UNIT4

I. Natural Language Processing (NLP) & Various Steps Followed In It:

Natural Language Processing (NLP) is a field within artificial intelligence focused on enabling computers to understand, interpret, and generate human language. NLP combines computational linguistics with machine learning to process language in a way that captures its underlying meaning and context. Here are the key steps in NLP:

1. Text Preprocessing

This step prepares the text by cleaning and organizing it:

- **Tokenization**: Breaks down text into smaller parts, like words or sentences, so the computer can process each piece separately.
- **Lowercasing**: Converts everything to lowercase to avoid treating "Apple" and "apple" as different words.
- **Removing Punctuation and Stop Words:** Cleans the text by removing symbols (like commas) and common words (like "the" or "and") that don't add much meaning.
- **Stemming and Lemmatization**: Shortens words to their root forms, like turning "running" into "run" or "children" into "child," to group similar words together.

2. Text Representation

Here, the cleaned text is turned into numbers that the computer can understand:

- **Bag of Words (BoW):** Counts how many times each word appears in the text, without considering the order of words.
- **TF-IDF:** Highlights important words by giving a higher score to words that are frequent in one text but rare in others.
- **Word Embeddings:** Converts words into number codes that capture their meaning and relationships (for example, making “king” and “queen” appear similar in the way they are represented).

3. <u>**Feature Extraction**</u>

   This step focuses on finding specific details in the text:

   - **Named Entity Recognition (NER):** Spots names, places, organizations, dates, and other important items in the text.
   - **Part-of-Speech (POS) Tagging:** Identifies each word’s role in a sentence, like whether it’s a noun, verb, adjective, etc.
   - **Sentiment Analysis:** Tries to detect the feeling or mood behind the words, such as whether the text is positive, negative, or neutral.

4. <u>**Model Training and Selection**</u>

   The next step is choosing a model that will learn from the text and make predictions:

   This could involve using simple algorithms like Naive Bayes for basic tasks or advanced neural networks like BERT for more complex tasks. The model is fed with examples of text and learns to understand patterns.

5. <u>**Evaluation and Optimization**</u>

   This step checks how well the model performs:

The model’s predictions are tested against real data, and adjustments are made to improve accuracy. Metrics like accuracy and precision are used to measure success and guide improvements.

6. **Post-Processing and Deployment**
   Finally, the model is prepared for real-world use:

   - **Error Analysis:** Reviewing where the model gets things wrong to improve its future performance.
   - **Integration:** Embedding the model into applications like chatbots, translation tools, or search engines to make it useful for users.

II. Game Theory

Game Theory is a branch of mathematics that studies strategies in situations where multiple people, or "players," make decisions that affect each other. The goal is to find the best strategy for each player, especially when they have conflicting interests or shared goals. In AI (Artificial Intelligence), game theory is important because it helps AI systems make decisions in complex environments involving other agents, whether they are people, robots, or other AI systems. Game theory is commonly used in economics, political science, cybersecurity, and social science, where decision-making involves multiple parties.

Importance of Game Theory in AI

In AI, game theory helps machines make intelligent choices when interacting with other machines or humans. For example, it is used in multi-agent systems, where different agents might have different goals but need to work together or compete. Game theory also helps in predicting opponent moves, negotiations, resource management, and designing algorithms for competitive scenarios, such as trading or gaming.

<u>**The Prisoner's Dilemma**</u>

The Prisoner's Dilemma is a classic example in game theory that illustrates why two rational individuals might not cooperate, even if it's in their best interest to do so.

How It Works:

Imagine two prisoners are arrested for a crime and interrogated separately. They have two choices: either Cooperate (stay silent) or Defect (betray the other).

- If both prisoners cooperate and stay silent, they each get a light sentence of, say, 1 year.
- If one defects and the other cooperates, the defector goes free, and the one who cooperated gets a heavy sentence of 5 years.
- If both defect, they each get a moderate sentence of 3 years.

The Dilemma:

Each prisoner thinks selfishly:

- If one defects while the other cooperates, the defector goes free, which seems like the best option individually.
- But if both defect, they each end up with a worse outcome than if they had cooperated.

Even though cooperating leads to the best combined result, the lack of trust and fear of betrayal often leads both to defect, resulting in a less favorable outcome for both. This demonstrates how difficult it can be to achieve cooperation, even when it benefits all parties. In AI, the Prisoner's Dilemma helps in designing systems that can predict behavior and decide when to cooperate or compete.

III. Game Playing Strategies/Algorithms:

Min-Max and **Alpha-Beta Pruning** are two important algorithms in game theory and artificial intelligence, especially for games like chess or tic-tac-toe, where two players compete against each other. These algorithms help AI make optimal moves by examining possible moves and outcomes.

Min-Max Algorithm:

The **Min-Max algorithm** is a decision-making algorithm used in two-player games. The goal is for one player (Max) to maximize their score, while the other player (Min) tries to minimize it. The algorithm works by creating a tree of possible moves and outcomes, evaluating each move from the perspective of maximizing or minimizing scores, and selecting the best move for the player.

How It Works:

- **Generate a Game Tree:** Starting from the current position, generate all possible moves up to a certain depth. Each "node" in the tree represents a possible game state.
- **Assign Scores to Terminal Nodes:** At the end of each possible game path, assign a score based on how good or bad the position is for the player. For instance, in tic-tac-toe:
  - ➢ +1 if Max wins,
  - ➢ -1 if Min wins,
  - ➢ 0 if it's a draw.
  - ➢

Propagate Scores Backward:

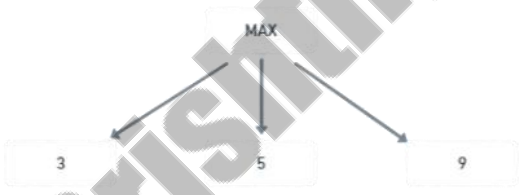
For Max nodes (Max's turn), choose the maximum score among child nodes, as Max tries to maximize their score.

For Min nodes (Min's turn), choose the minimum score among child nodes, as Min tries to minimize the score.

Choose the Best Move:
Starting from the top of the tree, select the move that leads to the best score, assuming both players play optimally.

Example:
Suppose we have a simple game with three moves available for Max at the start. Each move leads to another set of moves for Min, who then makes decisions that lead to different end-game states with scores.

If the tree looks like this:



Max would choose **9**, as it's the highest score. In a deeper tree, the Min-Max algorithm would go several levels down, evaluating all moves and their results, selecting the best overall path for Max.

Alpha-Beta Pruning Algorithm:

Alpha-Beta Pruning is an optimization of the Min-Max algorithm. It "prunes" or cuts off branches in the game tree that don't need to be explored because they won't affect the final decision. This makes the algorithm much faster.

How It Works:

1. **Introduce Alpha and Beta:**
    - Alpha is the best score Max can guarantee so far.
    - Beta is the best score Min can guarantee so far.

2. **Traverse the Tree with Pruning:**
    - As you explore the tree, update alpha and beta values at each node.
    - If, at any point, a node's value becomes worse than the current alpha or beta, stop exploring that path (prune it). This is because further exploration won't influence the final decision.

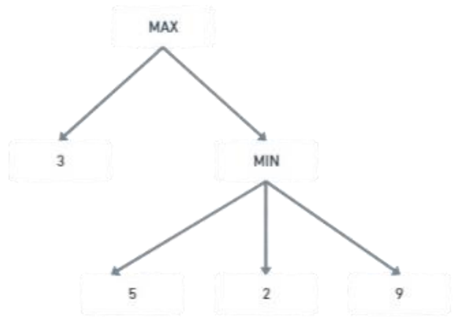
Steps in Alpha-Beta Pruning:

1. **Initialize alpha and beta with -∞ and +∞ respectively.**

2. **Traverse the Game Tree:**
    - At Max nodes, update alpha with the maximum value found so far.
    - At Min nodes, update beta with the minimum value found so far.
3. **Prune When Possible:**
    - If a Max node's score is greater than or equal to beta, stop exploring further (since Min won't allow it).
    - If a Min node's score is less than or equal to alpha, stop exploring further (since Max won't allow it).

Example of Alpha-Beta Pruning:

Suppose a simplified game tree for Max and Min looks like this:



- Max first explores the left child with score 3, updating alpha to 3.
- Next, Max explores the right child. At Min's turn, the first child has a score of 5.
- Since 5 is greater than Max's alpha (3), Min would pick 5 if no better move is found.
- Then, the next child of Min has a score of 2. Now, beta becomes 2 for this branch.
- When Max sees beta 2 on Min's branch, further exploration (the 9 in this case) is pruned because it won't affect the decision.

Alpha-Beta pruning effectively reduces the number of nodes the algorithm needs to evaluate, making it more efficient while still guaranteeing the best move.