens.
- A tokenizer breaks unstructured text into smaller parts, treating each part as a separate piece of information.
- Tokens: Words or Sub-words in the context of natural language processing. Example: A word is a token in a sentence, A character is a token in a word, etc.
- Application: Multiple NLP tasks, text processing, language modelling, and machine translation

Types of Tokenization: Tokenization can be classified into several types based on how the text is segmented

1. Word Tokenization:

Word tokenization is the most commonly used method where text is divided into individual words. It works well for languages with clear word boundaries, like English.

Another word of word tokenization is white space tokenization

For example

Input: ["Machine learning is fascinating"]

Output when tokenized by word: ["Machine", "learning", "is", "fascinating"]

2. Character Tokenization:

In Character Tokenization, the textual data is split and converted to a sequence of individual characters. This is beneficial for tasks that require a detailed analysis, such as spelling correction or for tasks with unclear boundaries. It can also be useful for modelling character-level language**.**

For example

Input ["You are helpful"]

Output when tokenized by characters: ["Y", "o", "u", " ", "a", "r", "e", " ", "h", "e", "l", "p", "f", "u", "l"]

3. Punctuation-based Tokenization:

Punctuation-based tokenization splits text into tokens by separating words and punctuation marks. This method treats punctuation (like commas, periods, question marks, etc.) as individual tokens instead of combining them

with words. It helps in preserving the meaning and structure of a sentence during text processing, which is useful in tasks like sentiment analysis and grammar correction.

For example:

Input ["Hello, how are you?"]

Output ["Hello", ",", "how", "are", "you", "?"]

4. **Sub word Tokenization:**

   This strikes a balance between word and character tokenization by breaking down text into units that are larger than a single character but smaller than a full word. This is useful when dealing with morphologically rich languages or rare words.

   For example

   Time table - ["Time", "table"]

   Rain coat - ["Rain", "coat"]

   Run way - ["Run", "way"]

   Sub-word tokenization helps to handle out-of-vocabulary words in NLP tasks and for languages that form words by combining smaller units.

5. **Sentence Tokenization:**

   Sentence tokenization is also a common technique used to make a division of paragraphs or large set of sentences into separated sentences as tokens. This is useful for tasks requiring individual sentence analysis or processing

   For example:

   Input ["He is here. She left."]

   Output ["He is here.", "She left."]

Detecting and correcting spelling errors:

Spelling correction in natural language processing involves detecting and correcting misspelled words in text. It is a process of detecting and some times providing suggestions for incorrectly spelled words in the text. In computing spell checker is an application program that flags word in a document that may not be spelled correctly.

This is typically achieved through combination of techniques, including dictionary lookups, error model-based approaches, and machine learning algorithms. The goal is to identify and correct errors like real-word errors (misused words) and non-word errors (typos).

- Real-word errors: Those error words that are acceptable words in the dictionary.
- Non-word errors: Those error words that are cannot be found in the dictionary.

Detection of spelling errors:

1. **Dictionary lookup technique:** In this dictionary lookup technique is used which checks every word of input text for its presence in dictionary. If that word present in the dictionary then it is a correct word otherwise it is put into the list of error words.
2. **Language modelling:** More advanced techniques involve building language models e.g., N-gram language models also widely used in error detection. These models evaluate the probability of a word appearing a given context by analyzing the frequency of a word sequences. If a sequence is statistically unlikely, it may indicate a real-word error.
3. **Part-of-speech (POS):** part-of-speech tagging helps find grammar mistakes by checking how words are used in a sentence. For example, if a verd is used where a noun should be, it can be flagged as a error.

Correction of spelling errors:

1. **Minimum edit distance:** This technique suggests replacement words that are closest in spelling to the detected error, based on the fewest number of edits needed to reach a valid word. While simple, it is very effective for common typographical mistakes.
2. **Contextual spelling correction:** Contextual spelling correction takes this a step further by using advanced language models like BERT or GPT. These models consider the surrounding words and suggest corrections that make sense in the given context, which is especially useful for real-word errors (e.g., “their” vs. “there”).
3. **Similarity key technique:** The Similarity Key Technique is a method where each word (or string) is changed into a special code, called a key. Words that are spelled in a similar way will get similar keys. This helps in finding and correcting spelling mistakes, because if two words have similar keys, they are likely to be similar or related.
4. **Rule-based correction:** Rule-based correction methods use fixed grammar rules to correct common mistakes. This method is often used in specific fields where the types of errors are already known.

Tools and Libraries:

TextBlod: A python library that provides a simple API for common NLP tasks, including spell checking.

SpellChecker: Another python library that focuses specifically on spelling correction.

SymSpell: An efficient algorithm for spelling correction, available in python.

Spark NLP: A library for NLP tasks, including spell checking built on Apache Spark.

Minimum edit distance:

Minimum Edit Distance (MED) is a technique used in Natural Language Processing (NLP) to find how similar two strings (words or sentences) are. It measures the minimum number of operations required to convert one string into another.

Minimum edit distance between two strings str1 and str2 is defined as the number of insert/ delete/ substitute operations required to transforms str1 and str2.

Common Operations:

1. **Insertion** – Insert any character before or after any index.
2. **Deletion** – Remove a character.
3. **Substitution** – Replace one character with another

Example: str1 = “ab”, str2 = “abc” the making on insert operation od char (c) on str1 transforms str l into str 2. Therefore edit distance between str 1 and str 2 is 1.

Input: s1 = "geek", s2 = "gesek"
Output: 1
Explanation: We can convert s1 into s2 by inserting an 's' between two consecutive 'e' in s1.

Input: s1 = "gfg", s2 = "gfg"
Output: 0
Explanation: Both strings are same.

Input: s1 = "abcd", s2 = "bcfe"
Output: 3
Explanation: We can convert s1 into s2 by removing 'a', replacing 'd' with 'f' and inserting 'e' at the end.

N-grams:

N-grams are defined as the contiguous sequence of n items that can be extracted 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.

N-grams are classified into different types depending on the value that n takes. When n=1, it is said to be a unigram. When n=2, it is said to be a bigram. When n=3, it is said to be a trigram.

Unsmoothed N-gram:

Unsmoothed n-gram models calculate the probability of word sequences based only on their observed frequencies in the training data

Unigram: A model that calculates the probability of a single word based on how often it appears in the corpus.

$$P(a) = \frac{\text{Count}(a)}{\text{Total number of words in the corpus}}$$

Data: "The dog barks. The cat sleeps. The dog runs. The cat jumps."

Total words =12

Vocabulary: {"The", "dog", "barks", "cat", "sleeps", "runs", "jumps"} = 7

P (dog) = 2/10 = 0.1666

P (Apple) = 0/12 = 0 (zero probability)

Bi gram:

In the bigram model, we calculate the probability of a word b given the previous word a. It assumes that the current word depends only on the word immediately before it.

$$P(b \mid a) = \frac{\text{Count}(a, b)}{\text{Count}(a)}$$

Data: "The dog barks. The cat sleeps. The dog runs. The cat jumps."

Total words =12

Vocabulary: {"the dog", "dog barks", "barks the", "the cat", "cat sleeps", "sleeps the", "dog runs", "runs the", "cat jumps"} = 9

P (dog | the) = count (The, dog) / count (The) = 2/4 = 0.5

P (jumps | dog) = 0/2 = 0 (zero probability)

Tri gram:

In the trigram model, we calculate the probability of a word c based on the two previous words, a and b. It assumes that a word depends on the last two words only.

$$P(c \mid a, b) = \frac{\text{Count}(a, b, c)}{\text{Count}(a, b)}$$

Data : “The dog barks. The cat sleeps. The dog runs. The cat jumps.”

Total words =12

Vocabulary: {“The dog barks”, “dog barks the, “barks the cat”, “The cat sleeps”, “cat sleeps the”, “sleeps the dog”, “The dog runs”, “dog runs the”, “runs the cat”, “The cat jumps”} = 10

P (barks | the, dog) = Count (the, dog, barks) / Count (the, dog) = ½ = 0.5

P (sleeps | the, dog) = 0/2 = 0 (zero probability)

These models are simple and useful for understanding basic language patterns. However, they have a major drawback: if an n-gram has never appeared in the training data, its probability is zero, even if it’s a valid phrase.

Therefore, unsmoothed n-grams are mainly used for educational purposes or as a base model, while practical systems apply smoothing techniques to handle unseen n-grams and improve accuracy.

Smoothing:

Smoothing in Natural Language Processing (NLP) is a method used to fix the problem of getting zero probability for word combinations that the model hasn't seen before. When a language model comes across a sentence or phrase that wasn't in its training data, it gives it a probability of zero, which can cause problems in tasks like text prediction or machine translation.

Smoothing helps by slightly adjusting the probabilities so that even unknown or new word combinations still get a small chance, instead of being completely ignored. Without smoothing, unseen word combinations are assigned a probability of zero, which can cause the entire sentence probability to become zero. Overall, smoothing is essential for building accurate and reliable NLP systems.

Types of smoothing

1. **Add one smoothing (Laplace):**

The easiest way to apply smoothing is to add one to all n-gram counts before converting them into probabilities. This means even unseen word combinations will get a count of 1 instead of 0. This method is called Laplace smoothing.

While it's not very effective for modern language models, it helps us understand how smoothing works and is still useful for simpler tasks like text classification.

$$P(b \mid a) = \frac{\text{count}(a, b) + 1}{\text{count}(a) + V}$$

Count(a,b) = how often "a b" appears together

Count(a) = how often "a" appears as the first word in a Unigram

V = vocabulary size (total unique words)

Example:

Data: “The dog barks. The cat sleeps. The dog runs. The cat jumps.”

Total words =12

Vocabulary: {“the dog”, “dog barks”, “barks the”, “the cat”, “cat sleeps”, “sleeps the”, “dog runs”, “runs the”, “cat jumps”} = 9

$$P(\text{"jumps"} \mid \text{"dog"}) = \frac{0+1}{2+9} = \frac{1}{11} \approx 0.0909$$

2. Add-K smoothing:

Add-k smoothing is like add-one smoothing, but instead of adding 1 to each count, we add a smaller value like 0.5 or 0.1. This way, we move less probability to unseen word pairs. The value of k can be chosen by testing different values on a development set. While add-k smoothing works well for some tasks like text classification, it is not very effective for language modelling.

$$P(b \mid a) = \frac{\text{count}(a,b) + k}{\text{count}(a) + k \cdot V}$$

Count(a,b) = how often "a b" appears together

Count(a) = how often "a" appears as the first word in a Unigram

V = vocabulary size (total unique words)

k = a small positive constant (e.g., 0.1 or 0.5)

Example:

$$P(\text{jumps} \mid \text{dog}) = \frac{0+0.5}{2+0.5 \cdot 7} = \frac{0.5}{5.5} \approx \boxed{0.091}$$

Interpolation:

Sometimes, a language model doesn't find a 3-word combination (trigram) in the training data. Instead of giving zero probability, we can look at shorter combinations like 2-word (bigram) or even single words (unigram). This method is called interpolation.

In interpolation, we combine the probabilities of the trigram, bigram, and unigram using weights. This helps the model make better guesses by mixing different levels of context instead of relying on just one.

$$P(b \mid a) = \lambda_1 \cdot \frac{\text{count}(a,b)}{\text{count}(a)} + \lambda_2 \cdot \frac{\text{count}(b)}{N}$$

Count(a,b) = how often "a b" appears together

Count(a) = how often "a" appears as the first word in a Unigram

Count(b) = how often "b" appears as the first word in a Unigram

P_{ML} = maximum likelihood estimate (regular probability from counts)

$\lambda 1+\lambda 2 = 1$ (they are weights)

N = Total number of tokens (words) in the corpus

Example:

$$P(\text{jumps} \mid \text{dog}) = 0.7 \cdot \frac{0}{2} + 0.3 \cdot \frac{1}{12} = 0 + 0.025 = 0.025$$

Interpolation and backoff:

Interpolation:

Sometimes, a language model doesn't find a 3-word combination (trigram) in the training data. Instead of giving zero probability, we can look at shorter combinations like 2-word (bigram) or even single words (unigram). This method is called interpolation.

In interpolation, we combine the probabilities of the trigram, bigram, and unigram using weights. This helps the model make better guesses by mixing different levels of context instead of relying on just one.

$$P(b \mid a) = \lambda_1 \cdot \frac{\text{count}(a,b)}{\text{count}(a)} + \lambda_2 \cdot \frac{\text{count}(b)}{N}$$

Count(a,b) = how often "a b" appears together

Count(a) = how often "a" appears as the first word in a Unigram

Count(b) = how often "b" appears as the first word in a Unigram

P_{ML} = maximum likelihood estimate (regular probability from counts)

$\lambda 1+\lambda 2 = 1$ (they are weights)

N = Total number of tokens (words) in the corpus

Example:

Data: “The dog barks. The cat sleeps. The dog runs. The cat jumps.”

Total words =12

Vocabulary: {“the dog”, “dog barks”, “barks the”, “the cat”, “cat sleeps”, “sleeps the”, “dog runs”, “runs the”, “cat jumps”} = 9

$$P(\text{jumps} \mid \text{dog}) = 0.7 \cdot \frac{0}{2} + 0.3 \cdot \frac{1}{12} = 0 + 0.025 = 0.025$$

Backoff:

Backoff is a smoothing technique used when an n-gram (like a trigram) has zero count in the data. In such cases, the model "backs off" to a lower-order n-gram (like a bigram or unigram) until it finds a match.

- If the trigram exists in the data, we use it.
- If not, we "back off" and use the bigram.

- If the bigram is also not found, we finally back off to the unigram.

We only use a lower-order n-gram when the higher-order one is missing or has zero count.

To ensure proper probability distribution, traditional backoff methods apply discounting to higher-order n-grams, saving some probability for lower orders. However, a simpler method called Stupid Backoff skips discounting. Instead, if a higher-order n-gram is missing, it directly backs off to a lower-order n-gram and multiplies its score by a fixed weight (like 0.4). While this doesn't form a true probability distribution, it's fast and works well for large datasets.

$$\text{score}(b \mid a) = \begin{cases} \frac{\text{count}(a,b)}{\text{count}(a)}, & \text{if count}(a,b) > 0 \\ \lambda \cdot \frac{\text{count}(b)}{N}, & \text{otherwise} \end{cases}$$

λ= fixed weight (e.g., 0.4)

N = total number of words in the corpus

Example:

$$\text{score}(\text{jumps} \mid \text{dog}) = 0.4 \cdot \frac{1}{12} = 0.033$$

Word classes:

In Natural Language Processing (NLP), word classes (also known as parts of speech (POS)) refer to the grammatical categories that words belong to based on their roles in sentences. Understanding word classes is fundamental in many NLP tasks like POS tagging, syntactic parsing, information extraction, and machine translation.

1. Noun (NN)

Nouns are words that name people, places, things, or ideas. Ex: cat, India, house.
Example: *Ravi went to the* ***market****.*

2. Pronoun (PRP)

Pronouns replace nouns to avoid repetition. Ex: he, she, they, them.
Example: ***He*** *is a doctor.*

3. Verb (VB)

Verbs describe an action, event, or state of being. Ex: run, write, sit, walk.
Example: *She* ***runs*** *every morning.*

4. Adjective (JJ)

Adjectives describe or modify nouns to give more information. Ex: beautiful, large, smart.
Example: *He is a* ***smart*** *boy.*

5. Adverb (RB)

Adverbs modify verbs, adjectives, often showing manner, time. Ex: quickly, slowly, silently.
Example: *She speaks* ***slowly****.*

6. Preposition (IN)

Prepositions show the relationship between a noun/pronoun and other words. Ex: in, on at, under.
Example: *The book is* ***on*** *the table.*

7. Conjunction (CC)

Conjunctions join words, phrases, or clauses. Ex: and, but, or.
Example: *I want tea* ***and*** *snacks.*

8. Interjection (UH)

Interjections express sudden emotions or feelings. Ex: wow, hey, oh.
Example: *"**Wow!** That's amazing."*

9. Determiner (DT)

Determiners are used before nouns to specify quantity or reference. Ex: the, an, a, this, those.
Example: ***This** apple is sweet.*

10. Numeral (CD)

Numerals indicate numbers or order. Ex: one, two, 3rd.
Example: *She has **two** dogs.*

Part of speech tagging:

Parts of Speech (PoS) tagging is a core task in NLP, It gives each word a grammatical category such as nouns, verbs, adjectives and adverbs. Through better understanding of phrase structure and semantics, this technique makes it possible for machines to study human language more accurately.

PoS tagging is essential in many NLP applications like machine translation, sentiment analysis and information retrieval. It serves as a link between language and machine understanding, enabling the creation of complex language processing systems.

POS Tag	**Meaning**	**Example**
NN	Noun (singular)	*dog, school, computer*
NNS	Noun (plural)	*dogs, schools*
VB	Verb (base)	*go, run, eat*
VBD	Verb (past)	*went, ran, ate*
VBG	Verb (gerund)	*going, running, eating*
JJ	Adjective	*brown, beautiful, quick*
RB	Adverb	*quickly, very*
PRP	Pronoun	*he, she, it, they*
IN	Preposition	*in, on, at, by*
DT	Determiner	*the, a, an, this*
CC	Coordinating Conjunction	*and, but, or*
UH	Interjection	*wow, oh, hey!*

Example: "The quick brown fox jumps over the lazy dog."

- "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 (VB)
- "over" is tagged as preposition (IN)
- "the" is tagged as determiner (DT)
- "lazy" is tagged as adjective (JJ)
- "dog" is tagged as noun (NN)

Workflow of POS Tagging in NLP

1. **Tokenization:**
   The input text is split into individual words or subwords, enabling word-level analysis.
2. **Loading a language model:** Tools like NLTK requires a pre-trained language model to perform POS tagging. These models are trained on large datasets and provide insights into the grammatical rules and structure of the language.
3. **Text Preprocessing**: The text is then cleaned to improve accuracy. Common preprocessing steps include converting text to lowercase, removing special characters and eliminating irrelevant content.
4. **Syntactic Analysis:** The sentence is parsed to understand grammatical roles, helping prepare for accurate POS assignment.
5. **POS tagging:** each token is labelled with its appropriate part of speech using context and syntax.
6. **Result evaluation:** The output is checked for accuracy and any tagging errors are corrected if needed.

Rule based POS tagging:

Rule-based POS tagging assigns grammatical tags to words using a predefined set of rules, as opposed to machine learning-based methods that require training on annotated corpora. These rules are crafted based on morphological features (like word endings) and syntactic context, making the approach highly interpretable and transparent.

Example

a rule might specify that words ending in “-tion” or “-ment” should be tagged as nouns, based on common suffix patterns found in English.

- **Rule:** Assign the POS tag "Noun" to words ending in -tion or -ment.
- **Text:** "Her dedication and commitment inspired the entire management team."

Tagged output:

- "Her" is tagged as Pronoun (PRP)
- "dedication" is tagged as Noun (NN)
- "and" is tagged as Conjunction (CC)
- "commitment" is tagged as Noun (NN)
- "inspired" is tagged as Verb (VB)
- "the" is tagged as Determiner (DT)
- "entire" is tagged as Adjective (JJ)
- "management" is tagged as Noun (NN)
- "team" is tagged as Noun (NN)

In this case, the rule-based tagger correctly identifies "dedication," "commitment," and "management" as nouns by applying suffix-based rule. Even though it's simple, this example shows how rule-based systems can understand many language patterns by using clear and organized rules.

Stochastic and Transformation-based tagging:

Transformation Based tagging

Transformation-Based Tagging is a method used to improve part-of-speech tags by applying correction rules step by step. It doesn't depend on fixed grammar rules like rule-based taggers or probabilities like statistical taggers. Instead, it starts with basic tags and then makes corrections using rules that look at the context of each word.

Example rule: "If a word is tagged as a verb, but it comes after 'the', change it to a noun"

Example Sentence:

"The walk was long "

Initial tags (before rule is applied):

The – determiner (DT)

Walk – verb (VB)

was – verb (VB)

long – adverb (RB)

Transformation rule applied:

If a word is tagged as VB (verb) but comes after "the", change it to NN (noun).

Final Tags (after rule applied):

The – determiner (DT)

Walk – noun (NN)

Was – verb (VB)

Long – adverb (RB)

Example sentence 2:

"Can birds fly?"

Initial tags (before rule is applied):

Can – modal verb (MD)

Birds – noun (NN)

Fly – noun (NN)

Transformation rule applied:

If a word is tagged as NN (noun) but comes at the end of a question starting with "Can", change it to VB (verb)

Final Tags (after rule applied):

Can – model verb (MD)

Birds – noun (NN)

Fly – verb (VB)

Stochastic POS tagging:

Stochastic POS tagging (also called statistical tagging) is a method in computational Grammatical Science that uses probability to decide the correct part of speech for each word in a sentence, like noun, verb, or adjective. Unlike rule-based methods that use fixed grammar rules, statistical tagging learns from examples. It studies large amounts of already tagged text and finds patterns using machine learning.

These models calculate the chance of a tag being correct for a word based on the words around it. This helps them handle confusing cases and understand complex grammar.

Issues in POS tagging:

1. Ambiguity:

Ambiguity happens when a word can have more than one part of speech, depending on the context. This makes it hard for the tagger to choose the correct tag.

Example:

- **“book”**
    - Verb: "Can you **book** a ticket?"
    - Noun: "I read a **book**."
- **“play”**
    - Noun: "The **play** was great."
    - Verb: "They **play** outside."

The tagger needs to look at the surrounding words to decide the correct tag.

2. Out of vocabulary:

Out-of-Vocabulary (OOV) words are words that do not appear in the training data of a POS tagger. These can include new terms, foreign words, slang, or names that the model has never seen before.

When a tagger encounters an OOV word, it struggles to assign the correct part of speech, because it has no past examples to learn from. This often leads to incorrect tagging, which can affect the accuracy of downstream NLP tasks.

Example: “He created a new app called **Zyntora**.”

- OOV Word: “Zyntora” (a made-up product name)
- Issue: The tagger may not know if it's a noun, a verb, or something else.

3. Errors in rule and statistical models:

Rule-based POS taggers can make errors if their rules are too strict or don’t cover all cases. They often fail with new word usages. Statistical taggers make mistakes when they rely too much on training data. If a word is rare or has multiple meanings, the model may choose the wrong tag.

Example:

Sentence: "They can fish in the lake."

- **Rule-based error:** Tags **"can"** as a **verb**, and **"fish"** as a **noun**
- **Statistical error:** Tags **"can"** as a **noun** (like a tin can), and **"fish"** as a **verb** (like swimming).

Correct tags:

- **"can"** – Modal verb
- **"fish"** – Main verb

4. Multi word expressions:

MWEs are phrases made of two or more words that act as one meaning, like idioms or phrasal verbs. Their meaning is often not clear from the individual words. This makes POS tagging harder, because tagging word by word may miss the actual meaning of the whole phrase.

Example: **Sentence:** "He kicked the bucket."

- **He** – Pronoun (PRP)
- **kicked** – Verb (VB)
- **the** – Determiner (DT)
- **bucket** – Noun (NN)

Literally, this makes sense. But **"kick the bucket"** is an idiom meaning *"to die."*

So, the tagger should ideally treat **"kick the bucket"** as a **single verbal expression** rather than tagging each word individually with its literal meaning.

Hidden Markov and maximum entropy model:

In Natural Language Processing (NLP), we use different types of models to understand and process human language. Two important models are the Hidden Markov Model (HMM) and the Maximum Entropy Model (MaxEnt). These models help computers to do tasks like part-of-speech (POS) tagging, named entity recognition, speech recognition, and many others.

Hidden Markov Model (HMM):

Hidden Markov Model is a statistical model used to predict hidden patterns or tags based on observed data, like words in a sentence. It is commonly used in NLP tasks such as POS tagging, where the system guesses the correct tag sequence using probabilities.

- It tries to find the **best tag** (like noun or verb) for each word.
- It uses the idea that the **next tag depends on the previous one**.
- It looks at how often a tag follows another tag.
- Useful for tasks like **POS tagging**, **name finding**, and **speech recognition**.
- Easy to understand, but may not work well with complex sentences.

Example:

Sentence **"Birds fly"**

Observed words (input): "Birds", "fly"

Possible tags: Noun (N), Verb (V)
We check all tag combinations:

- Noun → Verb (Birds/N, fly/V)
- Noun → Noun (Birds/N, fly/N)

HMM calculates which tag path has the **highest total probability**.

The best option will be: **Birds/Noun → fly/Verb**

Advantages:

- Simple and easy to implement.

- Works well with small datasets.

Disadvantages:

- Makes strong assumptions (only looks at the previous tag).
- Cannot handle complex context or long-range information.

Maximum Entropy Model:

The Maximum Entropy model is used to predict the most suitable tag or label for a word based on useful clues or features from the sentence. It works on the idea of making no extra assumptions — only using the information available.

- It is a model that directly tries to guess the correct tag or label for a word.
- It looks at useful clues like the word itself, the word before it, capital letters, endings like "-ing" or "-ed", etc.
- It learns which clues are important by using training data.
- It is used in tasks like part-of-speech tagging, text classification, and f Named Entity Recognition (NER).
- It is more accurate and flexible, especially when we have more data to train it.

Example: Sentence: **"Apple is sweet"**

- For the word "Apple", MaxEnt looks at features like: Is the word capitalized? Is it the first word in the sentence?
- Based on these features, it might predict the tag: **Proper Noun (NNP)**
- For "sweet", it might check if it comes after "is", and guess: **Adjective (JJ)**

Advantages:

- Can use **many types of features**.
- More **flexible and accurate** in many cases.

Disadvantages:

- Needs more data and feature design.
- Takes longer to train than HMM.