AI ONESHOT NOTES

AI UNIT - 1

UNIT 1: Introduction to Artificial Intelligence & Problem Solving

1. **What is AI Technique? [PYQ]**
    - Real-world knowledge properties: Huge volume, next to unimaginable; Not well-organized or well-formatted; Keeps changing constantly.
    - AI Technique: A method to organize & use knowledge efficiently such that: It should be perceivable by the people who provide it; It should be easily modifiable to correct errors; It should be useful in many situations though it is incomplete or inaccurate.
    - Elevates speed of execution for complex programs it is equipped with.
2. **Applications of AI [PYQ]**
    - Gaming: [PYQ] Crucial role in strategic games (chess, poker, tic-tac-toe, etc.); Machine can think of large number of possible positions based on heuristic knowledge.
    - Natural Language Processing (NLP): [PYQ] Possible to interact with the computer that understands natural language spoken by humans.
    - Expert Systems: [PYQ] Integrate machine, software, and special information to impart reasoning and advising; Provide explanation and advice to the users.
    - Vision Systems: Understand, interpret, and comprehend visual input on the computer.
        - Examples: Spying airplane takes photographs for spatial information or maps; Doctors use clinical expert system to diagnose patients; Police use software to recognize criminal faces from forensic portraits.
    - Speech Recognition: Systems capable of hearing and comprehending language (sentences, meanings); Can handle different accents, slang, background noise, etc.
    - Handwriting Recognition: Reads text written on paper/screen by pen/stylus; Recognizes letter shapes and converts to editable text.
    - Intelligent Robots: Able to perform tasks given by a human; Have sensors for physical data (light, heat, temperature, movement, sound, bump, pressure); Efficient processors, multiple sensors, huge memory; Capable of learning from mistakes and adapting to new environments.
    - Astronomy: Solve complex universe problems (how it works, origin, etc.).
    - Healthcare: [PYQ] Better and faster diagnosis than humans; Can inform when patients are worsening for timely medical help.
    - Finance: [PYQ] Automation, chatbot, adaptive intelligence, algorithm trading, machine learning in financial processes.

- Data Security: Make data more safe and secure; Examples (AEG bot, AI2 Platform) determine software bugs and cyber-attacks.
- Social Media: Manage billions of user profiles efficiently; Analyze data for latest trends, hashtags, user requirements.
- Travel & Transport: Travel arrangements, suggesting hotels, flights, best routes; AI-powered chatbots for human-like customer interaction.
- Automotive Industry: [PYQ] Virtual assistants for better performance (e.g., TeslaBot); Developing self-driven cars for safer journeys.
- Robotics: [PYQ] Create intelligent robots that perform tasks with own experiences, not just pre-programmed; E.g., Humanoid robots (Erica, Sophia) that talk and behave like humans.
- Entertainment: AI-based applications (Netflix, Amazon) using ML/AI for recommendations.
- Agriculture: Agriculture robotics, solid and crop monitoring, predictive analysis.
- E-commerce: Competitive edge; helps shoppers discover products with recommended size, color, brand.
- Education: [PYQ] Automate grading, allowing tutors more teaching time; AI chatbot as teaching assistant; Personal virtual tutor accessible anytime, anywhere.

3. **What is Artificial Intelligence (AI)? [PYQ]**
   - A branch of computer science to create intelligent machines which can behave like humans, think like humans, and make decisions. [PYQ]
   - Composed of two words: "Artificial" (man-made) and "Intelligence" (thinking power).
   - Hence AI means "a man-made thinking power."
   - AI exists when a machine can have human-based skills such as learning, reasoning, and solving problems. [PYQ]
   - The goal is for machines to work with their own intelligence, not just programmed algorithms.
4. **Why Artificial Intelligence?**
   - Create software/devices to solve real-world problems easily and accurately (health issues, marketing, traffic issues, etc.).
   - Create personal virtual assistants (e.g., Cortana, Google Assistant, Siri, Alexa).
   - Build robots to work in environments where human survival is at risk.
   - Opens a path for other new technologies, new devices, and new Opportunities.
5. **Goals of Artificial Intelligence**
   - Replicate human intelligence. [PYQ]
   - Solve Knowledge-intensive tasks.
   - An intelligent connection of perception and action.
   - Building a machine which can perform tasks that require human intelligence such as: [PYQ] Proving a theorem; Playing chess; Plan some surgical operation; Driving a car in traffic.

- Creating systems that can exhibit intelligent behavior, learn new things by itself, demonstrate, explain, and advise its user.

6. **What Comprises Artificial Intelligence? / What is Intelligence Composed of? [PYQ]**
    - Intelligence is an intangible part of our brain.
    - It is a combination of: Reasoning (The set of processes that enables us to provide basis for judgment, making decisions, and prediction); Learning (The activity of gaining knowledge or skill by studying, practicing, being taught, or experiencing something); Problem Solving [PYQ] (The process in which one perceives and tries to arrive at a desired solution. Includes decision making - selecting the best alternative); Perception (The process of acquiring, interpreting, selecting, and organizing sensory information. Presumes sensing); Linguistic Intelligence (One's ability to use, comprehend, speak, and write verbal and written language).
    - Disciplines required to achieve AI factors for a machine/software: Mathematics, Biology, Psychology, Sociology, Computer Science, Neurons Study, Statistics.
7. **Advantages of Artificial Intelligence**
    - High Accuracy with less errors: AI machines are prone to fewer errors as decisions are based on pre-experience or information.
    - High-Speed: AI systems can be very high-speed and enable fast decision-making (e.g., beating chess champions).
    - High reliability: AI machines are highly reliable and can perform the same action multiple times with high accuracy.
    - Useful for risky areas: Helpful in situations like defusing bombs, exploring the ocean floor.
    - Digital Assistant: Useful for providing digital assistance (e.g., E-commerce product suggestions).
    - Useful as a public utility: Self-driving cars for safer journeys, facial recognition for security, NLP for communication.
8. **Disadvantages of Artificial Intelligence**
    - High Cost: Hardware and software requirements are very costly; requires maintenance.
    - Can't think out of the box: Robots only do work for which they are trained/programmed.
    - No feelings and emotions: AI machines cannot make emotional attachments; may be harmful if proper care is not taken.
    - Increase dependency on machines: People become more dependent, losing mental capabilities.
    - No Original Creativity: AI machines cannot beat human intelligence in creativity and imagination.
9. **History of Artificial Intelligence [PYQ]**
    - Maturation of Artificial Intelligence (1943-1952): Year 1943: [PYQ] Warren McCulloch & Walter Pitts proposed a model of artificial neurons. Year 1949: Donald Hebb demonstrated Hebbian learning (modifying connection strength between neurons). Year 1950: [PYQ] Alan Turing published "Computing Machinery and Intelligence," proposing the Turing test. [FIG]

- The birth of Artificial Intelligence (1952-1956): [PYQ] Year 1955: [PYQ] Allen Newell & Herbert A. Simon created the "Logic Theorist," the first AI program. Year 1956: [PYQ] John McCarthy coined "Artificial Intelligence" at the Dartmouth Conference. High-level languages (FORTRAN, LISP, COBOL) invented.
- The golden years-Early enthusiasm (1956-1974): Year 1966: [PYQ] Joseph Weizenbaum created ELIZA, the first chatbot. Year 1972: WABOT-1, the first intelligent humanoid robot, built in Japan.
- The first AI winter (1974-1980): Severe shortage of funding, decreased publicity.
- A boom of AI (1980-1987): Year 1980: [PYQ] AI came back with "Expert System." First national AAAI conference at Stanford University. Year 1986: NETtalk (neural network for reading and pronouncing words).
- The second AI winter (1987-1993): Investors stopped funding due to high cost and inefficient results (though XCON was cost-effective).
- The emergence of intelligent agents (1993-2011): Year 1997: [PYQ] IBM Deep Blue beat world chess champion Gary Kasparov. Year 2002: AI entered homes with Roomba (vacuum cleaner). Year 2006: AI in Business world (Facebook, Twitter, Netflix).
- Deep learning, big data and artificial general intelligence (2011-present): [PYQ] Year 2011: [PYQ] IBM's Watson won Jeopardy. Apple's Siri launched. Year 2012: Google launched "Google now." Year 2014: Chatbot "Eugene Goostman" won a Turing test competition variant. Microsoft Cortana. Year 2015: Amazon Echo with Alexa. Year 2018: IBM's "Project Debater." Google's "Duplex" virtual assistant.

10. **Types of Artificial Intelligence [PYQ] [FIG]**
    - Type-1: Based on Capabilities
        - 1. Weak AI or Narrow AI: [PYQ] Able to perform a dedicated task with intelligence. Most common type. Cannot perform beyond its field or limitations; trained for one specific task. Examples: Apple Siri, IBM's Watson, playing chess, e-commerce suggestions, self-driving cars, speech recognition, image recognition.
        - 2. General AI: [PYQ] Could perform any intellectual task with efficiency like a human. System would be smarter and think like a human by its own. Currently, no such system exists; still under research.
        - 3. Super AI: [PYQ] Level of Intelligence where machines could surpass human intelligence. Can perform any task better than humans with cognitive properties. An outcome of general AI. Characteristics: ability to think, reason, solve puzzles, make judgments, plan, learn, communicate by its own. Hypothetical concept. [FIG] (showing progression: Narrow -> General -> Super)
    - Type-2: Based on Functionality [PYQ]

-
  1. Reactive Machines: Most basic types. Do not store memories or past experiences for future actions. Only focus on current scenarios and react based on possible best action. Examples: IBM's Deep Blue, Google's AlphaGo.
-
  2. Limited Memory: [PYQ] Can store past experiences or some data for a short period of time. Can use stored data for a limited time period only. Examples: Self-driving cars (store recent speed of nearby cars, distance of other cars, speed limit, etc.).
-
  3. Theory of Mind: Should understand human emotions, people, beliefs, and be able to interact socially like humans. This type of AI machine is still not developed; researchers are making efforts.
-
  4. Self-Awareness: Future of Artificial Intelligence. Machines will be super intelligent, with their own consciousness, sentiments, and self-awareness. These machines will be smarter than human mind. Does not exist in reality yet; a hypothetical concept.

11. **AI Agents [PYQ]**
    - Definition: [PYQ] [FIG] Anything that perceives its environment through sensors and acts upon that environment through actuators.
    - Agent runs in the cycle of perceiving, thinking, and acting.
    - Types of Agents (Examples): Human-Agent (Eyes, ears (sensors); hands, legs, vocal tract (actuators)); Robotic Agent (Cameras, infrared range finder, NLP (sensors); various motors (actuators)); Software Agent (Keystrokes, file contents (sensory input); display output on screen (actions)).
    - Agent Terminology (PEAS Components): [PYQ] (as in "What is PEAS?") P - Performance Measure (Criteria for success of an agent's behavior); E - Environment (The world in which the agent operates); A - Actuators (Devices agent uses to make changes in the environment e.g., motors, display screen); S - Sensors (Devices agent uses to perceive its environment e.g., camera, keyboard).
    - PEAS Representation: [PYQ] [FIG] A way to describe an AI agent. Example for self-driving cars: Performance (Safety, time, legal drive, comfort); Environment (Roads, other vehicles, road signs, pedestrians); Actuators (Steering, accelerator, brake, signal, horn); Sensors (Camera, GPS, speedometer, odometer, accelerometer, sonar).
    - Rational Agent: [PYQ] An agent which has clear preference, models uncertainty, and acts to maximize its performance measure with all possible actions. Performs the "right thing." Rationality depends on: Performance measure, Agent's prior knowledge, Actions agent can perform, Percept sequence.
    - Intelligent Agent vs. Rational Agent: Intelligent Agent (A system that can perceive its environment and take actions to achieve a specific goal); Rational Agent (An Intelligent Agent

that makes decisions based on logical reasoning and optimizes its behavior to achieve a specific goal).

- Structure of an AI Agent: [FIG] Agent = Architecture + Agent program.
    - Architecture: Machinery that an AI agent executes on.
    - Agent Function: Maps a percept to an action (f: P* → A).
    - Agent program: Implementation of agent function, executes on physical architecture.
- Types of AI Agents (Program Structure): [PYQ] [FIG] for each type
    - Simple Reflex Agent: [FIG] Acts based only on the current percept, ignoring percept history; Works on Condition-action rules; Succeeds only in fully observable environments.
    - Model-Based Reflex Agent: [FIG] Maintains an internal state to track the part of the world it cannot currently see; Uses a model of how the world evolves and how its actions affect the world; Can handle partially observable environments.
    - Goal-Based Agent: [FIG] Acts to achieve explicit goals. Knowledge of goals describes desirable situations; May require search and planning. More flexible.
    - Utility-Based Agent: [FIG] Acts to maximize expected utility (a measure of "happiness" or success at a given state); Useful with multiple conflicting goals or uncertainty.
    - Learning Agent: [PYQ] [FIG] Can learn from past experiences to improve performance; Starts with basic knowledge, adapts automatically through learning; Components: Learning element, Critic, Performance element, Problem generator.

12. **Agent Environment [PYQ]**
    - Everything in the world which surrounds the agent, but not part of the agent itself.
    - Where the agent lives, operates, and gets something to sense and act upon.
    - Properties/Features of Environments: [PYQ] Fully observable vs. Partially Observable (Agent's sensors give access to complete state vs. not); Deterministic vs. Stochastic [PYQ] (Next state completely determined by current state and action vs. randomness involved); Episodic vs. Sequential [PYQ] (Agent's experience is series of one-shot actions vs. agent requires memory of past actions); Static vs. Dynamic [PYQ] (Environment cannot change while agent is deliberating vs. it can change); Discrete vs. Continuous (Finite number of percepts/actions vs. continuous values); Single-agent vs. Multi-agent [PYQ] (Only one agent involved vs. multiple agents operating - cooperative/competitive); Known vs. Unknown (Outcomes for all actions known vs. agent needs to learn how environment works); Accessible vs. Inaccessible (Agent can obtain complete/accurate state info vs. not - subtle, related to observability).
13. **Turing Test [PYQ] [FIG]**
    - Proposed by Alan Turing (1950) to check if a machine can think like a human.
    - Involves three players: Computer, Human responder, Human Interrogator.
    - Interrogator, isolated, tries to identify the machine based on text conversation.
    - If the machine's responses are indistinguishable from a human's, it passes.

- Features required for a machine to pass the Turing test: [PYQ] Natural language processing (NLP) (To communicate); Knowledge representation (To store information); Automated reasoning (To use stored information for answers); Machine learning (To adapt and detect generalized patterns); Vision (For total Turing test) (To recognize interrogator actions/objects); Motor Control (For total Turing test) (To act upon objects if requested).
- Chatbots that attempted the test: ELIZA, Parry, Eugene Goostman.

14. **Problem Solving Agents & State Space Search [PYQ]**
    - Problem-solving agents: Goal-based agents that use algorithms to search for a sequence of actions leading to a goal.
    - Search Problem Components: [PYQ] Initial State [PYQ] (The state from where the search begins); Actions [PYQ] (Set of possible operations applicable to a state); Transition Model [PYQ] (Describes the result of performing an action in a state); Goal Test [PYQ] (Determines if a given state is a goal state); Path Costing [PYQ] (Assigns a numerical cost to a path).
    - State Space: [PYQ] [FIG] Set of all possible states reachable from the initial state by any sequence of actions.
    - State Space Search: [FIG] (general search process diagram) Process of finding a path from an initial state to a goal state.
    - Types of Problems in AI (based on solution step recoverability): Ignorable (Solution steps can be ignored e.g., theorem proving); Recoverable (Solution steps can be undone e.g., 8-puzzle); Irrecoverable (Solution steps cannot be undone e.g., chess).
    - Problem Formulation: Defining states, initial state, actions, transition model, goal test, path cost.
        - Example: Vacuum World [FIG] States (Agent location and dirt location e.g., 2 rooms, 2^2 dirt configs = 8 states); Initial State (Any state can be assigned); Actions (Move Left, Move Right, Suck); Goal Test (All squares are clean); Path Cost (Each step costs 1).
    - Properties of Search Algorithms: [PYQ] Completeness (Does algorithm guarantee finding solution if one exists?); Optimality [PYQ] (e.g. A*) (Does it find the best (least-cost) solution?); Time Complexity (How long does it take as function of problem size?); Space Complexity (How much memory does it require?).
15. **Uninformed Search (Blind Search) Algorithms [PYQ]**
    - No domain-specific knowledge beyond problem definition. Explores in a brute-force way.
    - Types:
        -
            1. Breadth-First Search (BFS): [PYQ] [FIG] Explores level by level (shallowest nodes first). Uses FIFO queue. Completeness: Yes. Optimality: Yes (if path cost is non-decreasing function of depth, e.g., all steps cost 1). Time Complexity: O(b^d) (b=branching factor, d=depth of shallowest goal). Space Complexity: O(b^d) (stores all nodes at current depth frontier).
        -

2. Depth-First Search (DFS): [PYQ] [FIG] Explores one branch of the tree as far as possible before backtracking. Uses LIFO stack (often implemented via recursion). Completeness: No (can get stuck in infinite loops on infinite graphs); Yes for finite graphs or if loop detection is used. Optimality: No. Time Complexity: O(b^m) (m=maximum depth of state space). Space Complexity: O(b*m) (stores only nodes on current path).

-

3. Depth-Limited Search (DLS): [PYQ] [FIG] DFS with a pre-defined depth limit (l). Solves infinite loop problem of DFS in infinite state spaces. Completeness: Yes, if l >= d. No, if l < d (goal deeper than limit). Optimality: No. Time Complexity: O(b^l). Space Complexity: O(b*l).

-

4. Iterative Deepening Depth-First Search (IDDFS): [PYQ] [FIG] Combines benefits of BFS (completeness, optimality for unit costs) and DFS (space efficiency). Performs DLS repeatedly with increasing depth limits (0, 1, 2, ..., d). Completeness: Yes. Optimality: Yes (if step costs are uniform). Time Complexity: O(b^d). Space Complexity: O(b*d).

-

5. Uniform Cost Search (UCS): [PYQ] [FIG] Expands the node with the lowest path cost (g(n)) from the start. Uses a priority queue. Finds the cheapest path in graphs where edge costs can vary and are non-negative. Completeness: Yes (if step costs > 0 or a very small positive ε). Optimality: Yes. Time Complexity: O(b^(1 + C*/ε)) (C*=cost of optimal solution, ε=minimum step cost). Space Complexity: O(b^(1 + C*/ε)).

-

6. Bidirectional Search: [FIG] Runs two simultaneous searches: one forward from initial state, one backward from goal state. Stops when the two searches meet in the middle. Reduces search complexity (roughly 2 * b^(d/2) instead of b^d). Requires an explicit goal state and ability to reverse actions.

16. **Informed (Heuristic) Search Algorithms [PYQ]**
    - Use problem-specific knowledge (heuristics) to guide the search more efficiently.
    - Heuristic function (h(n)): [PYQ] An estimate of the cost from the current state 'n' to the nearest goal state. Value is always positive or zero (h(goal)=0).
        - Admissibility: [PYQ] h(n) <= h*(n) (where h*(n) is the true cost to the nearest goal). An admissible heuristic never overestimates the cost.
        - Consistency (Monotonicity): For every node n and any successor n' generated by an action a, h(n) <= cost(n,a,n') + h(n'). This is a stronger condition than admissibility. If h is consistent, f(n) is non-decreasing along any path.
    - Pure Heuristic Search: Expands the node with the lowest h(n) value. It's essentially Greedy Best-First Search. Not optimal or complete.
    - Types:
        -

1. Greedy Best-First Search [Best-first Search]: [PYQ] [FIG] Expands the node that appears to be closest to the goal, based solely on the heuristic h(n). Uses a priority queue, ordering nodes by h(n). Evaluation function f(n) = h(n). Completeness: No (can get stuck in loops, similar to DFS). Optimality: No. Time Complexity: O(b^m) in the worst case. Space Complexity: O(b^m) in the worst case (keeps all nodes in memory).

-

2. A* Search Algorithm: [PYQ] [FIG] Combines Uniform Cost Search (g(n) - actual cost from start to node n) and Greedy Best-First Search (h(n) - estimated cost from n to goal). Evaluation function: f(n) = g(n) + h(n). [PYQ] Expands the node with the lowest f(n) value from the OPEN list (priority queue). Completeness: Yes. Optimality: [PYQ] Yes, if h(n) is admissible (for tree search) or consistent (for graph search, more common). Time Complexity: Exponential in the worst case, but can be much better with a good heuristic. Space Complexity: Exponential (keeps all generated nodes in memory, its main drawback).

-

3. AO* Algorithm (for AND-OR graphs): [PYQ] [FIG] Used for problems that can be decomposed into subproblems. AND nodes: All subproblems must be solved. OR nodes: Any one of the alternative subproblems can be solved. A best-first search algorithm applied to AND-OR graphs. Divides a problem into smaller subproblems and uses heuristics. Evaluation f(n) = g(n) + h(n) (cost definitions complex for AND-OR). More effective in searching AND-OR trees than A* (A* is for state-space graphs). Doesn't guarantee optimality like A* for standard state spaces, can use less memory, designed to avoid endless loops in cyclic AND-OR graphs. Example: "Want to buy a car" OR node: [Steal car], [Buy with own money]; If [Buy with own money] chosen: AND node: [Get money], [Find a car to buy].

17. **Hill Climbing Algorithm [PYQ] [FIG]**
   - A local search algorithm that continuously moves in the direction of increasing value (uphill) to find the peak of the mountain or the best solution to the problem.
   - Terminates when it reaches a peak value where no neighbor has a higher value. It's a greedy approach; it only looks at its immediate neighbors. No backtracking. Keeps only a single current state in memory.
   - State-space Diagram for Hill Climbing: [FIG] A graphical representation showing states vs. Objective function/Cost.
     - Local Maximum: A state better than all its neighbors but not the best overall. Global Maximum: The best possible state in the landscape. Flat local maximum (Plateau): A flat region where all neighbors have the same value as the current state. Shoulder: A plateau region that has an uphill edge. Ridge: A path where movement along it might be uphill, but any move off the ridge is downhill.
   - Types of Hill Climbing: Simple Hill Climbing (Evaluates neighbor nodes one by one and selects the first one that is better than the current state); Steepest-Ascent Hill Climbing [PYQ] (Examines all neighboring nodes of the current state and selects the one that is closest to the

goal state i.e., the best successor. Consumes more time); Stochastic Hill Climbing (Does not examine all neighbors. Randomly selects one neighbor and decides whether to choose it or examine another).

  - Problems in Hill Climbing: [PYQ] Local Maximum [PYQ] (Algorithm gets stuck on a peak that isn't the global maximum. Solution: Backtracking, random restart hill climbing); Plateau [PYQ] (All neighbors have the same value, making it hard to choose a direction. Solution: Make a big step/random jump); Ridge [PYQ] (A narrow path of higher elevation. Steps off the ridge are downhill. Solution: Use bidirectional search or allow multiple moves before re-evaluating).
  - Simulated Annealing: [PYQ] A variation that attempts to overcome local maxima. Allows occasional "bad" moves (to a state with lower value) with a probability that decreases over time (controlled by a "temperature" parameter). Aims for both efficiency and completeness (finding the global optimum if cooled slowly enough).

18. **Production Systems (Rule-Based Systems) [PYQ] [FIG]**
    - Based on a set of rules about behavior, typically in IF-THEN format.
    - Used in expert systems, automated planning, and action selection.
    - Components: [PYQ] [FIG]
      - Global Database (Working Memory): Central data structure containing facts about the current state of the world.
      - Set of Production Rules: Rules that operate on the global database. Each rule has a precondition (IF part) and an action (THEN part). If the precondition is satisfied by the database, the rule can be applied, changing the database.
      - Control System (Inference Engine): Chooses which applicable rule to "fire" if multiple rules match (conflict resolution); Determines when to stop computation (e.g., when a goal is reached or no more rules apply).
    - Types of Inference Rules: [FIG] Deductive Inference Rules (Logic used to draw specific conclusions from general premises or facts. e.g., Modus Ponens); Abductive Inference Rules (Used to make educated guesses or hypotheses based on observed data; generating plausible explanations).
    - Features of Production Systems: Simplicity (Uniform IF-THEN structure for knowledge representation); Modularity (Knowledge is coded in discrete pieces (rules), easy to add/delete); Modifiability (Facility for modifying rules); Knowledge-intensive (Knowledge base stores pure knowledge, separate from control).
    - Classes of Production Systems: [PYQ] Monotonic Production System (Application of a rule never prevents the later application of another rule that could also have been applied. Once a fact is true, it remains true); Non-Monotonic Production System [PYQ] (Application of a rule may prevent later application of another. Facts can be retracted. More dynamic and adaptive); Partially Commutative Production System (Rules can be applied with some flexibility in order, but the order might still matter for efficiency or intermediate states); Commutative Production System (The order of rule application does not affect the final outcome).

- Applications: [PYQ] Customer support chatbots, fraud detection systems, medical diagnosis systems, traffic management systems.
- Water Jug Problem as a Production System Example: [PYQ] [FIG] Goal (Obtain a specific amount of water e.g., 4 liters in a designated jug, given jugs of fixed capacities e.g., 5l and 3l and a water source); Rules (Operators) (Fill jug X, Empty jug X, Pour water from jug X to jug Y until Y is full or X is empty); Global Database (Current amount of water in each jug e.g., [x, y]).

19. **Means-Ends Analysis (MEA) [PYQ] [FIG]**
    - A problem-solving technique that combines forward and backward reasoning.
    - Focuses on reducing the "difference" between the current state and the goal state.
    - Steps: 1. Compare CURRENT state to GOAL state. If no differences, return Success. 2. Else, select the most significant difference. 3. Select an operator O relevant to reducing this difference. If no such operator, signal failure. 4. Attempt to apply operator O:
        - Operator Subgoaling: If O's preconditions are not met in CURRENT, set a sub-goal to reach a state (O-START) where they are met. Recursively call MEA(CURRENT, O-START).
        - If successful, apply O to get O-RESULT.
        - Recursively call MEA(O-RESULT, GOAL) to solve the remaining problem.
        - If both recursive calls are successful, return the combined plan.
    - Example: Planning a trip. Difference: Current location vs. Destination. Operator: Fly. Precondition: Have ticket. Sub-goal: Buy ticket.
20. **Constraint Satisfaction Problems (CSP) [PYQ]**
    - Defined by: A set of Variables (X1, ..., Xn); For each variable Xi, a Domain Di of possible values; A set of Constraints C that specify allowable combinations of values for subsets of variables.
    - Goal: To find an assignment of a value to each variable from its domain such that all constraints are satisfied.
    - Examples: [PYQ] Map Coloring (Assign colors to regions so no adjacent regions have same color); N-Queens Problem (Place N queens on an N×N chessboard so no two queens attack each other); Sudoku; Cryptarithmetic Puzzles (e.g., SEND + MORE = MONEY) [PYQ].
    - Solution Methods: Typically involve search (e.g., backtracking search) often enhanced by: Heuristics (e.g., most constrained variable, least constraining value); Inference/Constraint Propagation (e.g., forward checking, arc consistency) to prune the search space.

AI UNIT - 2

UNIT 2: Knowledge Representation and Reasoning

1. **Knowledge Representation (KR) in AI [PYQ]** (From Unit 4 document, Part 1 related content)
    - "Fuzzy" Meaning: Things that are not clear, vague, or imprecise. (This is specific to Fuzzy Logic introduction, not general KR definition).

- Purpose: Handles situations where decisions/statements are not strictly true or false. (Again, Fuzzy Logic specific).
- Provides many values between true (1) and false (0), offering flexibility. (Fuzzy Logic).
- Allows representation of human-like linguistic uncertainty. [FIG] (Boolean Yes/1, No/0 vs. Fuzzy Very much/0.9, Little/0.25, Very less/0.1) (Fuzzy Logic).
- Introduced by: Lotfi Zadeh in 1965 based on Fuzzy Set Theory. (Fuzzy Logic).

2. **The Synergy of Knowledge and Intelligence** (General AI concept, not explicitly detailed under a "Unit 2" heading in provided OCR for "Unit 4 notes")
3. **Core Methods of Knowledge Representation [PYQ]** (From Unit 4 document, Part 1 related content)
   - Fuzzy Set Theory: Formalized by Lotfi Zadeh (1965) to generalize classical set theory.
   - Crisp Set: [FIG] Elements have either full (1) or no (0) membership. Defined by a characteristic function $\chi A: X \rightarrow \{0, 1\}$.
   - Fuzzy Set: [PYQ] [FIG] Elements have a degree of membership between 0 and 1. Defined by a membership function (MF) $\mu A: X \rightarrow [0, 1]$. [PYQ] ($\mu A(x)=1$: fully in, $\mu A(x)=0$: not in, $0<\mu A(x)<1$: partly in). Formal definition: $A = \{(x, \mu A(x)) \mid x \in X\}$. Totally characterized by its MF.
4. **The AI Knowledge Cycle [PYQ] [FIG]** (General AI Learning concept, not explicitly detailed under a "Unit 2" heading in provided OCR for "Unit 4 notes")
5. **Types of Knowledge in AI [PYQ] [FIG]** (General AI concept)
6. **What to Represent in AI Systems:** (General AI concept)
7. **Approaches to Knowledge Representation (Categorization)** (General AI concept)
8. **Techniques of Knowledge Representation (Implementation Methods) [PYQ] [FIG]** (General AI concept, Fuzzy logic is one technique)
9. **Propositional Logic (PL) / Statement Logic [PYQ]** (This is a core Unit 2 topic, but Fuzzy Logic is the focus of Part 1 in Unit 4 notes)
10. **Predicate Logic / First-Order Logic (FOL) [PYQ]** (Core Unit 2 topic)
11. **Rules of Inference in Logic [PYQ] [FIG]** (Core Unit 2 topic)
12. **Unification and Resolution in FOL [PYQ]** (Core Unit 2 topic)
13. **Frames in AI: Knowledge Representation and Inheritance [PYQ] [FIG]** (Core Unit 2 topic)
14. **Forward and Backward Chaining [PYQ] [FIG]** (Core Unit 2 topic)
15. **Reasoning in Artificial Intelligence [PYQ]** (Core Unit 2 topic)
16. **Probabilistic Reasoning & Uncertainty [PYQ]** (Core Unit 2 topic)
    - Bayesian Networks: [PYQ] [FIG] Type of Probabilistic Graphical Model (PGM). Represents random variables and their conditional dependencies using a Directed Acyclic Graph (DAG). Uses Bayesian inference for probability computations. Nodes: Random variables. Edges (Arcs): Represent conditional dependencies. Also called: Bayes network, belief network, decision network, Bayesian model. (From Unit 4 document, Part 3)

- Conditional Probability Table (CPT): [PYQ] [FIG] Each node has a CPT specifying P(Node | Parents(Node)). For root nodes (no parents), CPT is just P(Node). (From Unit 4 document, Part 3)

AI UNIT - 3

UNIT 3: Learning, Game Playing, and NLP

Part 1: Learning in Artificial Intelligence [PYQ]

1. **What is Learning?**
   - Core Definition: Learning denotes changes in the system that enable it to do the same task more efficiently next time.
   - Alternative Definitions: Constructing or modifying representations of what is being experienced; Making useful changes in minds (or system's knowledge/parameters).
   - Goals/Benefits: Improves understanding and efficiency; Enables discovery of new, previously unknown things/structures (e.g., data mining); Allows filling in incomplete observations/specifications about a domain (expands expertise, lessens brittleness); Facilitates building adaptive software agents; Reproduces an important aspect of intelligent behavior.
2. **What Characterizes a Learning System?**
   - Iterative Process: 1. Produce a result. 2. Evaluate against an expected result (if available). 3. Tweak the system based on evaluation.
   - Discovery: Can discover patterns without prior expected results (unsupervised learning).
   - Transparency Types: Open Box (System changes e.g., in KB are clearly visible and interpretable by humans); Black Box (System changes are not readily visible or understandable).
3. **What is the Architecture of a Learning Agent/System? [PYQ] [FIG]**
   - (See Unit 1, Learning Agent for diagram: Performance Element, Critic, Learning Element, Problem Generator, interacting with Environment via Sensors/Actuators, and a Knowledge Base/Generalizer).
   - Main Components: Knowledge Base (KB) (Stores what is being learned, domain representation, problem space); Performer (Does something with the KB to produce results); Critic (Evaluates results against expected results/performance standards); Learner (Takes feedback from critic; modifies KB or performer).
   - Optional Component: Problem Generator (May generate test cases to evaluate performance).
4. **Examples of Learning Systems [FIG] (Table format)**
   - Animal Guessing Game: Binary decision tree (Representation); Walk tree, ask questions (Performer); Human feedback (Critic); Elicit & add question (Learner).
   - Playing Chess: Board layout, rules, moves (Representation); Chain rules, identify move (Performer); Who won (credit assignment) (Critic); Adjust rule weights (Learner).

    - Categorizing Documents: Vector of word frequencies (Representation); Apply functions to categorize (Performer); Human-categorized documents (Critic); Modify function weights (Learner).
    - Fixing Computers: Frequency matrix of causes/symptoms (Representation); Use symptoms to ID causes (Performer); Human input on symptoms/cause (Critic); Update frequency matrix (Learner).
    - Identifying Digits (OCR): Probability of digits, pixel matrix (Representation); Input features, output probability (Performer); Human-categorized training set (Critic); Modify association weights (Learner).
5. **Different Learning Paradigms [PYQ]**
    - Rote learning: Direct entry of rules/facts; Memorization.
    - Learning by taking advice: [PYQ] Human/system interaction; Advice operationalization.
    - Learning in problem solving: Parameter adjustments, learning macro-operators, chunking.
    - Learning from examples (Induction): [PYQ] Using specific examples to reach general conclusions.
    - Explanation-based learning (EBL): [PYQ] Learn from one example, then generalize; knowledge-intensive.
    - Learning through discovery: Unsupervised; specific goal not given; finding patterns.
    - Learning through analogy: [PYQ] Determining correspondence between different representations; case-based reasoning.
    - Formal learning theory: Mathematical model of learning (e.g., PAC learning).
    - Neural net learning & genetic learning: [PYQ] Evolutionary search, biologically inspired network learning.
6. **How does Rote Learning work?**
    - Definition: Basic learning activity; memorization.
    - Mechanism: Knowledge copied into KB without modification. Direct entry of rules/facts.
    - Application: Developing ontologies; Data caching for performance.
    - Benefit: Saves re-computation time by storing computed values.
    - Key Capabilities: Organized storage (For fast retrieval); Generalization (To keep stored objects manageable for large state spaces).
7. **How does Learning by Taking Advice work? [PYQ]**
    - Simplicity: Easiest way of learning.
    - Process: 1. Expert/programmer writes instructions/advice. 2. System integrates/learns advice. 3. System can do new things based on advice.
    - Inference Requirement: More inference than rote learning.
    - Operationalization: [PYQ] Stored KB knowledge transformed into an operational form. Program turns advice into usable expressions (concepts, actions). Critical ability.

  - Consideration: Reliability of knowledge source.

8. **How does Learning Occur in Problem Solving?**
   - Context: Program learns by generalizing from its own experiences.
   - Types Discussed:
     - a. Learning by Parameter Adjustment: [PYQ] Scenario (System uses an evaluation procedure combining features into a score e.g., polynomial: Score = ∑ ci * ti). Challenge (Knowing a priori weights (ci) for features (ti)). Process (Start with estimates, modify weights based on experience. Increase weights for good predictors, decrease for poor). Key Questions (When/how much to change coefficients? Credit Assignment Problem [PYQ] - assigning responsibility for outcome). Method (Hill-climbing search).
     - b. Learning with Macro-Operators (MACROPs): [PYQ] Definition (Sequence of actions (operators) treated as a whole). Process (Solve problem, store computed plan (sequence of actions) as a single MACROP with preconditions (initial conditions) and postconditions (goal achieved)). Benefit (Efficiently uses past experience. Critical for non-serializable subgoals. Allows domain-specific learning). Generalization (Replacing constants with variables allows MACROPs for similar problems). Example (STRIPS planning).
     - c. Learning by Chunking: [PYQ] Similarity (Similar to macro-operators. Used in production systems). Chunking Process (When a useful sequence of rule firings occurs (solves sub-problem/impasse), store it as a single new rule ("chunk")). Benefit (Captures search control knowledge. Reduces problem-solving steps). Example (SOAR architecture learns via chunking when impasses are resolved).
     - d. The Utility Problem in Learning: [PYQ] Definition (Knowledge learned to improve performance degrades it instead). Context (Common in speedup learning systems (learning control rules). Systems slow down if they learn too much unrestrainedly). Paradox (Individual rules may be positive, but collectively negative). Solutions: Hardware (Parallel memory systems ("active memories")); Utility Measurement (e.g., PRODIGY - Maintain utility for each rule: savings, frequency, match cost. Discard negative/low utility rules).

9. **How does Learning by Analogy work? [PYQ]**
   - Definition: Acquiring new knowledge about an input entity by transferring it from a known similar entity.
   - Central Intuition: If two entities are similar in some respects, they could be similar in others.
   - Types:
     - Transformational Analogy: [FIG] (Slide 23) Find similar solution. Copy it, make suitable substitutions. Focuses on final solution, not derivation steps.
     - Derivational Analogy: [FIG] (Slides 24-25) Finds problems sharing aspects based on similarity metric. Retrieves derivation (steps) of previous solution. Perturbs old derivation incrementally for new problem. Considers how problem was solved.

10. **What is Explanation-Based Learning (EBL)? [PYQ]**

   - Definition: Learning from a single example by: 1. Explaining (Using existing domain knowledge (theory) to explain why the training example is an instance of the target concept). 2. Generalizing (Turning the explanation into a more general rule/concept definition).
   - Approach: Analytical, knowledge-intensive (requires good domain theory).
   - (Slides 26-31 likely show EBL process: explanation proof tree, generalization, new rule formation). [FIG]

11. **How does Learning by Discovery work? (Unsupervised)**
   - Definition: Acquiring knowledge without a teacher.
   - Types Discussed:
     - a. Theory-Driven Discovery (e.g., AM program - 1976): Goal (Discovers concepts in elementary math/set theory). How it Works (Uses frame-based representation, heuristic search (~250 heuristics), Hypothesis & Test, agenda control). Discoveries (Integers, Addition, Multiplication, Prime Numbers, Goldbach's Conjecture).
     - b. Data-Driven Discovery (e.g., BACON program - 1981): Context (Makes sense of empirical data). How it Works (Starts with variables, inputs experimental data, holds some constant, notices trends, infers mathematical laws). Discoveries (Ideal gas law, Kepler's 3rd law, Ohm's law).
     - c. Clustering: [PYQ] Definition (Grouping data into new classes/clusters. Similar objects in same cluster, dissimilar in others). Goal (Construct meaningful partitioning; maximize intra-class similarity, minimize inter-class). Process (Given feature vectors, find partition into 'c' subsets. Discover labels automatically). AutoClass (Example) (Bayesian approach for optimal classes. Class membership is probabilistic).

12. **What is Formal Learning Theory?**
   - Focus: Provides a formal mathematical model of learning; analyzes learnability.
   - Example: Theory of the Learnable (Valiant - leading to PAC Learning): [PYQ]
     - Probably Approximately Correct (PAC) Learning: A device learns a concept if, given positive/negative examples, it produces a hypothesis h that classifies future examples correctly with probability 1-ε (error tolerance).
     - Complexity Factors: Error tolerance (ε), number of features (t), complexity/size of hypothesis (f).
     - Trainability: Concept class is efficiently learnable if training examples needed is polynomial in ($1/\varepsilon$, t, f).
   - Goal: Quantify knowledge use in learning mathematically.

13. **Neural Net Learning and Genetic Learning [PYQ]**
   - A. Neural Net Learning (Artificial Neural Networks - ANNs): [PYQ] [FIG] Biologically inspired by brain structure. Networks of interconnected processing units (artificial neurons). Learning by adjusting connection strengths (weights) based on training data and error (e.g., backpropagation). Architectures: MLPs, CNNs, RNNs, Transformers.

- B. Genetic Learning (Genetic Algorithms - GAs): [PYQ] Inspired by biological evolution ("survival of the fittest"). Evolutionary search technique. Maintains a population of candidate solutions. Uses operators: fitness evaluation, selection, crossover (recombination), mutation to evolve better solutions.

Part 2: Game Playing in AI [PYQ]
14. **What is the Minimax Algorithm? [PYQ] [FIG]**
* Type: Backtracking algorithm for decision making in game theory/AI.
* Application: Two-player, zero-sum, perfect information games (Tic-Tac-Toe, Chess).
* Goal: Find optimal move for MAX player, assuming MIN opponent also plays optimally.
* Players: Maximizer (MAX) (Tries to get highest score - max benefit); Minimizer (MIN) (Tries to get lowest score - min benefit for MAX).
* Evaluation: Each game state (node) assigned an evaluation score. Positive favors MAX, negative favors MIN.
15. **How does the Minimax Algorithm Work? [PYQ] [FIG]**
* 1. Generate Game Tree: Possible moves and resulting states to a certain depth or terminal states.
* 2. Apply Utility Function: Assign scores (utility values) to terminal (leaf) nodes.
* 3. Backtrack and Propagate Values: (DFS traversal) At Terminal Nodes (Use utility value); At MAX Nodes (Choose MAXIMUM value from children); At MIN Nodes (Choose MINIMUM value from children); Continue to root. Root value is best MAX score; move leading to it is optimal.
* (Example workflow from notes p.10, visualized in p.26-28, and handwritten notes p.35-38).
16. **Properties of Minimax [PYQ]**
* Completeness: Yes, if game tree is finite.
* Optimality: Yes, if both players play optimally.
* Time Complexity: O(b^m) (b=branching factor, m=max depth). Exponential.
* Space Complexity: O(b*m) (for DFS, stores path).
17. **Limitation of the Minimax Algorithm**
* Slow Performance: For complex games (Chess, Go) with large b and m, due to exponential time. Explores entire tree to depth 'm'.
18. **What is Alpha-Beta Pruning? [PYQ] [FIG]**
* Definition: Modified Minimax; an optimization technique.
* Goal: Reduce nodes examined by Minimax, returning same optimal move. Prunes branches that can't influence final decision.
* Mechanism: Uses two threshold parameters: Alpha (α) (Best highest-value choice found so far for MAX. Initial: -∞); Beta (β) (Best lowest-value choice found so far for MIN. Initial: +∞).
* Pruning Condition: Pruning occurs when α ≥ β. [PYQ]
* Scope: Applicable at any depth; can prune single leaves or entire sub-trees.
19. **Key Points and Working of Alpha-Beta Pruning [PYQ] [FIG]**
* Key Points: MAX player only updates α; MIN player only updates β; Node's computed value (min/max of children) passed upwards, not α/β values; α and β values passed down to child nodes during exploration.
* (Working Example from notes p.12, visualized in p.30-34). 1. Start at Root (A): α = -∞, β = +∞. 2. Explore D (MAX children): α updated (e.g., to 3). D returns 3. 3. At B (MIN parent of D): β updated to min(+∞, 3) = 3. Now B has α=-∞, β=3. 4. Explore E (MAX sibling of D, under B): α passed as -∞, β as 3.

5. At E: if first child's value makes E's α (say 5) ≥ B's β (3) -> PRUNE remaining children of E. 6. And so on...
20. **How does Move Ordering Affect Alpha-Beta Pruning? [PYQ]**
* High Dependence: Pruning effectiveness highly depends on node examination order.
* Worst Ordering: Best moves examined last. Minimal/no pruning. Time ≈ Minimax O(b^m).
* Ideal Ordering: Best move always examined first. Maximum pruning. Time ≈ O(b^(m/2)).
* Rules for Good Ordering: Explore best move from shallowest node first; Use heuristics to order nodes (potentially best ones first); Domain knowledge (e.g., Chess: captures > threats > forward moves); Iterative deepening, transposition tables.

Part 3: Natural Language Processing (NLP) [PYQ]
21. **What is Natural Language Processing (NLP)? [PYQ]**
* Definition: Branch of CS/AI focused on enabling computers to communicate with people using everyday, natural language.
* Related Field: Computational Linguistics.
* Two Goals: Science Goal (Understand how language operates); Engineering Goal (Build systems to analyze/generate language; reduce man-machine gap).
* Two Views: Classical (Symbolic, Rule-based) vs. Statistical/ML (Data-driven).
22. **Core Areas/Levels of NLP (NLP Trinity/Stages) [PYQ] [FIG]**
* (Diagram likely shows increasing complexity from Morphology to Discourse).
* Morphological Analysis: Word structure (morphemes).
* Part-of-Speech (POS) Tagging: [PYQ] Assigning word categories (noun, verb).
* Chunking/Shallow Parsing: Identifying basic phrases (Noun Phrases, Verb Phrases).
* Parsing/Syntactic Analysis: [PYQ] Determining full sentence structure (grammar).
* Semantics: [PYQ] Extracting meaning (word sense, sentence meaning, semantic roles).
* Discourse & Coreference: Analyzing relationships between sentences, resolving pronouns.
* Algorithms/Models: HMM, MEMM, CRF, RNN.
23. **Why is NLP/Language Technology Relevant in the Indian Context?**
* Linguistic Diversity: Highly multilingual (22 official, 122 major languages).
* Digital Divide: Only ~20% understand English; 80% potentially excluded if content not in local languages.
* Internet/Social Media Growth: Necessitates NLP for processing Indian language content.
* Goal: Effective communication, inclusive digital society, citizen flexibility.
24. **Indian Government Initiatives in NLP (e.g., TDIL)**
* TDIL Programme (Technology Development for Indian Languages - MeiTY): Objective (Develop tools/techniques for human-machine interaction without language barriers; create/access multilingual resources).
* Major Initiatives: Machine Translation (MT) (Anuvadaksh (English to Indian Langs), Angla-Bharti (English to Indian Langs), Sampark (Indian Lang to Indian Lang)); Cross-Lingual Information Access (CLIA) (For major Indian languages); OCR (Robust Document Analysis & Recognition) (For 14 languages); Text-to-Speech (TTS) (In Indian Languages); Automatic Speech Recognition (ASR) (In Indian Languages); Domain-Specific MT (E.g., Hindi to English (Judicial Domain)).
25. **NLP in Governance (Case Study: MyGov.in Portal)**

* MyGov.in Portal: Citizen-centric platform for participatory governance.
* NLP Challenge: Manually mining relevant info from huge user post volume is infeasible.
* Code-Mixing Issue: Users mix languages (e.g., Hindi + English).
* Need for NLP/ML: To analyze feedback, understand public opinion (Sentiment Analysis), handle code-mixed data.
26. **Why Analyse Public Opinion using NLP?**
* Importance: Public opinions aid betterment of human lives.
* Opportunity: User-generated content offers new ways to understand social behavior.
* Benefit for Governance: Helps anticipate social changes, adapt to population needs.
* Relevant Field: Opinion Mining / Sentiment Analysis. [PYQ]
27. **Projected Growth and Evolution of NLP**
* Growth: Exponential; e.g., $16 billion market by 2021 (~16% CAGR).
* Reasons for Growth: Rise of Chatbots, need for customer insights, automation (messaging, translation, speech tasks).
* Major Players: Amazon, Google, Microsoft, Facebook, IBM.
* Evolution: Human-computer interaction → human-computer conversation.
* Critical Advancements: Biometrics, Humanoid Robotics.
28. **How can NLP be used specifically in Governance?**
* Goal: Improve service delivery, decrease citizen-government interaction gap.
* Uses on Government Websites: Making e-governance info available in multiple languages (MT); Natural Language Generation (NLG) for reports, summaries; Chatbots for information access/service requests (multilingual support).
29. **NLP in Business and Healthcare**
* Business: Sentiment Analysis (Public/customer opinion on products/services); Email Filters (Spam filtering, categorization); Voice Recognition (Smart voice-driven interfaces/services); Information Extraction (Key data from documents).
* Healthcare: Improved EHR Experience (Analyzing EHRs, clinical notes. Voice-support, predictive analytics); Reduced Communication Gap (Patient interaction in own language (portals)); Improved Quality of Care (Calculating inpatient care measures, monitoring guidelines); Patient Identification (Identifying patients needing improved care coordination).
30. **NLP in Finance and Other Domains**
* Finance: Credit Scoring (ML/NLP to assess creditworthiness); Document Search/Analysis (Automating document processing); Fraud Detection (Analyzing transactions/communications for fraud); Stock Market Prediction (Sentiment analysis on news/social media).
* Other Domains: National Security (Sentiment analysis in cross-border languages, hate speech/radicalization detection); Recruitment (Searching/screening applications/resumes).
31. **Perspectives and Allied Disciplines of NLP [FIG]**
* AI Inter-dependencies: NLP interacts with Search, Logic, ML, Vision, KR, Planning, Robotics, Expert Systems.
* Allied Disciplines: Philosophy, Linguistics, Probability & Statistics, Cognitive Science, Psychology, Brain Science, Physics (Info Theory), CS & Engg.
32. **What is the Turing Test? (NLP Context) [PYQ] [FIG]**
* (Reiteration from Unit 1) Test of machine's ability to exhibit human-equivalent intelligent behavior via

natural language conversation.
33. **Natural Languages vs. Computer Languages**
* Ambiguity: Primary difference. Natural languages are inherently ambiguous at multiple levels.
* Formal Languages (Programming): Designed to be unambiguous. Defined by grammar for unique parse. Efficient, deterministic parsing.
34. **Stages of NLP Processing and Ambiguity Examples [PYQ] [FIG]**
* (NLP architecture involves stages, each with ambiguity).
* Phonetics & Phonology (Speech Processing): Challenges (Homophones (bank/bank), word boundary segmentation (I got [ua]plate), disfluencies).
* Morphological Analysis (Word Structure): [PYQ] Definition (Study of internal word structure. Morpheme = smallest meaningful unit). Task (Segmenting words (carried → carry + ed)). Challenge (Ambiguity (unlockable → un+lockable OR unlock+able?)).
* Lexical Analysis (Dictionary Lookup): [PYQ] Task (Access dictionary, get word properties (POS, semantic features)). Challenge (Lexical Disambiguation (POS: dog noun/verb; WSD: dog animal/person). Neologisms).
* Syntactic Analysis (Sentence Structure / Parsing): [PYQ] [FIG] Task (Detect structure e.g., S → NP VP). Challenge (Structural Ambiguity (Scope: "old men and women"; PP Attachment: "I saw the boy with a telescope"). Garden Path Sentences).
* Semantic Analysis (Meaning Representation): [PYQ] Task (Represent meaning (Predicate Calculus, Semantic Nets, Frames). Identify semantic roles). Challenge (Semantic Role Labeling (SRL) Ambiguity ("Visiting aunts can be a nuisance"). WSD ("strong interest" vs "pay interest")).
* Pragmatics (Contextual Meaning / User Intent): [PYQ] Task (Model user intention, understand meaning beyond literal. Requires world knowledge). Challenge (Very hard. (Tourist asking about sandals implies wanting them)).
* Discourse (Multi-Sentence Analysis): Task (Processing sentence sequences, understanding inter-sentence relationships). Challenge (Anaphora / Co-reference Resolution ("John put carrot on plate and ate it.")).
35. **Specific NLP Tasks [PYQ]**
* Word Segmentation: Breaking character strings into words (e.g., for Chinese).
* Part-of-Speech (POS) Tagging: [PYQ] Annotating words with POS tags.
* Phrase Chunking: Finding non-recursive NPs, VPs.
* Parsing: [PYQ] Full syntactic analysis, generating parse tree.
* Word Sense Disambiguation (WSD): [PYQ] Determining correct meaning of ambiguous words.
* Semantic Role Labeling (SRL): Identifying semantic roles of phrases relative to verbs.
* Textual Entailment: Determining if one sentence logically implies another.
* Anaphora/Co-reference Resolution: Identifying phrases referring to the same entity.
* Information Extraction (IE): [PYQ] Named Entity Recognition (NER) [PYQ] (Identifying names (people, places, orgs)); Relation Extraction (Identifying relations between entities).
* Question Answering (QA): [PYQ] Answering natural language questions from text.
* Text Summarization: [PYQ] Producing short summaries.
* Sentiment Analysis: [PYQ] Extracting subjective info, polarity.
* Machine Translation (MT): [PYQ] Translating between languages.
36. **Why is Ambiguity Resolution Hard and What Knowledge is Needed? [PYQ]**

* Requirement: Correct interpretation needs combining knowledge from multiple levels: Syntax (Grammatical structure); Semantics (Word meanings); Pragmatics (Contextual understanding); World Knowledge (Common sense facts).
37. **Approach to Acquiring Knowledge for NLP: Traditional vs. Modern**
* Traditional / Rationalist Approach: Manual knowledge acquisition. Human specialists specified rules (lexicons, grammars). Problems: Difficult, time-consuming, error-prone, brittle, expensive.
* Modern / Empirical / Statistical / Learning Approach: [FIG] Uses ML to automatically acquire knowledge from data (annotated text corpora). Process: Labeled data → ML Algorithm → NLP System. Benefits: More robust, adaptable, handles variability better. Dominant since 1990s.
38. **Key Milestones in NLP History [PYQ]**
* 1950s: Shannon (probabilistic models), Chomsky (formal grammars), first parsers, Bayesian OCR.
* 1960s: MIT AI Lab (BASEBALL, ELIZA), semantic nets, Brown Corpus.
* 1970s: Deeper understanding systems (SHRDLU), Prolog for parsing, early HMMs for speech.
* 1980s: More complex grammars (unification, TAG), symbolic discourse, initial statistical POS tagging.
* 1990s: "Statistical Revolution": Rise of empirical methods, annotated corpora (Penn Treebank), statistical MT, robust parsers, IE systems.
* 2000s: Diverse ML methods (SVMs, MaxEnt, CRFs), shared tasks/corpora, unsupervised/semi-supervised focus, shift to semantics (WSD, SRL).
* 2010s-Present: Deep Learning dominates: Word2vec, GloVe, CNNs, RNNs, Transformers (BERT), end-to-end learning, large pre-trained models, XAI. Turing Award 2018 for Deep Learning (Bengio, Hinton, LeCun).
39. **Types of Machine Learning Relevant to NLP [PYQ]**
* Machine Learning: Acquiring models from data/experience. Learning parameters, structure, hidden concepts.
* Types: Unsupervised Learning [PYQ] (No teacher feedback; detect patterns e.g., clustering); Reinforcement Learning [PYQ] (Feedback via rewards/punishments); Supervised Learning [PYQ] (Given examples of correct input-output pairs - labeled data).
* Classification: [PYQ] Discrete outputs (e.g., spam/ham, topic, sentiment).
* Regression: Continuous outputs.
40. **How does Supervised Learning Work (e.g., Classification)?**
* Goal: Given training set (x1,y1)...(xn,yn) where yi=f(xi) for unknown f, discover hypothesis h ≈ f.
* Process: 1. Data: Collect labeled instances. Split: Training, Validation (Held-Out), Test sets. 2. Features: Define attributes characterizing input x. 3. Experimentation Cycle: Learn model parameters on Training set; Tune hyperparameters on Validation set; Evaluate final performance on Test set. Crucial: Never "peek" at Test set during training/tuning.
* Evaluation: Accuracy, Precision, Recall, F1-score.
* Challenge: Overfitting: Fitting training data too closely, poor generalization.
41. **How is Text Classification used in NLP? [PYQ]**
* Task: Assigning predefined labels/categories to documents.
* Examples: Topic Labeling, Genre Classification, Opinion/Sentiment Analysis, Spam Detection.
* Methods History: Manual Classification, Hand-coded Rules, Supervised Machine Learning (k-NN, Naive Bayes, SVMs, Deep Learning - widely used).
42. **Why has Deep Learning become prominent in NLP? [PYQ]**

* Transforms how machines understand/interact with complex data. Mimics brain's neural networks.
* Limitations of Shallow ML: Required hand-crafted features, suffered from curse of dimensionality.
* Deep Learning (DL) Advantages: Learns representations (features) automatically from data; Effective at complex patterns via multiple layers; Flexible, (almost) universal framework. Can learn supervised/unsupervised; Effective end-to-end learning. Can leverage large training data; Outperformed other ML ~2010 (Speech, Vision, then NLP).
* Key DL Developments in NLP: Distributed Representations (Word2vec, ELMo, BERT), Neural Architectures (CNNs, RNNs/LSTMs, Transformers), RL apps, Attention.
* Impact: Led to state-of-the-art results in many NLP tasks.
43. **What is Stemming? [PYQ]**
* NLP process to reduce inflected/derived words to their word stem, base, or root form.
* Stem isn't necessarily a linguistically correct root; often a truncated version.
* Examples: "running" → "run"; "studies" → "studi"; "beautiful" → "beauti".
* Other Related NLP Preprocessing Processes: [PYQ]
* 1. Tokenization: [PYQ] Breaking text into smaller units (tokens - words, punctuation).
* 2. Lowercasing: Converting all text to lowercase.
* 3. Stop Word Removal: [PYQ] Removing common, low-meaning words ("a", "the", "is").
* 4. Punctuation Removal/Handling: [PYQ] Removing/replacing punctuation.
* 5. Lemmatization: [PYQ] Reduces words to base/dictionary form (lemma). Unlike stemming, considers meaning & POS. More accurate than stemming; produces actual dictionary words. Slower, more computationally intensive than stemming. Examples: "running" → "run"; "studies" → "study"; "better" → "good".
* 6. Part-of-Speech (POS) Tagging: [PYQ] (Covered earlier) Assigning grammatical category.
* 7. Named Entity Recognition (NER): [PYQ] (Covered earlier) Identifying/categorizing named entities.
* 8. Sentence Segmentation (Sentence Boundary Disambiguation): Dividing text into sentences.

AI UNIT - 4 CONCISE

UNIT 4: Clustering, Uncertainty, Expert Systems, Fuzzy Logic & ML Intro

Part 1: Fuzzy Logic [PYQ]

1. **Introduction to Fuzzy Logic [PYQ]**
    - "Fuzzy" Meaning: [PYQ] Things that are not clear, vague, or imprecise.
    - Purpose: Handles situations where decisions/statements are not strictly true or false.
    - Provides many values between true (1) and false (0), offering flexibility.
    - Allows representation of human-like linguistic uncertainty. [FIG] (Boolean Yes/1, No/0 vs. Fuzzy Very much/0.9, Little/0.25, Very less/0.1)
    - Introduced by: Lotfi Zadeh in 1965 based on Fuzzy Set Theory.
    - Comparison with Boolean Logic: Boolean (Only two possibilities 0/1 - absolute false/true); Fuzzy (Multiple possibilities between 0 and 1 - partially false/true).

- Implementation: Can be in hardware or software (micro-controllers, workstations, network systems).
  1.1. **Characteristics of Fuzzy Logic [PYQ]**
- Flexible and easy to understand/implement.
- Helps minimize human-created logic complexity.
- Best for problems suitable for approximate or uncertain reasoning.
- Often offers two values (denoting possible solutions for a problem/statement).
- Allows users to build non-linear functions of arbitrary complexity.
- Everything is a matter of degree.
- Any logical system can be "fuzzified."
- Based on natural language processing.
- Used by quantitative analysts for algorithm execution improvement.
- Allows users to integrate with programming.
  1.2. **Fuzzy Thinking & Fuzzy Sets**
- Fuzzy Thinking: [PYQ] Deals with concepts that are not precise or crisp, but rather "fuzzy" or vague (e.g., "tall man," "high profit"). Classical set theory struggles with this.
- Fuzzy Set Theory: Formalized by Lotfi Zadeh (1965) to generalize classical set theory.
- Crisp Set: [FIG] Elements have either full (1) or no (0) membership. Defined by a characteristic function $\chi A: X \rightarrow \{0, 1\}$.
- Fuzzy Set: [PYQ] [FIG] Elements have a degree of membership between 0 and 1. Defined by a membership function (MF) $\mu A: X \rightarrow [0, 1]$. [PYQ]
  - $\mu A(x) = 1$: x is fully in A. $\mu A(x) = 0$: x is not in A. $0 < \mu A(x) < 1$: x is partly in A.
  - Formal definition: $A = \{(x, \mu A(x)) \mid x \in X\}$.
  - Totally characterized by its MF.
    1.4. **Fuzzy Terminology [PYQ]**
- Universe of Discourse (U): The range of all possible values for an input to the fuzzy system.
- Fuzzy Set: [PYQ] A set whose elements have degrees of membership (between 0 and 1) in that set. Empowers members to have different grades of membership.
- Membership Function (μ): (Defined above) Forms the basis of a fuzzy set.
- Support of a fuzzy set (S_f): [PYQ] The set of all elements in the universal set U that have a non-zero membership value in fuzzy set f. Example: S_adult = {21,30,35,40,45,60,70} for adult ages.
- Depiction of a fuzzy set: Often represented as a sum of (membership_value / element_value) terms, e.g., Old = 0.1/21 + 0.3/30 + ... + 1/70. (Note: slash is not algebraic division).

2. **Fuzzy Membership Functions (MF) [PYQ] [FIG]**
   - Definition: Defines the fuzzy set; provides a measure of similarity of an element to the fuzzy set.

- Determination: Chosen by user arbitrarily (based on experience); Designed using machine learning methods (ANNs, GAs).
- Shapes of MFs: [PYQ] [FIG] Triangular MF (Defined by 3 points a,b,c); Trapezoidal MF (Defined by 4 points α,β,γ,δ); Gaussian MF (Bell-shaped curve defined by center (c), width (s), fuzzification factor (m). μA(x,c,s,m) = exp[-0.5 * (|x-c|/s)^m]).
- Other shapes: Piecewise-linear, bell-shaped, etc.

3. **Crisp Sets vs. Fuzzy Sets [PYQ] [FIG]**
   - Crisp Sets: Mutually exclusive membership (either in or out).
   - Fuzzy Sets: Define a degree of membership, allowing an element to belong to multiple fuzzy sets simultaneously with different degrees.
   - Example: "Tall Men" [FIG] Crisp (A fixed height e.g., 180cm is the cutoff); Fuzzy (Men have a degree of "tallness" e.g., 175cm might be 0.6 tall, 185cm might be 0.9 tall).
     3.1. **Difference Between Fuzzy and Classical (Crisp) Set Theory [PYQ]**
   - Boundaries: Classical (Sharp boundaries - element either in or out. Defined by exact boundaries 0 and 1); Fuzzy (Un-sharp boundaries - degrees of membership. Defined by ambiguous boundaries).
   - Uncertainty: Classical (No uncertainty about boundary location); Fuzzy (Always exists uncertainty about boundary location).
   - Usage: Classical (Widely used in digital system design); Fuzzy (Mainly used for fuzzy controllers).
4. **Fuzzy Set Operations [PYQ] [FIG] (for Union, Intersection, Complement)**
   - Defined in terms of membership functions (Standard Fuzzy Operations - Min/Max):
   - Union (A U B): [PYQ] μAUB(x) = max(μA(x), μB(x)) (Corresponds to OR).
   - Intersection (A ∩ B): [PYQ] μA∩B(x) = min(μA(x), μB(x)) (Corresponds to AND).
   - Complement (Ā or not A): [PYQ] μĀ(x) = 1 - μA(x).
   - Fuzzy Set Inclusion (A ⊂ B): If and only if ∀x, μA(x) ≤ μB(x).
   - Fuzzy Set Equality (A = B): If and only if ∀x, μA(x) = μB(x).
   - Properties: Commutativity, associativity, idempotency, distributivity hold. De Morgan's laws hold.
   - Laws that FAIL in Min-Max Fuzzy Logic: [PYQ] Law of Excluded Middle (A∪Ā ≠ X, max not always 1 but ≥0.5); Law of Contradiction (A∩Ā ≠ Ø, min not always 0 but ≤0.5).
5. **Fuzzy Relations [PYQ] [FIG]**
   - Definition: A fuzzy relation R on X x Y is a fuzzy subset of X × Y, characterized by a 2D membership function μR(x,y).
   - Representations: List of ordered pairs with membership grades {((x,y), μR(x,y))}; Matrix Representation [FIG] (Rows for X, columns for Y, cell (i,j) contains μR(xi, yj)).
   - Operations on Fuzzy Relations: Similar to fuzzy sets (Union, Intersection, Complement, Containment).

- Composition of Fuzzy Relations (e.g., R on XxY, S on YxZ → T on XxZ): [PYQ] [FIG]
  - Max-Min Composition (T = R∘S): [PYQ] μT(x,z) = max_y {min(μR(x,y), μS(y,z))}. Essentially, find "strongest path" through intermediate y.
  - Max-Product Composition: μT(x,z) = max_y {μR(x,y) * μS(y,z)}. Used when max-min not mathematically tractable.

6. **Linguistic Variables and Hedges [PYQ] [FIG]**
   - Linguistic Variable: [PYQ] A variable whose values are words or sentences in a natural or artificial language (i.e., fuzzy sets). E.g., "Temperature" values: "cold," "warm," "hot". "Ram is tall": "Ram's height" takes value "tall".
   - Hedges (Fuzzy Set Qualifiers): [PYQ] [FIG] Adverbs that modify the shape of fuzzy sets (linguistic values). Examples: very, somewhat, quite, more or less, slightly. Mathematical operations on MFs: very A (μA(x))^2 (concentration); slightly A / somewhat A (sqrt(μA(x))) (dilation); A little (μA(x))^1.3; Extremely A (μA(x))^3.
7. **Fuzzification and Defuzzification [PYQ] [FIG]**
   - Fuzzification: [PYQ] [FIG] Transforming crisp (numerical) inputs into fuzzy sets (degrees of membership to linguistic terms). Maps input values to membership functions.
   - Defuzzification: [PYQ] [FIG] Mapping fuzzy output sets (from inference process) back to a single crisp (numerical) output value. Needed for real-world actuators. Methods: Centroid (Center of Gravity), Max-Membership, Weighted Average, etc.
   - Operation of Fuzzy System: Crisp Input → Fuzzification → Fuzzy Input → Rule Evaluation (Inference) → Fuzzy Output → Defuzzification → Crisp Output. [FIG]
     7.1. **Architecture of a Fuzzy Logic System (FLS) [PYQ] [FIG]**
   - Four Main Components:
     - 1. Rule Base (Knowledge Base): [PYQ] Stores a set of rules (often IF-THEN conditions) given by experts. Used for controlling decision-making systems. Modern fuzzy theory offers methods for designing/tuning fuzzy controllers, reducing fuzzy set rules.
     - 2. Fuzzification Module: [PYQ] Transforms crisp system inputs (numerical values from sensors) into fuzzy sets (linguistic variables with degrees of membership). Divides input signals into fuzzy states (e.g., Large Positive (LP)).
     - 3. Inference Engine (Decision-Making Unit): [PYQ] Main component; processes fuzzy input and rules. Finds matching degree between current fuzzy input and rules. Determines which rule(s) to fire based on input. Combines fired rules to develop control actions.
     - 4. Defuzzification Module: [PYQ] Converts the fuzzy output sets (from inference engine) back into a single crisp numerical value that can be used as a control action. Example methods: Centre of Gravity (Centroid), Maxima methods. [FIG] (Defuzzification graphs).

8. **Fuzzy Rule-Based Systems / Fuzzy Classifiers [PYQ] [FIG]**
    - Use fuzzy IF-THEN rules.
    - Components of Fuzzy Logic System Architecture: [PYQ] [FIG] Rule Base; Fuzzification Module; Inference Engine; Defuzzification Module.
    - Fuzzy IF-THEN Rules: Antecedent (IF part) (IF x is A AND y is B); Consequent (THEN part) (THEN z is C). (A,B,C are fuzzy sets).
    - Inference Process (Example for AND antecedent): Firing Level (ai) of a rule (min{μA(input_x),μB(input_y)} for AND, max for OR). If multiple rules same consequence, combine strengths (max/OR). Overall classifier output by choosing class with max combined strength.
9. **Applications of Fuzzy Logic [PYQ]**
    - Businesses (Decision-making support); Automotive systems (Controlling traffic/speed, transmission); Defence (Target recognition); Pattern Recognition and Classification (Handwriting recognition, image searching); Securities; Microwave oven (Setting power); Modern control systems (Expert systems); Finance (Predicting stock market); Controlling brakes; Industries of chemicals (Controlling pH); Industries of manufacturing (Milk/cheese optimization); Vacuum cleaners, washing machines (timings); Heaters, air conditioners, humidifiers.
10. **Differences between Fuzzy Logic and Probability [PYQ]**
    - Fuzzy Logic: Deals with imprecision or vagueness of facts/statements. Membership in vaguely defined sets. E.g., "The water is hot." (Degree of hotness).
    - Probability: Deals with uncertainty or chances of an event occurring (likelihood of a precise event). E.g., "There is an 80% chance it will rain tomorrow." (Event precise: rain/no rain).
    - Fuzzy sets are based on vague definitions, not randomness.
    - Lotfi Zadeh: Fuzzy logic different in character from probability, not a replacement. Possibility theory is fuzzy alternative to probability.
11. **Advantages of Fuzzy Logic [PYQ]**
    - Methodology similar to human reasoning. Easily understandable structure. Does not need large memory (algorithms described with fewer data). Widely used, provides effective solutions for high-complexity problems. Simple (based on set theory of mathematics). Allows control of machines and consumer products. Shorter development time compared to conventional methods. Flexibility allows easy addition/deletion of rules in FLS.
12. **Disadvantages of Fuzzy Logic [PYQ]**
    - Run time can be slow, takes time to produce outputs. Users understand it easily only if systems are simple. Possibilities produced are not always accurate. Multiple ways to solve a statement can lead to ambiguity. Not suitable for problems requiring high accuracy. Systems need extensive testing for verification and validation.
13. **Case Study: Fuzzy Room Cooler [FIG]**

- Fuzzy Regions/Terms: Input Parameters (Temperature, Fan Motor Speed/Pressure); Fuzzy Terms for Temperature (Cold, Cool, Moderate, Warm, Hot); Fuzzy Terms for Fan Motor Speed (Slack, Low, Medium, Brisk, Fast); Output (Flow-rate) (Strong-Negative, Negative, Low-Negative, Low-Positive, High-Positive).
- Fuzzy Profiles: Membership function graphs for each fuzzy term. [FIG] (Temp, Motor Speed, Flow Rate graphs)
- Fuzzy Rules: Form triggers for the fuzzy engine (linguistic IF-THEN rules). [FIG] E.g., R1: IF temperature is HOT AND fan motor speed is SLACK THEN flow-rate is HIGH-POSITIVE.
- Fuzzification: Mapping crisp sensor inputs (e.g., temp=42°C, speed=31 rpm) to membership degrees in respective fuzzy regions. [FIG]
- Inference: (Implicit in combining rules) Rules fire based on fuzzified inputs. If multiple rules apply, their outputs are combined.
- Defuzzification: Converting the combined fuzzy output for flow-rate into a single crisp value for the pump. [FIG] (Showing how outputs of multiple rules combined for final crisp output using centroid method). Common methods: Centre of Gravity, Composite Maxima.
- NeuroFuzzy Room Cooler: [FIG] (Diagram showing a neural network structure implementing the fuzzy logic system for the room cooler).

Part 2: K-Means Clustering [PYQ]

1. **What is K-Means Clustering? [PYQ] [FIG]**
   - Popular unsupervised machine learning algorithm.
   - Partitions a dataset into a pre-defined number (K) of clusters.
   - Goal: Group similar data points together; discover underlying patterns/structures.
   - Centroid-based / Distance-based: Calculates distances to assign points to clusters. Each cluster associated with a centroid.
   - Property of Clusters: Points within a cluster should be similar (minimize intra-cluster distance); Points in different clusters should be dissimilar (maximize inter-cluster distance).
   - Main Objective: Minimize the sum of squared distances between data points and their respective cluster centroids (WCSS - Within-Cluster Sum of Squares).
2. **How K-Means Clustering Works? [PYQ] [FIG]**
   - 1. Initialization: Randomly select K points from the dataset as initial cluster centroids.
   - 2. Assignment Step: For each data point, calculate its distance (e.g., Euclidean) to all K centroids. Assign point to cluster whose centroid is closest.
   - 3. Update Centroids Step: Recalculate the centroid of each cluster by taking the mean of all data points assigned to that cluster.

-
  4. Repeat: Repeat steps 2 and 3 until convergence (centroids no longer change significantly, or max iterations reached).
-
  5. Final Result: Algorithm outputs final cluster centroids and data point assignments.

3. **What is Clustering (General)? [PYQ]**
   - Technique in data mining & ML that groups similar objects into clusters.
   - Unsupervised Learning Problem: No target variable to predict. Aims to find inherent structure in data.
   - Example: Bank segmenting customers based on income for targeted credit card offers. [FIG]
4. **Properties of K-Means Clustering (Reiteration of Cluster Properties)**
   - First Property: Data points in a cluster should be similar to each other (high cohesiveness).
   - Second Property: Data points from different clusters should be as different as possible (high separation).

Part 3: Bayesian Networks [PYQ]

1. **What are Bayesian Networks (BNs)? [PYQ] [FIG]**
   - Type of Probabilistic Graphical Model (PGM).
   - Represents a set of random variables and their conditional dependencies using a Directed Acyclic Graph (DAG).
   - Uses Bayesian inference for probability computations.
   - Nodes: Random variables (discrete or continuous).
   - Edges (Arcs): Represent conditional dependencies (often interpreted as causal relationships). If edge (A,B) exists, A is a parent of B.
   - Also called: Bayes network, belief network, decision network, Bayesian model.
2. **Key Concepts of Bayesian Networks**
   - Conditional Probability Table (CPT): [PYQ] [FIG] Each node has a CPT specifying P(Node | Parents(Node)). For root nodes (no parents), CPT is just P(Node).
   - Local Markov Property: [PYQ] A node is conditionally independent of its non-descendants given its parents. This allows simplification of the Joint Probability Distribution (JPD).
   - Joint Probability Distribution (JPD): [PYQ] BNs provide a compact, factorized representation of the JPD. $P(X1, ..., Xn) = \Pi P(Xi \mid Parents(Xi))$ for all nodes i.
   - Conditional Probability: $P(A|B) = P(A \cap B) / P(B)$.
3. **Bayesian Network Example (Student Marks) [PYQ] [FIG]**
   - Variables: Exam Level (e), IQ Level (i), Aptitude Score (s), Marks (m), Admission (a).
   - Dependencies: e,i $\rightarrow$ m; i $\rightarrow$ s; m $\rightarrow$ a. (e and i are parents of m, etc.)

  - JPD: P(a,m,i,e,s) = P(a|m) * P(m|i,e) * P(s|i) * P(i) * P(e).
  - Given CPTs for each variable, can compute probability of specific events (e.g., P(admission=yes, marks=pass, IQ=low, exam=difficult, aptitude=low)).

4. **Bayesian Network Example (Burglary Alarm) [PYQ] [FIG]**
   - Variables: Burglary (B), Earthquake (E), Alarm (A), David Calls (D), Sophia Calls (S).
   - Dependencies: B,E → A; A → D; A → S.
   - JPD: P(D,S,A,B,E) = P(D|A) * P(S|A) * P(A|B,E) * P(B) * P(E) (assuming B and E independent initially, or P(B|E)P(E)).
   - Used to calculate probability of complex events, e.g., P(Alarm | DavidCalls, ¬Burglary, ¬Earthquake).
5. **Semantics of Bayesian Network**
   - Two ways to understand: 1. As a representation of the Joint Probability Distribution. (Useful for construction). 2. As an encoding of a collection of conditional independence statements. (Useful for designing inference procedures).

Part 4: Expert Systems [PYQ]

1. **What is an Expert System (ES)? [PYQ]**
   - Computer program designed to solve complex problems and provide decision-making ability like a human expert within a specific domain.
   - Extracts knowledge from its Knowledge Base (KB) using reasoning and inference rules.
   - First ES developed in 1970s (e.g., DENDRAL, MYCIN).
   - Uses both facts and heuristics.
   - Designed for specific domains (medicine, science, finance).
   - Note: ES assists human experts, does not replace them.
2. **Examples of Expert Systems [PYQ]**
   - DENDRAL: Chemical analysis (organic chemistry).
   - MYCIN: [PYQ] Medical diagnosis (bacterial infections, antibiotic recommendation). Used backward chaining.
   - PXDES: Determines type/level of lung cancer from images.
   - CaDeT: Diagnostic support for early cancer detection.
3. **Characteristics of Expert Systems**
   - High Performance: Solves complex domain problems with high efficiency/accuracy.
   - Understandable: Responds in a way easily understandable by user (human language input/output).
   - Reliable: Generates efficient and accurate output.
   - Highly responsive: Provides results for complex queries quickly.

4. **Components of an Expert System [PYQ] [FIG]**
    - A. User Interface: [PYQ] Allows non-expert user to interact with ES, input queries, receive responses in a readable format.
    - B. Inference Engine (Rule Engine / "Brain"): [PYQ] Main processing unit. Applies inference rules to the KB to derive conclusions or deduce new info. Extracts knowledge from KB.
        - Types: Deterministic Inference Engine (Conclusions assumed true - based on facts/rules); Probabilistic Inference Engine (Contains uncertainty in conclusions - based on probability).
        - Modes (Chaining): [PYQ] Forward Chaining, Backward Chaining.
    - C. Knowledge Base (KB): [PYQ] Stores knowledge acquired from domain experts. Contains information and rules for a particular domain/subject.
        - Components of KB: Factual Knowledge (Based on facts, accepted by knowledge engineers); Heuristic Knowledge (Based on practice, ability to guess, evaluation, experiences).
        - Knowledge Representation: [PYQ] Formalizes knowledge (e.g., IF-ELSE rules).
        - Knowledge Acquisition: [PYQ] Process of extracting, organizing, structuring domain knowledge, specifying rules, storing in KB.
5. **Development of Expert System (e.g., MYCIN)**
    -
        1. Feed ES with expert knowledge (e.g., medical info for MYCIN). 2. KB is updated. Doctor tests with new problem (patient details, symptoms). 3. ES may use questionnaire for general patient info. 4. System finds solution by applying IF-THEN rules (inference engine) to facts in KB. 5. Provides response via user interface.
6. **Participants in ES Development**
    - Expert: Provides specialized domain knowledge.
    - Knowledge Engineer: Gathers knowledge from experts, codifies it into the system.
    - End-User: Person/group (may not be expert) needing solution/advice from ES.
7. **Why Expert Systems? (Capabilities & Need)**
    - Capabilities: Advising, decision-making, demonstrating devices, problem-solving, explaining problems, interpreting input, predicting results, diagnosis.
    - Need / Advantages: No memory limitations (can store vast data); High Efficiency (if KB is correct); Consolidated Expertise in a domain (mixes knowledge from multiple experts); Not affected by emotions; High security; Considers all facts; Regular updates improve performance; Highly reproducible; Usable in risky places (unsafe for humans).
    - Limitations: [PYQ] Wrong output if KB has wrong info; Cannot produce creative output; High maintenance/development costs; Knowledge acquisition is difficult; Domain-specific (one ES for each domain); Cannot learn from itself; requires manual updates.
8. **Applications of Expert Systems [PYQ]**

    - Designing and manufacturing domain (e.g., camera lenses).
    - Knowledge domain (publishing relevant knowledge, e.g., tax advisor).
    - Finance domain (fraud detection, loan advice).
    - Diagnosis and troubleshooting of devices (medical diagnosis was an early area).
    - Planning and Scheduling.

Part 5: Machine Learning (Introduction - from Aditya College Notes) [PYQ]

1. **Introduction: AI, Machine Learning (ML), Deep Learning (DL) [PYQ] [FIG]**
    - Artificial Intelligence (AI): Branch of CS to create intelligent machines that behave, think, and decide like humans.
    - Machine Learning (ML): [PYQ] Enables computers to learn automatically from past data/experience without explicit programming. Uses algorithms to build models and make predictions. Introduced by Arthur Samuel (1959).
    - Deep Learning (DL): [PYQ] Subset of ML based on Artificial Neural Networks (ANNs), inspired by human brain. Learns feature hierarchies. [FIG] (ANN diagram)
2. **Types of Machine Learning Systems [PYQ] [FIG]**
    - Categorized by: Human supervision (Supervised, Unsupervised, Semi-supervised, Reinforcement); Incremental learning (Online vs. Batch); Generalization method (Instance-based vs. Model-based).
    - A. Supervised Learning: [PYQ] Trained on labeled data (input-output pairs). Machine predicts output based on training. Goal: Map input x to output y.
        - Categories: Classification [PYQ] (Output variable is categorical Yes/No, Spam/Ham. Algorithms: Random Forest, Decision Tree, Logistic Regression, SVM); Regression [PYQ] (Output variable is continuous - market trends, weather. Algorithms: Linear Regression, Decision Tree, Lasso Regression).
        - Advantages: Exact idea of object classes (from labels), predicts based on prior experience.
        - Disadvantages: Can't solve complex tasks, wrong prediction if test data differs from training, high computational time to train.
    - B. Unsupervised Learning: [PYQ] Trained on unlabeled data. No explicit supervision. Goal: Find hidden patterns, similarities, structures in data.
        - Categories: Clustering [PYQ] (Grouping objects into clusters - similar within, dissimilar between. E.g., K-Means, DBSCAN); Association Rule Learning [PYQ] (Finding interesting relations/dependencies between variables e.g., Apriori, Eclat).
        - Advantages: Can use for complex tasks, easier to get unlabeled data.
        - Disadvantages: Output less accurate, harder to work with (no mapped output).
    - C. Semi-Supervised Learning: [PYQ] Lies between supervised and unsupervised. Uses a combination of labeled and unlabeled data. Overcomes drawbacks of both. Advantages: Simple,

efficient, solves drawbacks of purely supervised/unsupervised. Disadvantages: Results may not be stable, not for network-level data, accuracy can be low.

- D. Reinforcement Learning (RL): [PYQ] [FIG] Agent learns by interacting with an environment (trial and error). Receives feedback as rewards (for good actions) or punishments (for bad actions). Goal: Maximize cumulative reward. No labeled data initially.
    - Categories: Positive RL (Adding something to increase behavior tendency); Negative RL (Avoiding negative condition to increase behavior tendency).
    - Applications: Game theory, robotics, text mining, video games.

3. **Main Challenges of Machine Learning [PYQ]**
    - Lack of Quality Data (noisy, incorrect, incomplete). Fault in specific applications (e.g., credit card fraud detection nuances). Getting Bad Recommendations. Talent Deficit (lack of experts). Implementation Complexity (integrating new ML with existing systems). Making Wrong Assumptions (e.g., handling missing data). Deficient Infrastructure (ML needs large data stirring abilities). Algorithms Becoming Obsolete as Data Grows. Absence of Skilled Resources. Customer Segmentation (dynamic behavior). Complexity of Models (many still experimental). Slow Results (training/inference can be time-consuming). Maintenance (models need updates as data/actions change). Concept Drift (target variable changes over time). Data Bias. High Chances of Error (biased programming/datasets). Lack of Explainability ("Black box" models).
4. **Statistical Learning: Introduction [PYQ]**
    - Mathematical analysis of data, especially to interpret models and quantify uncertainty.
    - Two Major Goals for Modeling Data: 1. Accurately predict future quantity of interest. 2. Discover unusual or interesting patterns.
    - Feature, Response: Input/feature vector x, predict output/response variable y. Prediction function g(x) is the guess for y.
    - Regression, Classification: Regression (y is real-valued); Classification (y is from a finite set - categories).
    - Loss Function: Measures accuracy of prediction g(x) w.r.t. true response y. E.g., Squared error loss (y - g(x))^2 for regression.
5. **Training and Test Loss [PYQ]**
    - Risk l(g): Expected loss of a prediction function g. Hard to compute.
    - Empirical Risk / Training Loss eT(g): Average loss over the training sample T. Unbiased estimator of risk if g is fixed. eT(g) = (1/n) Σ Loss(Yi, g(Xi)).
    - Optimal Prediction Function g:* Minimizer of true risk l(g).
    - Learner gT:* Minimizer of training loss eT(g).
    - Generalization Risk / Test Loss l(gT*): [FIG] Accuracy of learner on new, unseen data. Measured on a fixed test set τ. lτ(gT*) = E[Loss(Y, gT*(X))] (expected loss on new data). Often estimated by average loss on a test sample T'.
6. **Tradeoffs in Statistical Learning [PYQ]**

- Goal: Make generalization risk small.
- Decomposition of generalization risk (for squared error loss): E[(Y-gT*(X))^2] = E[(Y-f(X))^2] (Irreducible Error / Bayes Error - Variance of Y given X. Optimal possible error. Cannot be reduced by any model) + E[(f(X)-E[gT*(X)])^2] (Squared Bias / Approximation Error - How much average model E[gT*(X)] differs from true function f(X). Due to model too simple) + E[(gT*(X)-E[gT*(X)])^2] (Variance / Estimation Error - How much specific model gT*(X) varies around its average. Due to model too complex/fitting noise).
- Bias-Variance Tradeoff: [PYQ] [FIG] Simple models (High bias, low variance); Complex models (Low bias, high variance); Need to find a balance for optimal generalization.

7. **Estimating Risk [PYQ]**
   - 1. In-Sample Risk: [FIG] Training loss (eT(g)) is not a good estimate of generalization risk due to overfitting.
   - 2. Cross-Validation (CV): [PYQ] [FIG] Method to estimate generalization risk by splitting data. K-Fold CV (Data split K folds. Train K-1, test 1. Repeat K times. Average results); Leave-One-Out CV (LOOCV) (K=N); Helps in model selection and hyperparameter tuning.
8. **Sampling Distributions of Estimators**
   - Estimators (e.g., model parameters) are statistics (functions of random data), so they have sampling distributions.
   - Distribution depends on sample size.
   - Used to: Calculate probability of estimator differing from true parameter; Obtain interval estimates (confidence intervals); Compare estimators (e.g., using Mean Squared Error).
9. **Empirical Risk Minimization (ERM) [PYQ] [FIG]**
   - Principle: Find the model/hypothesis h from a hypothesis class H that minimizes the empirical risk (average loss on the training sample). h_ERM = argmin_h∈H { (1/N) Σ Loss(h(Xi), Yi) }.
   - Fundamental concept in ML.
   - Theory behind ERM explains VC-dimension, PAC Learning.
   - ERM is a nice idea, if used with care (can lead to overfitting if hypothesis class is too complex or data is noisy).

Part 6: Perception and Action (General AI Concept) [PYQ]

- **Perception: [PYQ]** The process by which an agent acquires information about its environment through its sensors. Involves sensing raw data (light, sound, touch, etc.) and interpreting it into a meaningful representation of the state of the environment. Examples: Computer vision (interpreting images), speech recognition (interpreting audio).
- **Action: [PYQ]** The process by which an agent affects its environment through its actuators. Decisions made by the agent (based on its goals, perceived state, and internal

knowledge/reasoning) lead to actions. Examples: Robot moving its arm, a software agent displaying information, self-driving car turning the wheel.

- **Perception-Action Cycle: [PYQ] [FIG]** Fundamental loop in intelligent agents: 1. Agent perceives the environment. 2. Agent reasons/thinks/plans based on percepts and internal state/goals. 3. Agent acts upon the environment. 4. The environment changes (partly due to agent's action), leading to new percepts. This cycle is continuous and forms the basis of agent behavior. Learning often occurs by evaluating the outcomes of actions based on new perceptions.
- **Link to Learning Agents:** The "Performer" component of a learning agent is responsible for selecting actions based on its knowledge and current perception. The "Critic" evaluates these actions, and the "Learner" updates the knowledge or performer based on this evaluation, improving future perception-action mappings.