What is the Role of Planning in Artificial Intelligence?

Artificial intelligence is an important technology in the future. Whether it is intelligent robots, self-driving cars, or smart cities, they will all use different aspects of artificial intelligence!!! But Planning is very important to make any such AI project.

Even Planning is an important part of Artificial Intelligence which deals with the tasks and domains of a particular problem. Planning is considered the logical side of acting.

Everything we humans do is with a definite goal in mind, and all our actions are oriented towards achieving our goal. Similarly, Planning is also done for Artificial Intelligence.

For example, Planning is required to reach a particular destination. It is necessary to find the best route in Planning, but the tasks to be done at a particular time and why they are done are also very important.

That is why Planning is considered the

For example, Planning is required to reach a particular destination. It is necessary to find the best route in Planning, but the tasks to be done at a particular time and why they are done are also very important.

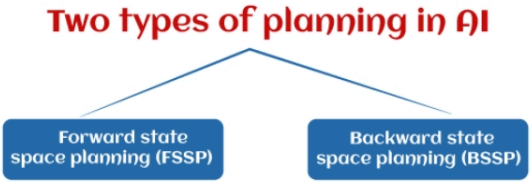
That is why Planning is considered the logical side of acting. In other words, Planning is about deciding the tasks to be performed by the artificial intelligence system and the system's functioning under domain-independent conditions.

What is a Plan?

We require domain description, task specification, and goal description for any planning system. A plan is considered a sequence of actions, and each action has its preconditions that must be satisfied before it can act and some effects that can be positive or negative.

So, we have **Forward State Space Planning (FSSP)** and **Backward State Space Planning (BSSP)** at the basic level.

Planning (BSSP) at the basic level.



1. Forward State Space Planning (FSSP)

FSSP behaves in the same way as forwarding state-space search. It says that given an initial state S in any domain, we perform some necessary actions and obtain a new state S' (which also contains some new terms), called a progression. It continues until we reach the target position. Action should be taken in this matter.

- **Disadvantage**: Large branching factor
- **Advantage**: The algorithm is Sound

2. Backward State Space Planning (BSSP)

BSSP behaves similarly to backward state-space search. In this, we move from the target state g to the sub-goal g, tracing the previous action to achieve that goal. This process is called regression (going back to the previous goal or sub-goal). These sub-goals should also be checked for

2. Backward State Space Planning (BSSP)

BSSP behaves similarly to backward state-space search. In this, we move from the target state g to the sub-goal g, tracing the previous action to achieve that goal. This process is called regression (going back to the previous goal or sub-goal). These sub-goals should also be checked for consistency. The action should be relevant in this case.

- **Disadvantages**: not sound algorithm (sometimes inconsistency can be found)
- **Advantage**: Small branching factor (much smaller than FSSP)

So for an efficient planning system, we need to combine the features of FSSP and BSSP, which gives rise to target stack planning which will be discussed in the next article.

What is planning in AI?

Planning in artificial intelligence is about decision-making actions performed by robots or computer programs to achieve a specific goal.

Execution of the plan is about choosing a sequence of tasks with a high probability of accomplishing a specific task.

Block-world planning problem

- The block-world problem is known as the Sussmann anomaly.
- The non-interlaced planners of the early 1970s were unable to solve this problem. Therefore it is considered odd.
- When two sub-goals, G1 and G2, are given, a non-interleaved planner either produces a plan for G1 that is combined with a plan for **G2** or vice versa.
- In the block-world problem, three blocks labeled 'A', 'B', and 'C' are allowed to rest on a flat surface. The given condition is that only one block can be moved at a time to achieve the target.

The start position and target position are shown in the following diagram.

Word planning problem

Components of the planning system

The plan includes the following important steps:

and skin) and actuators (limbs) to perform actions on these surroundings. The agent starts from the initial state and performs a series of actions in order to reach the goal state. For instance, a vacuum cleaner agent will perform actions of moving right and left, and sucking in dirt to reach the goal of successfully cleaning the environment.

Planning and its types

This activity of coming up with a sequence of actions in order to accomplish the target or goal is called as Planning. Planning can be Classical or Non-classical. In case of Classical Planning, the environment is fully observable, deterministic, static and discrete, whereas in case of Non-classical Planning, the environment is partially observable (i.e. the entire state of the environment is not visible at a given instant) or non-deterministic (or stochastic, i.e. the current state and chosen action cannot completely determine the next state of the environment).

State Space Search: Used in Problem Solving.

- It is a process used in A·I in which successive Configurations or States of an instance are Considered with intention of finding a GOAL State with desired property.
  - ↳ Problems are modelled as State Space
  - ↳ Representation:

S: (S, A, Action(s), Result(s,a), Cost(s,a))

- S: set of all possible States.
- A: Set of all possible actions
- Action(s): Func^n [which action is possible for Current state]
- Result(s,a): Func^n [State reached by performing action 'a' on state 's]
- Cost(s,a): Costing.

State Space: set of states in which a problem Can be.

EIGHT TILE PUZZLE

Start (S)

1	4	3
2		5
8	6	7

GOAL (G)

1	2	3
8		4
7	6	5

Actions Possible: up, down, Left, Right } Legal moves for a state.

'Right'

1	4	3
2	5	
8	6	7

right is not possible now
- up
- down
- left

Comparison Chart

BASIS FOR COMPARISON	FORWARD REASONING	BACKWARD REASONING
Basic	Data-driven	Goal driven
Begins with	New Data	Uncertain conclusion
Objective is to find the	Conclusion that must follow	Facts to support the conclusions
Type of approach	Opportunistic	Conservative
Flow	Incipient to consequence	Consequence to incipient

performance of a particular solution method. The state space search can be in forward and backward direction. The forward state space planning is also known as progression planning in which searching always takes place in forward direction. In backward search, it finds only the relevant actions. An agent with several immediate options of unknown values can decide what to do by first examining the different possible sequences of actions that lead to states of known values and then choosing the best one.

A state space search can be searched in two directions like from the inputs towards the goal or from the goals towards the inputs. In data driven search, one starts with the given facts of a problem and uses a set of legal moves or rules to change the states. This process is continued until it generates a path that satisfies the goal condition. In goal driven search, first determine the rules or legal moves that can be used to generate the goal and identifies the condition that can be applied to use these rules. These conditions form the new goals or sub goals for the search. One must continue the search process by working backwards through successive sub goals until it returns of moves or rules leading from the data to a goal, even if it performs this process backwards. Data driven search is suggested if one is provided with almost all the data at the time of formulation of the problem

Planning with State-Space Search

- The most straightforward approach of planning algorithm, is state-space search
  - Forward state-space search (Progression)
  - Backward state-space search (Regression)
- The descriptions of actions in a planning problem, and specify both preconditions and effects
- It is possible to search in both direction: either forward from the initial state or backward from the goal
- We can also use the explicit action and goal representations, to derive effective heuristics automatically.

Problem Formulation for Progression

- Initial state:
  - Initial state of the planning problem
- Actions:
  - Applicable to the current state.
  - First actions' preconditions are satisfied, Successor states are generated
  - Add positive literals to add list and negative literals to delete list.
- Goal test:
  - Whether the state satisfies the goal of the planning
- Step cost:
  - Each action is 1 (assumed)

Example : Transportation of air cargo between airports.

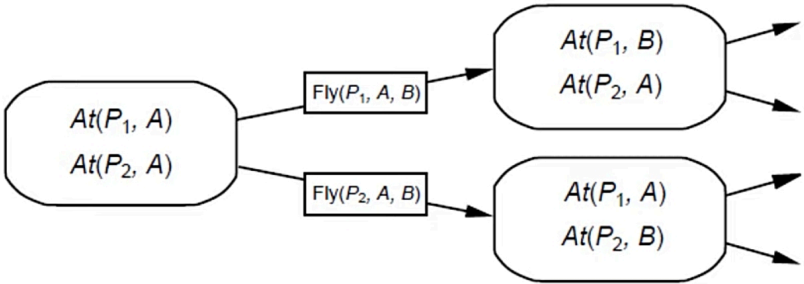
$$
\begin{aligned}
&Init(At(C_1, SFO) \land At(C_2, JFK) \land At(P_1, SFO) \land At(P_2, JFK) \\
&\quad \land Cargo(C_1) \land Cargo(C_2) \land Plane(P_1) \land Plane(P_2) \\
&\quad \land Airport(JFK) \land Airport(SFO)) \\
&Goal(At(C_1, JFK) \land At(C_2, SFO)) \\
&Action(Load(c, p, a), \\
&\quad \text{PRECOND: } At(c, a) \land At(p, a) \land Cargo(c) \land Plane(p) \land Airport(a) \\
&\quad \text{EFFECT: } \neg At(c, a) \land In(c, p)) \\
&Action(Unload(c, p, a), \\
&\quad \text{PRECOND: } In(c, p) \land At(p, a) \land Cargo(c) \land Plane(p) \land Airport(a) \\
&\quad \text{EFFECT: } At(c, a) \land \neg In(c, p)) \\
&Action(Fly(p, from, to), \\
&\quad \text{PRECOND: } At(p, from) \land Plane(p) \land Airport(from) \land Airport(to) \\
&\quad \text{EFFECT: } \neg At(p, from) \land At(p, to))
\end{aligned}
$$

Progression

- From initial state, search forward by selecting operators whose preconditions can be unified with literals in the state
- New state includes positive literals of effect; the negated literals of effect are deleted
- Search forward until goal unifies with resulting state
- This is forward state-space search using STRIPS operators

Forward (progression) state-space search

- Starting from the initial state and using the problem's actions to search forward for the goal state.



Planning Graph

- It is an algorithm for automated planning, developed by Avrim and Merrick in 1995.
- The Graph Plan's input is planning problem, expressed in STRIPS and produces a sequence of operations for reaching a goal state.

Planning Graph…

- Convert the planning problem structure into planning graph called as GRAPHPLAN, in the increment nature.
- It gives the relation between action and states, the precondition must be satisfy the action.
- The Planning graph is a layered graph, with alternate layers of propositions and actions.
  - Layer p0
  - Layer a1
  - Layer p1

Planning Graph…

- Propositional problem will look at, what the starting state, what the objects in the domains are, and it will produce all the possible actions, and works with those actions.
- We construct the planning graph from left to right,
  - we keep inserting actions and propositions, and
  - then actions and propositions
  - until we get the goal proposition appear on the proposition layer, and
  - they are not mutually exclusive.

Planning Graph…

- There are two states in the planning graph problem
  - Construct the planning graph
  - Search for solution
- If you cannot get solution, then extend the planning graph and search for solution, and keep doing that until you get the solution.

PLANNING GRAPHS

- Planning Graph can give better heuristic estimates.
- Here we can extract a solution directly from the planning graph, using a specialized algorithm such as GRAPHPLAN
- A planning graph consists of a sequence of **levels** that correspond to time steps in the plan, where level 0 is the initial state.
- Each level contains a set of literals and a set of actions.
- The literals are true at that time step, depending on, the actions executed at preceding time steps.
- Actions *could* have their preconditions, that should be satisfied at that time step, depending on the literals actually hold.

In short

- Planning graphs are an efficient way to create a representation of a planning problem, that can be used to
  - Achieve better heuristic estimates
  - Directly construct plans
- Planning graphs only work for propositional problems.

Planning graphs

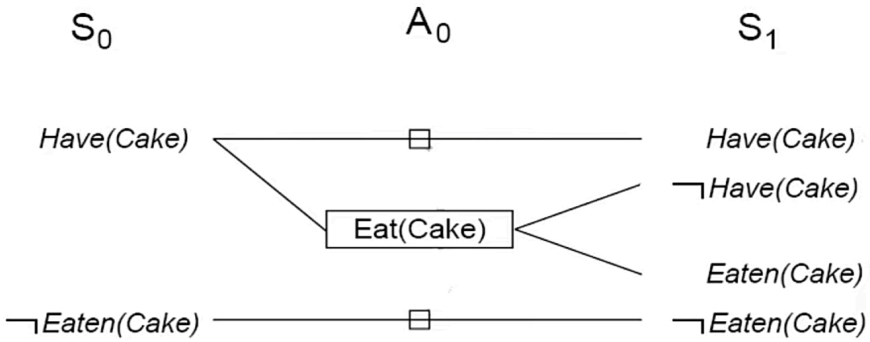
- It consists of a seq of levels that correspond to time steps in the plan.
  - Level 0 is the initial state.
  - Each level consists of a set of literals and a set of actions that represent, what might be possible at that step in the plan
  - Records only a restricted subset of possible negative interactions among actions.

Planning Graph

- Each level consists of
  - Literals = all those that could be true at that time step, depending upon the actions executed at preceding time steps.
  - Actions = all those actions have their preconditions, that satisfied at that time step, depending on which of the literals actually hold.

Example - The “have cake and eat cake too” problem.

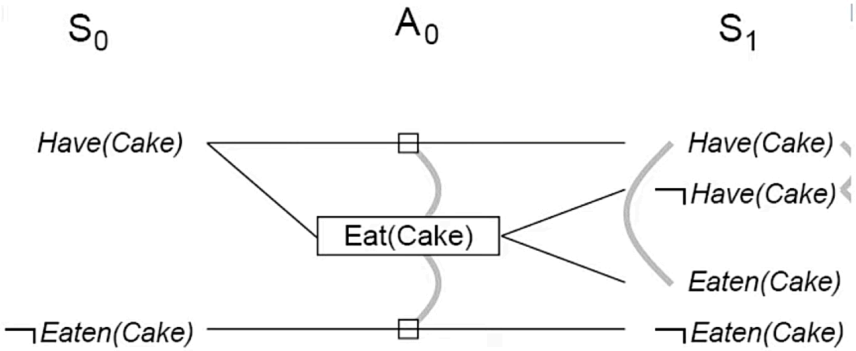
$$
\begin{aligned}
&Init(Have(Cake)) \\
&Goal(Have(Cake) \ \wedge \ Eaten(Cake)) \\
&Action(Eat(Cake) \\
&\quad \text{PRECOND: } Have(Cake) \\
&\quad \text{EFFECT: } \neg \, Have(Cake) \ \wedge \ Eaten(Cake)) \\
&Action(Bake(Cake) \\
&\quad \text{PRECOND: } \neg \, Have(Cake) \\
&\quad \text{EFFECT: } Have(Cake)
\end{aligned}
$$



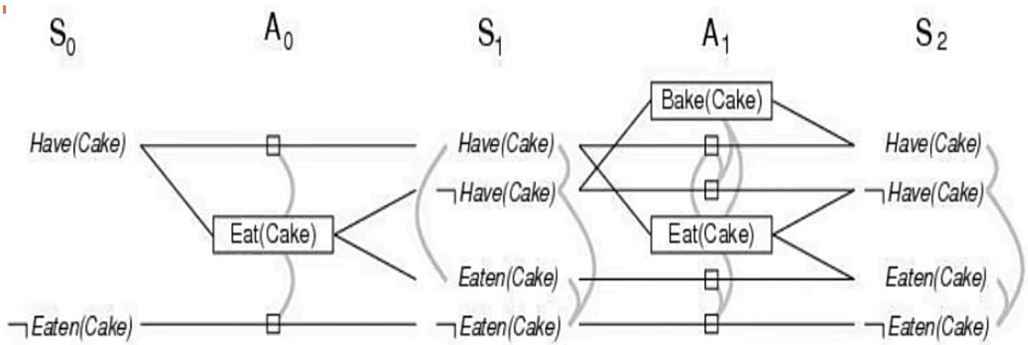
Add *persistence actions* (inaction = no-ops) to map all literals in state S_i to state S_{i+1}.

Mutual exclusion

- A mutex relation holds between two actions when:
  - Inconsistent effects: one action negates the effect of another.
  - Interference: one of the effects of one action, is the negation of a precondition of the other.
  - Competing needs: one of the preconditions of one action, is mutually exclusive with the precondition of the other.
- A mutex relation holds between two literals when:
  - one is the negation of the other
  - each possible action pair that could achieve the literals is mutex (inconsistent support).



- Level S1 contains all literals, that could result from, picking any subset of actions in A0
  - Conflicts between literals that can not occur together (as a consequence of the selection action) are represented by mutex links.
  - S1 defines multiple states, and the mutex links are the constraints, that define this set of states.



- Repeat process until graph levels off:
  - two consecutive levels are identical, or
  - contain the same amount of literals

The GRAPHPLAN Algorithm

```
function GRAPHPLAN(problem) returns solution or failure

    graph ← INITIAL-PLANNING-GRAPH(problem)
    goals ← GOALS[problem]
    loop do
        if goals all non-mutex in last level of graph then do
            solution ← EXTRACT-SOLUTION(graph, goals, LENGTH(graph))
            if solution ≠ failure then return solution
            else if NO-SOLUTION-POSSIBLE(graph) then return failure
        graph ← EXPAND-GRAPH(graph, problem)
```

Figure 11.13 The GRAPHPLAN algorithm. GRAPHPLAN alternates between a solution extraction step and a graph expansion step. EXTRACT-SOLUTION looks for whether a plan can be found, starting at the end and searching backwards. EXPAND-GRAPH adds the actions for the current level and the state literals for the next level.

Satisfiability Problem (SAT) Prooblem

- **Boolean Satisfiability Problem**
  - Boolean Satisfiability or simply **SAT** is the problem of determining if a Boolean formula is satisfiable or unsatisfiable.
- **Satisfiable :**
  - If the Boolean variables can be assigned values such that the formula turns out to be TRUE, then we say that the formula is satisfiable.
- **Unsatisfiable :**
  - If it is not possible to assign such values, then we say that the formula is unsatisfiable.

- 2-SAT is a special case of Boolean Satisfiability Problem and can be solved in polynomial time.
- 2-SAT limits the problem of SAT to only those Boolean formula which are expressed as a CNF with every clause having only **2 terms.**

Example:

$$F = (A_1 \vee B_1) \wedge (A_2 \vee B_2) \wedge (A_3 \vee B_3) \wedge ..... \wedge (A_m \vee B_m)$$

Thus, Problem of 2-Satisfiabilty can be stated as:

Given CNF with each clause having only 2 terms, is it possible to assign such values to the variables so that the CNF is TRUE?

- Cook and Levin independently proved that SAT is NP-complete.
- A boolean formula is in 3-conjunctive normal form, or 3-CNF, if each clause has exactly three distinct literals.
- (x1 ^¬x1^ ¬x2)V (x3 ^ x2 ^ x4)V (¬x1 ^ ¬x3 ^ ¬x4)
- is in 3-CNF. The first of its three clauses is (x1 ^ ¬x1 ^ ¬x2), which contains the three literals x1, ¬x1, and ¬x2.
- In 3-CNF-SAT, we are asked whether a given boolean formula φ in 3-CNF is satisfiable.

Temporal Models

- Agents in uncertain environments must be able to keep track of the current state of the environment, just as logical agents must.
- This is difficult by partial and noisy data, because the environment is uncertain over time.
- At best, the agent will be able to obtain only a probabilistic assessment of the current situation.

Temporal Models

- Two sections in Temporal Model,
  - Time and Uncertainty
    - **States and observations**
    - **Stationary processes and the Markov assumption**
  - Inference in Temporal Model

Temporal Models - Time and Uncertainty

- A changing world is modeled using a random variable for each aspect of the environment state, *at each point in time.*
- The relations among these variables describe how the state evolves.

Example - Treating a Diabetic Patient.

- We have evidence, such as, recent insulin doses, food intake, blood sugar measurements, and other physical signs.
- The task is to assess the current state of the patient, including the actual blood sugar level and insulin level.
- Given this information, the doctor (or patient) makes a decision about the patient's food intake and insulin dose.

Example - Treating a Diabetic Patient...

- The dynamic aspects of the problem are essential.
- Blood sugar levels and measurements thereof can change rapidly over time, depending on one's recent food intake and insulin doses, one's metabolic activity, the time of day, and so on.
- To assess the current state from the history of evidence and to predict the outcomes of treatment actions, we must model these changes.

- Two sections in Temporal Model,
  - Time and Uncertainty
    - **States and observations**
    - **Stationary processes and the Markov assumption**
  - Inference in Temporal Model

States and Observations

- The process of change can be viewed as a series of snapshots (results), describes the state of the world at a particular time.
- Each snapshot or time slice, contains a set of random variables, some of which are observable and some of which are not.

State and observation ...

- We will assume that the same subset of variables is observable in each slice
- X_t – set of unobservable state variable at time t
- E_t – set of observable evidence variable
- The observation at time t is $E_t = e_t$ for some set of values e_t

Inductive Learning Algorithm

In this article, we will learn about Inductive Learning Algorithm which generally comes under the domain of Machine Learning.

What is Inductive Learning Algorithm?

Inductive Learning Algorithm (ILA) is an iterative and inductive machine learning algorithm that is used for generating a set of classification rules, which produces rules of the form "IF-THEN", for a set of examples, producing rules at each iteration and appending to the set of rules.

There are basically two methods for knowledge extraction firstly from domain experts and then with machine learning. For a very large amount of data, the domain experts are not very useful and reliable. So we move towards the machine learning approach for this work. To use machine learning One method is to replicate the expert's logic in the form of algorithms but this work is very tedious, time taking, and expensive. So we move towards the inductive algorithms which generate the strategy for performing a task and need not instruct separately at each step.

Why you should use Inductive Learning?

The ILA is a new algorithm that was needed even when other reinforcement learnings like ID3 and AQ were available.

- The need was due to the pitfalls which were present in the previous algorithms, one of the major pitfalls was the lack of generalization of rules.
- The ID3 and AQ used the decision tree production method which was too specific which were difficult to analyze and very slow to perform for basic short classification problems.
- The decision tree-based algorithm was unable to work for a new problem if some attributes are missing.
- The ILA uses the method of production of a general set of rules instead of decision trees, which overcomes the above problems

Basic Requirements to Apply Inductive Learning Algorithm

1. List the examples in the form of a table 'T' where each row corresponds to an example and each column contains an attribute value.
2. Create a set of m training examples, each example composed of k attributes and a class attribute with n possible decisions.
3. Create a rule set, R, having the initial value false.
4. Initially, all rows in the table are unmarked.

Necessary Steps for Implementation

- **Step 1:** divide the table 'T' containing m examples into n sub-tables (t1, t2,.....tn). One table for each possible value of the class attribute. (repeat steps 2-8 for each sub-table)
- **Step 2:** Initialize the attribute combination count ' j ' = 1.
- **Step 3:** For the sub-table on which work is going on, divide the attribute list into distinct combinations, each combination with 'j ' distinct attributes.
- **Step 4:** For each combination of attributes, count the number of occurrences of attribute values that appear under the same combination of attributes in unmarked rows of the sub-table under consideration, and at the same time, not appears under the same combination of attributes of other sub-tables. Call the first combination with the maximum number of occurrences the max-combination ' MAX'.
- **Step 5:** If 'MAX' == null, increase ' j ' by 1 and go to Step 3.
- **Step 6:** Mark all rows of the sub-table where working, in which the values of 'MAX' appear, as classified.
- **Step 7:** Add a rule (IF attribute = "XYZ" –> THEN decision is YES/ NO) to R whose left-hand side will have attribute names of the 'MAX' with their values separated by AND, and its right-hand side contains the decision attribute value associated with the sub-table.
- **Step 8:** If all rows are marked as classified, then move on to process another sub-table and go to Step 2. Else, go to Step 4. If no sub-tables are available, exit with the set of rules obtained till then.

An example showing the use of ILA suppose an example set having attributes Place type, weather, location, decision, and seven examples, our task is to generate a set of rules that under what condition is the decision.

Example no.	Place type	weather	location	decision

- **Step 5:** If 'MAX' == null, increase 'j' by 1 and go to Step 3.
- **Step 6:** Mark all rows of the sub-table where working, in which the values of 'MAX' appear, as classified.
- **Step 7:** Add a rule (IF attribute = "XYZ" –> THEN decision is YES/ NO) to R whose left-hand side will have attribute names of the 'MAX' with their values separated by AND, and its right-hand side contains the decision attribute value associated with the sub-table.
- **Step 8:** If all rows are marked as classified, then move on to process another sub-table and go to Step 2. Else, go to Step 4. If no sub-tables are available, exit with the set of rules obtained till then.

An example showing the use of ILA suppose an example set having attributes Place type, weather, location, decision, and seven examples, our task is to generate a set of rules that under what condition is the decision.

Example no.	Place type	weather	location	decision
1.	hilly	winter	kullu	Yes
2.	mountain	windy	Mumbai	No
3.	mountain	windy	Shimla	Yes
4.	beach	windy	Mumbai	No
5.	beach	warm	goa	Yes
6.	beach	windy	goa	No
7.	beach	warm	Shimla	Yes

Subset – 1

s.no	place type	weather	location	decision
1.	hilly	winter	kullu	Yes
2.	mountain	windy	Shimla	Yes
3.	beach	warm	goa	Yes
4.	beach	warm	Shimla	Yes

Subset – 2

s.no	place type	weather	location	decision
5.	mountain	windy	Mumbai	No
6.	beach	windy	Mumbai	No
7.	beach	windy	goa	No

- **At iteration 1** rows 3 & 4 column weather is selected and rows 3 & 4 are marked. the rule is added to R IF the weather is warm then a decision is yes.
- **At iteration 2** row 1 column place type is selected and row 1 is marked. the rule is added to R IF the place type is hilly then the decision is yes.
- **At iteration 3** row 2 column location is selected and row 2 is marked. the rule is added to R IF the location is Shimla then the decision is yes.
- **At iteration 4** row 5&6 column location is selected and row 5&6 are marked. the rule is added to R IF the location is Mumbai then a decision is no.
- **At iteration 5** row 7 column place type & the weather is selected and row 7 is marked. the rule is added to R IF the place type is beach AND the weather is windy then the decision is no.

Finally, we get the rule set:- Rule Set

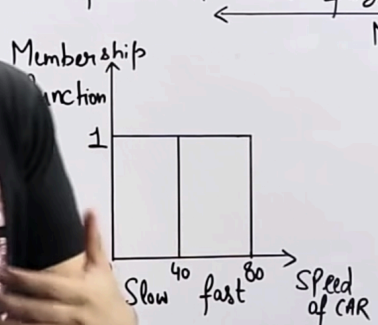
- **Rule 1:** IF the weather is warm THEN the decision is yes.
- **Rule 2:** IF the place type is hilly THEN the decision is yes.
- **Rule 3:** IF the location is Shimla THEN the decision is yes.
- **Rule 4:** IF the location is Mumbai THEN the decision is no.
- **Rule 5:** IF the place type is beach AND the weather is windy THEN the decision is no.

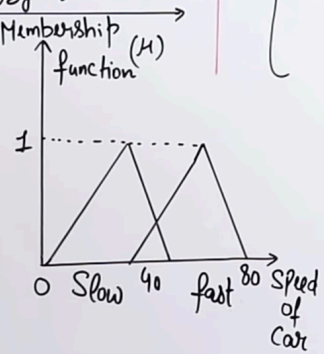
'Fuzzy Logic' (Lotfi Zadeh)

→ Represent uncertainty [0,1]

→ Represent with degree

→ Represent the belongingness of a member of a crisp set to fuzzy set.





'Check the degree of fastness'

$$\begin{cases} 0, & \text{if } Speed(x) \leq 40 \\ \dfrac{Speed(x) - 40}{10}, & \text{if } 40 < Speed(x) < 50 \\ 1, & \text{if } Speed(x) \geq 50 \end{cases}$$

'Operations in fuzzy logic'

→ Union: $\text{Max}\{\mu_A(x), \mu_B(x)\}, x \in U$

→ Intersection: $\text{Min}\{\mu_A(x), \mu_B(x)\}, x \in U$

→ Complement: $\mu_{\bar{A}}(x) = [1 - \mu_A(x)], x \in U$

→ Bold Union: $\mu_{A \oplus B} = \min[1, \mu_A(x) + \mu_B(x)]$

→ Bold Intersection: $\mu_{A \odot B}(x) = \text{Max}[0, \mu_A(x) + \mu_B(x) - 1]$

→ Equality $A = B$ if $\mu_A(x) = \mu_B(x) \; \forall x \in S$

$U = \{5, 10, 20, 25, 30, 40\}$

$A = \{(10, 0.2), (20, 0.4), (25, 0.7), (30, 0.9), (40, 1)\}$

$B = \{(10, 0.4), (20, 0.1), (25, 0.9), (30, 0.2), (40, 0.6)\}$

'Operations in fuzzy logic'

$\frac{x}{x+2}$ $\frac{x}{x+5}$ $U = \{5, 10, 20, 25, 30, 40\}$

→ Union: $\text{Max}\{\mu_A(x), \mu_B(x)\}, x \in U$

→ Intersection: (OR) $\text{Min}\{\mu_A(x), \mu_B(x)\}, x \in U$

→ Complement: (AND) $\mu_{\bar{A}}(x) = [1 - \mu_A(x)], x \in U$

→ Bold Union: (NOT) $\mu_{A \oplus B} = \min[1, \mu_A(x) + \mu_B(x)]$

→ Bold Intersection: $\mu_{A \odot B}(x) = \text{Max}[0, \mu_A(x) + \mu_B(x) - 1]$

→ Equality $A = B$ if $\mu_A(x) = \mu_B(x) \; \forall x \in S$

$A = \{(10, 0.2), (20, 0.4), (25, 0.7), (30, 0.9), (40, 1)\}$ (x, μ)

$B = \{(10, 0.4), (20, 0.1), (25, 0.9), (30, 0.2), (40, 0.6)\}$

10, 0.6 20, 0.5

(1, 0.6)

(10, 0.6), (20, 0.5)

25 1.6

in union if anyone is not given that is taken but is intersection it does not take

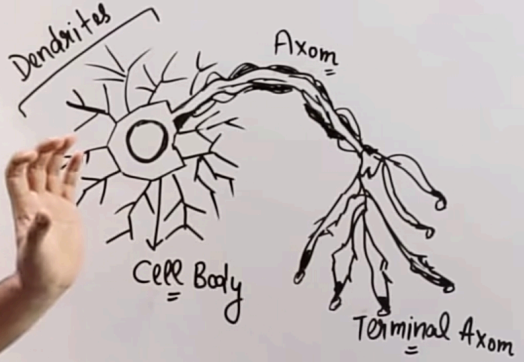
'Neural Networks' → Artificial Neural Networks

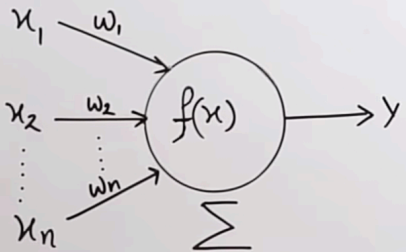
Brain → Machines

Input Data → Meaning → Learning and Improvement

Algorithms

Neural Networks





Natural language Processing (NLP)

→ How Human Communicate with each other
→ Computer should replicate the same thing
→ Applications of NLP

* Speech Recognition
* Sentimental Analysis
* Machine Translations
* Chat bots etc.

NLU ⇒ What do the users say? their intent? Meaning?

Challenges:
- Lexical Ambiguity
- Syntactic Ambiguity
- Semantic ,,
- Pragmetic ,,

→ The tank was full of water.
→ Old men and women were taken to safe place.
→ The car hit the pole while it was moving.
→ The police are coming.

NLG → What should we say to User?
→ it should be Intelligent and Conversational.
→ Deal with structured data.
→ Text / Sentence Planning

Below is the implementation of the above idea:

C++ | Java | Python3 | C#

```python
from queue import PriorityQueue
v = 14
graph = [[] for i in range(v)]

# Function For Implementing Best First Search
# Gives output path having lowest cost

def best_first_search(actual_Src, target, n):
    visited = [False] * n
    pq = PriorityQueue()
    pq.put((0, actual_Src))
    visited[actual_Src] = True

    while pq.empty() == False:
        u = pq.get()[1]
        # Displaying the path having lowest cost
        print(u, end=" ")
        if u == target:
            break

        for v, c in graph[u]:
            if visited[v] == False:
                visited[v] = True
                pq.put((c, v))
    print()

# Function for adding edges to graph

def addedge(x, y, cost):
    graph[x].append((y, cost))
    graph[y].append((x, cost))

# The nodes shown in above example(by alphabets) are
# implemented using integers addedge(x,y,cost);
addedge(0, 1, 3)
addedge(0, 2, 6)
addedge(0, 3, 5)
addedge(1, 4, 9)
addedge(1, 5, 8)
addedge(2, 6, 12)
addedge(2, 7, 14)
addedge(3, 8, 7)
addedge(8, 9, 5)
addedge(8, 10, 6)
addedge(9, 11, 1)
addedge(9, 12, 10)
addedge(9, 13, 2)

source = 0
target = 9
best_first_search(source, target, v)

# This code is contributed by Jyotheeswar Ganne
```

Output

```
0 1 3 2 8 9
```

Analysis :

- The worst-case time complexity for Best First Search is O(n * log n

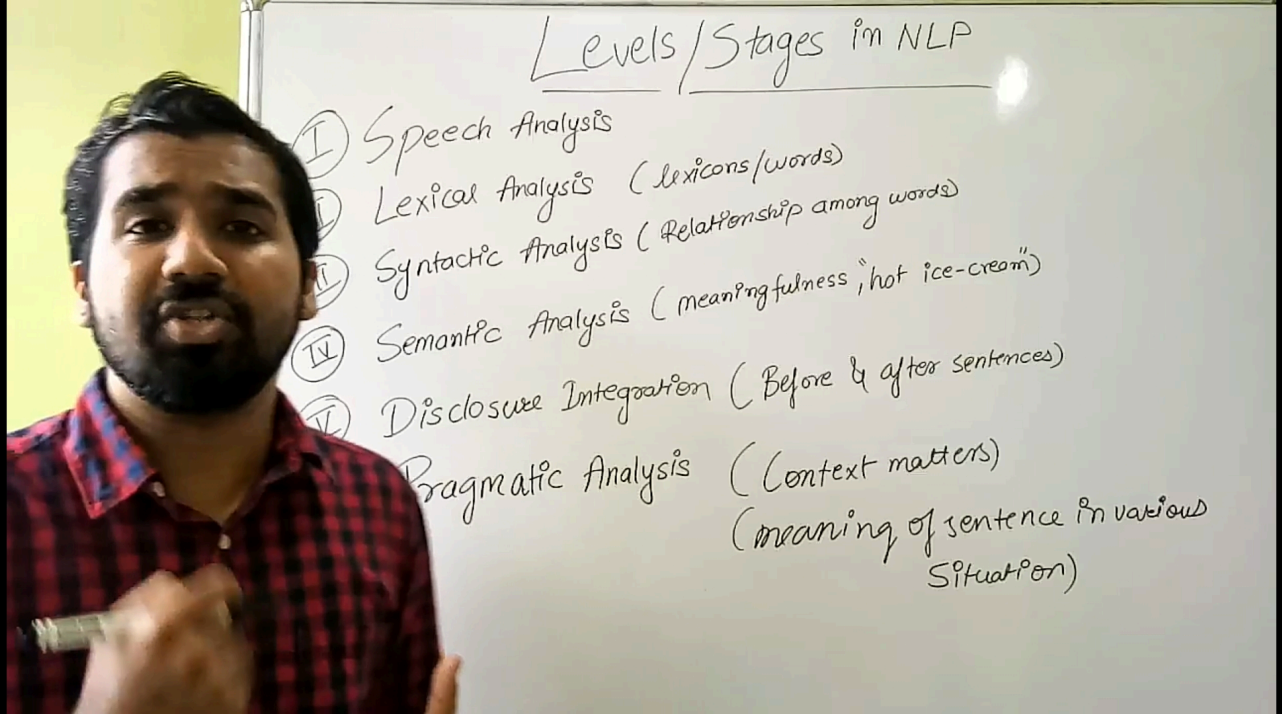
Components of NLP

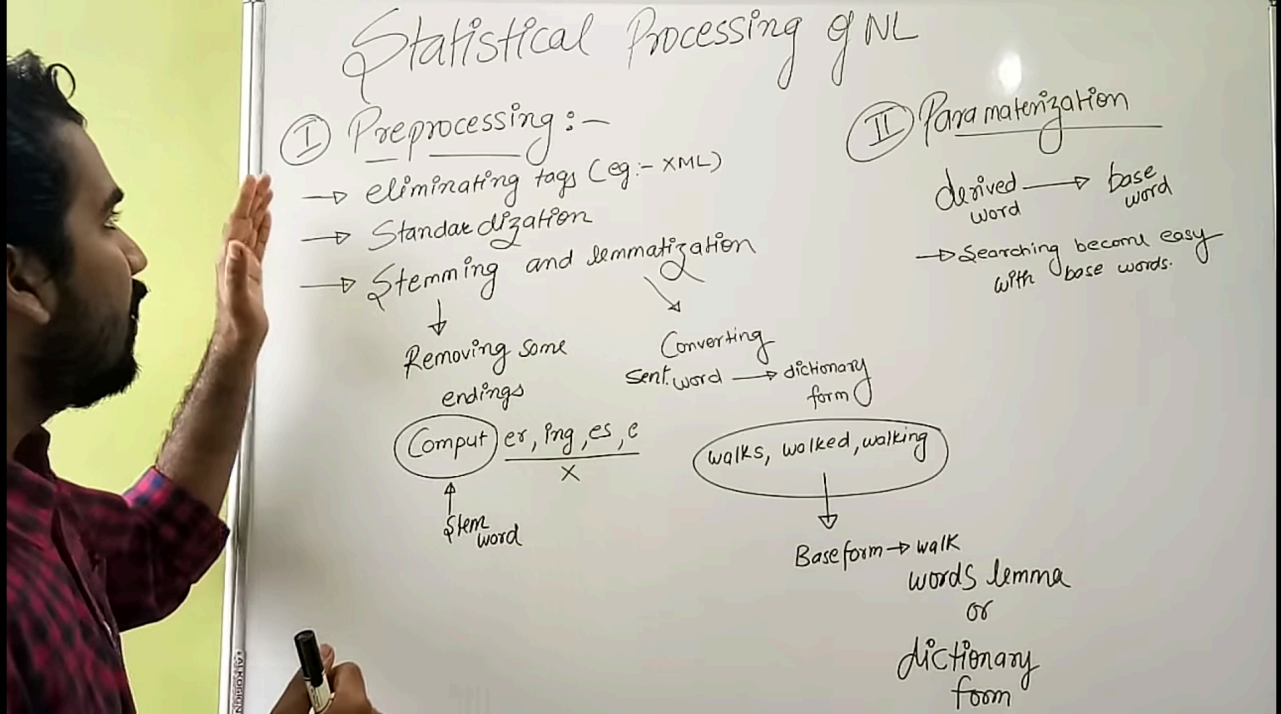
(I) Natural Language Understanding:-

- Lexical Ambiguity (word level)
- Syntactical Ambiguity (Sentence level)
- Referential Ambiguity (Referring issue)

(II) Natual Language Generation:-

- Text planning (Retrive Relevant Content from KB)
- Sentence planning (words, meaning full phrases, setting tone)
- Text Realization (Mapping Sentence planning into sentence structure)


Levels / Stages in NLP
I Speech Analysis
II Lexical Analysis (lexicons/words)
III Syntactic Analysis (Relationship among words)
IV Semantic Analysis (meaningfulness "hot ice-cream")
V Disclosure Integration (Before & after sentences)
VI Pragmatic Analysis (Context matters)
(meaning of sentence in various Situation)


Statistical Processing of NL
(I) Preprocessing :-
→ eliminating tags (eg:- XML)
→ Standardization
→ Stemming and lemmatization
Removing some endings
Comput er, ing, es, e
×
Stem word
Converting sent. word → dictionary form
walks, walked, walking
Base form → walk
words lemma or dictionary form
(II) Parameterization
derived word → base word
→ Searching become easy with base words.

Statistical Learning Methods
(Bayesian Learning & EM Algorithm)

Bayesian Learning

- Basics of Bayesian Network has already been introduced earlier.
- In Bayesian learning, each training sample can increase or decrease the estimated probability supporting the hypothesis.
- Prior knowledge is combined with the observed data to determine the probability of hypothesis from the space, say H, and this is done using unobserved training data D.
- Popularly, this is done by finding out the most probable hypothesis with reference to the observed data and knowledge about initial probabilities.
- Conditional probability helps us in calculating the probabilities to determine the best hypothesis.

Note: Naive-Bayes-Classifiers would be discussed in Unit 5.

Learning with Hidden Variables – The EM Algorithm

- Real world problems contain hidden variables. They are also called **latent variables**.

- In many practical problems, we treat symptoms rather than actually observing the event. This is very common in many applications like medical treatments. The hidden variables reduce the number of parameters one needs to specify using Bayesian.

- Expectation maximization (EM) algorithm simplifies difficult maximum likelihood problems. The EM algorithm can be used even for the variables whose value is not observable.
- Each iteration of the EM algorithm consists of two processes – the E-step, and the M-step.
- In the expectation, or E-step, the intention is to determine the missing values. The missing data is estimated based on the observed data. This is done given the observed data and the current estimate of the model parameters. For this expectation the conditional expectation explaining the choice of terminology is used.
- In M-step, the likelihood function is maximized under the assumption that the missing data is known. In place of missing data, the estimated data in E-step is used. Convergence is assured, since the algorithm is guaranteed to increase the likelihood during each iteration.

Statistical Learning Methods

- Statistical Learning based on the Learning of uncertainty in real environments.
- The methods probability and decision theory are used to handle uncertainty by the Agents
- First the agent must learn its probabilistic theories of the world from **experience**.
- A Bayesian view of learning is extremely powerful, providing general solutions to the problems of noise, overfitting, and optimal prediction.

Artificial Intelligence Knowledge in Learning:

Slides:

A LOGICAL FORMULATION OF LEARNING

- The hypothesis is represented by a set of logical sentences.
- Example descriptions and classifications will also be logical sentences,
- A new example can be **classified by inferring** a classification sentence from the hypothesis and the example description.

5

Artificial Intelligence Knowledge in Learning:

Slides:

A LOGICAL FORMULATION OF LEARNING

- Hypothesis space is the set of all hypotheses the learning algorithm is designed to entertain.
- One of the hypotheses is correct:
  $H_1 \ V \ H_2 \ V \ldots V \ H_n$
- Each H_i predicts a certain set of examples: the ***extension*** of the goal predicate.
- Two hypotheses with different extensions are *logically inconsistent* with each other, otherwise, they are *logically equivalent*.

7

Statistical Learning Methods

- Statistical Learning based on the Learning of uncertainty in real environments.
- The methods probability and decision theory are used to handle uncertainty by the Agents
- First the agent must learn its probabilistic theories of the world from **experience**.
- A Bayesian view of learning is extremely powerful, providing general solutions to the problems of noise, overfitting, and optimal prediction.

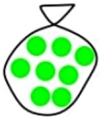
Statistical Learning Methods...

- Statistical Learning is about inferences
- The idea is generated from the **Data and Hypothesis** and these are called as key terms of statistical learning.
- Data (Samples and Population) are **Evidence**

Surprise Candy

- Let us consider a very simple example.
- Our favorite Surprise candy comes in two flavors:
- Cherry (yum) and 
- Lime (ugh). 
- The candy manufacturer has a peculiar sense of humor and wraps each piece of candy in the same opaque wrapper, regardless of flavor.

- The candy is sold in very large bags, of which there are known to be five kinds—again, cannot identify from the outside:

1. h1: 100% cherry 
2. h2: 75% cherry + 25% lime 
3. h3: 50% cherry + 50% lime 
4. h4: 25% cherry + 75% lime 
5. h5: 100% lime 

- Suppose there are five kinds of bags of candies:
- 10% are h1: 100% cherry candies 
- 20% are h2: 75% cherry candies + 25% lime candies 
- 40% are h3: 50% cherry candies + 50% lime candies 
- 20% are h4: 25% cherry candies + 75% lime candies 
- 10% are h5: 100% lime candies 
- Then we observe candies drawn from some bag:
- What kind of bag is it? What flavor will the next candy be?

Surprise Candy…

- Given a new bag of candy, the random variable H (for *hypothesis*) denotes the type of the bag, with possible values h1 through h5.
- H is not directly observable, of course.
- As the pieces of candy are opened and inspected,
- data are revealed—$D_1, D_2, \ldots, D_N$,
- where each D_i is a random variable with possible values cherry and lime.
- The basic task faced by the agent is to **predict the flavor of the next piece of candy.**
- The agent need to learn a theory of its world,

- **Bayesian learning** simply calculates the probability of each hypothesis, given the data, and makes predictions on that basis.
- Let **D** represent all the data, with observed value **d**; then the probability of each hypothesis is obtained by Bayes' rule:
- $P(h_i/\mathbf{d}) = \alpha P(\mathbf{d}/h_i)P(h_i)$:

- suppose we want to make a prediction about an unknown quantity X

$$\mathbf{P}(X|\mathbf{d}) = \sum_i \mathbf{P}(X|\mathbf{d}, h_i)\mathbf{P}(h_i|\mathbf{d}) = \sum_i \mathbf{P}(X|h_i)P(h_i|\mathbf{d})$$

- where each hypothesis determines a probability distribution over X.
- This equation shows that predictions are weighted averages over the predictions of the individual hypotheses.
- The key quantities in the Bayesian approach are the **prior hypothesis**, $P(h_i)$, and the **likelihood** of the data under each hypothesis, $P(\mathbf{d}/h_i)$.

- Our candy example, we will assume for the time being that the prior distribution over
- h1, h2, h3, h4, h5 is given by
- <0.1 0.2 0.4 0.2 0.1>

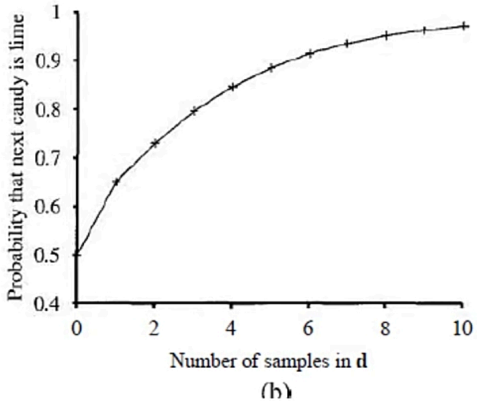
- as advertised by the manufacturer.
- The likelihood of the data ($P(d/h_i)$) is calculated under the assumption that the observations are **i.i.d.**—that is, **independently and identically distributed**—so that

$$P(\mathbf{d}|h_i) = \prod_j P(d_j|h_i)$$

- The predicted probability that the next candy is lime, based on Equation with respect to h5

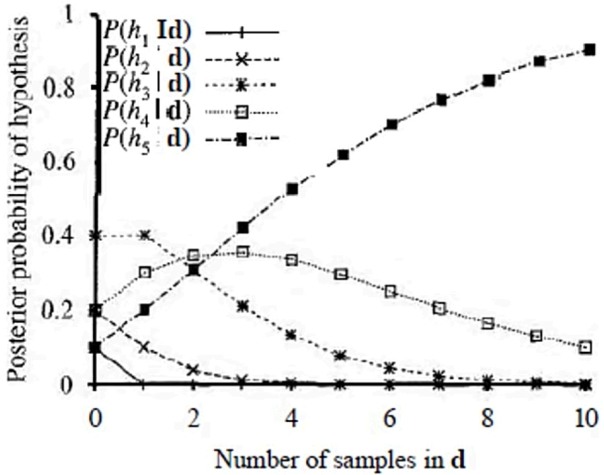
$$\mathbf{P}(X|\mathbf{d}) = \sum_i \mathbf{P}(X|\mathbf{d}, h_i)\mathbf{P}(h_i|\mathbf{d}) = \sum_i \mathbf{P}(X|h_i)P(h_i|\mathbf{d})$$

- As we would expect, it increases monotonically toward 1.



- the posterior probabilities of the five hypotheses change as the sequence of *10* lime candies is observed.

- H1 
- H2 
- H3 
- H4 
- h5 



Step 4: Choose feature for each node to split on!

"Sunny node":

IG(Y, weather) = IG(humidity) = 0.9183

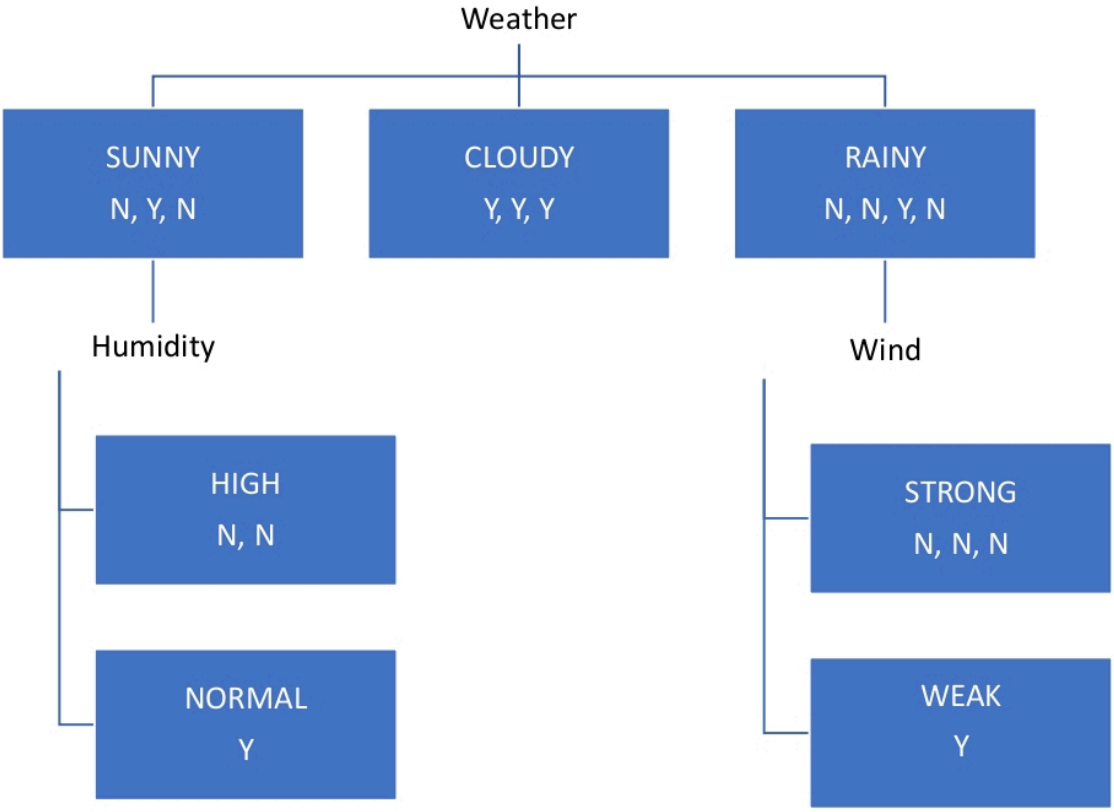
IG(Y, wind) = 0.2516

"Rainy node":

IG(Y, weather) = IG(Y, humidity) = 0.1226

IG(Y, wind) = 0.8113

Final Tree!



Solution of Ai big question