UNIT 1

What is Machine Learning?

Machine learning is programming computers to optimize a performance criterion using example data or past experience. We have a model defined up to some parameters, and learning is the execution of a computer program to optimize the parameters of the model using the training data or past experience. The model may be predictive to make predictions in the future, or descriptive to gain knowledge from data, or both.

Definition of learning A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks T, as measured by P, improves with experience E.

Example- Handwriting recognition learning problem

- Task T: Recognising and classifying handwritten words within images
- Performance P: Percent of words correctly classified
- Training experience E: A dataset of handwritten words with given classifications.

Ensemble learning is the process by which multiple models, such as classifiers or experts, are strategically generated and combined to solve a particular computational intelligence problem. Ensemble learning is primarily used to improve the (classification, prediction, function approximation, etc.) performance of a model, or reduce the likelihood of an unfortunate selection of a poor one.

Design of a learning system

The design choices will be to decide the following key components:

1. Type of training experience
2. Choosing the Target Function
3. Choosing a representation for the Target Function
4. Choosing an approximation algorithm for the Target Function
5. The final Design

We will look into the game - checkers learning problem and apply the above design choices. For a checkers learning problem, the three elements will be,

1. Task T: To play checkers

2. Performance measure P: Total percent of the game won in the tournament.
3. Training experience E: A set of games played against itself

1 .Type of training experience: During the design of the checker's learning system, the type of training experience available for a learning system will have a significant effect on the success or failure of the learning.

Direct or Indirect training experience — In the case of direct training experience, an individual board states and correct move for each board state are given. In case of indirect training experience, the move sequences for a game and the final result (win, loss or draw) are given for a number of games.
Teacher or Not — **Supervised** — The training experience will be labeled, which means, all the board states will be labeled with the correct move. So the learning takes place in the presence of a supervisor or a teacher. **Unsupervised** — the training experience will be unlabeled, which means, all the board states will not have the moves. So the learner generates random games and plays against itself with no supervision or teacher involvement.
— Learner generates game states and asks the teacher for help in finding the correct move if the board state is confusing
Is the training experience good — Do the training examples represent the distribution of examples over which the final system performance will be measured? Performance is best when training examples and test examples are from the same/a similar distribution.

2. Choosing the Target Function: When you are playing the checkers game, at any moment of time, you make a decision on choosing the best move from different possibilities. You think and apply the learning that you have gained from the experience. Here the learning is, for a specific board, you move a checker such that your board state tends towards the winning situation. Now the same learning has to be defined in terms of the target function.

Let us therefore define the target value V(b) for an arbitrary board state b in B, as follows:
1. if b is a final board state that is won, then V(b) = 100
2. if b is a final board state that is lost, then V(b) = -100
3. if b is a final board state that is drawn, then V(b) = 0
4. if b is a not a final state in the game, then V (b) = V (b'), where b' is the best final board state that can be achieved starting from b and playing optimally until the end of the game.

3. Choosing a representation for the Target Function: Now that we have specified the ideal target function V, we must choose a representation that the learning program will use to describe the function ^V that it will learn.
let us choose a simple representation: for any given board state, the function ^V will be calculated as a linear combination of the following board features:

- x1(b) — number of black pieces on board b
- x2(b) — number of red pieces on b
- x3(b) — number of black kings on b
- x4(b) — number of red kings on b
- x5(b) — number of red pieces threatened by black (i.e., which can be taken on black's next turn)
- x6(b) — number of black pieces threatened by red

^V = w0 + w1 · x1(b) + w2 · x2(b) + w3 · x3(b) + w4 · x4(b) +w5 · x5(b) + w6 · x6(b) Where w0 through w6 are numerical coefficients or weights to be obtained by a learning algorithm. Weights w1 to w6 will determine the relative importance of different board features.

4.Choosing an approximation algorithm for the Target Function

Generating training data — To train our learning program, we need a set of training data, each describing a specific board state b and the training value V_train (b) for b. Each training example is an ordered pair

For example, a training example may be <(x1 = 3, x2 = 0, x3 = 1, x4 = 0, x5 = 0, x6 = 0), +100>. This is an example where black has won the game since x2 = 0 or red has no remaining pieces. However, such clean values of V_train (b) can be obtained only for board value b that are clear win, lose or draw.

In above case, assigning a training value V_train(b) for the specific boards b that are clean win, lose or

draw is direct as they are direct training experience. But in the case of indirect training experience,

Assigning a training value V_train(b) for the intermediate boards is difficult. In such case, the training

values are updated using temporal difference learning.

Let Successor(b) denotes the next board state following b for which it is again the program's turn to

move. ^V is the learner's current approximation to V. Using these information, assign the training value

of V_train(b) for any intermediate board state b as below :

$$V\_train(b) \leftarrow \hat{V}(Successor(b))$$

Adjusting theweights -Now it's time to define the learning algorithm for choosing the weights and best fit the set of training examples. One common approach is to define the best hypothesis as that which minimizes the squared error E between the training values and the values predicted by the hypothesis ^V. The learning algorithm should incrementally refine weights as more training examples become available and it needs to be robust to errors in training data Least Mean Square

(LMS) training rule is the one training algorithm that will adjust weights a small amount in the direction that reduces the error.

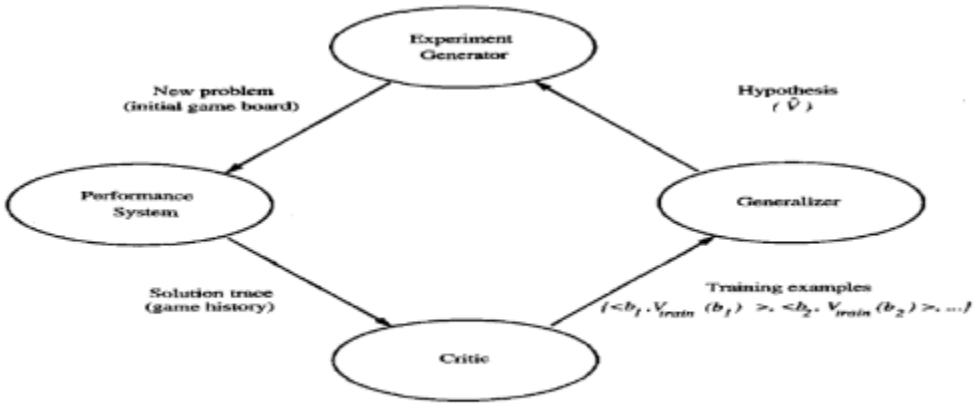
5. Final Design for Checkers Learning system -The final design of our checkers learning system can be naturally described by four distinct program modules that represent the central components in many learning systems.

1. **The performance System** — Takes a new board as input and outputs a trace of the game it played against itself.

2**. The Critic** — Takes the trace of a game as an input and outputs a set of training examples of the target function.

3**. The Generalizer** — Takes training examples as input and outputs a hypothesis that estimates the target function. Good generalization to new cases is crucial.

4. **The Experiment Generator** — Takes the current hypothesis (currently learned function) as input and outputs a new problem (an initial board state) for the performance system to explore.



Final design of the checkers learning program.

Perspectives and Issues in Machine Learning

Following are the list of issues in machine learning:

1. What algorithms exist for learning general target functions from specific training examples? In what settings will particular algorithms converge to the desired function, given sufficient training data? Which algorithms perform best for which types of problems and representations?

2. How much training data is sufficient? What general bounds can be found to relate the confidence in learned hypotheses to the amount of training experience and the character of the learner's hypothesis space?

3. When and how can prior knowledge held by the learner guide the process of generalizing from examples? Can prior knowledge be helpful even when it is only approximately correct?

4. What is the best strategy for choosing a useful next training experience, and how does the choice of this strategy alter the complexity of the learning problem?

5. What is the best way to reduce the learning task to one or more function approximation problems? Put another way, what specific functions should the system attempt to learn? Can this process itself be automated?

6. How can the learner automatically alter its representation to improve its ability to represent and learn the target function?

Find S Algorithm

The find-S algorithm is a basic concept learning algorithm in machine learning. The find-S algorithm finds the most specific hypothesis that fits all the positive examples. We have to note here that the algorithm considers only those positive training example. The find-S algorithm starts with the most specific hypothesis and generalizes this hypothesis each time it fails to classify an observed positive training data. Hence, the Find-S algorithm moves from the most specific hypothesis to the most general hypothesis.

ImportantRepresentation:

1. **?** indicates that any value is acceptable for the attribute.
2. specify a single required value (e.g., Cold) for the attribute.
3. **ϕ**indicates that no value is acceptable.
4. The most **general hypothesis** is represented by: **{?, ?, ?, ?, ?, ?}**
5. The most **specific hypothesis** is represented by: **{ϕ, ϕ, ϕ, ϕ, ϕ, ϕ}**

Steps Involved In Find-S :

1. Start with the most specific hypothesis.
   h = {ϕ, ϕ, ϕ, ϕ, ϕ, ϕ}
2. Take the next example and if it is negative, then no changes occur to the hypothesis.
3. If the example is positive and we find that our initial hypothesis is too specific then we update our current hypothesis to a general condition.
4. Keep repeating the above steps till all the training examples are complete.

5. After we have completed all the training examples we will have the final hypothesis when can use to classify the new examples.

Algorithm:

1. Initialize h to the most specific hypothesis in H
2. For each positive training instance x

   For each attribute constraint a, in h

   If the constraint a, is satisfied by x

   Then do nothing

   Else replace a, in h by the next more general constraint that is satisfied by x
3. Output hypothesis h

Implementation of Find-S Algorithm

To understand the implementation, let us try to implement it to a smaller data set with a bunch of examples to decide if a person wants to go for a walk.

The concept of this particular problem will be on what days does a person likes to go on walk.

Time	Weather	Temperature	Company	Humidity	Wind	Goes
Morning	Sunny	Warm	Yes	Mild	Strong	Yes
Evening	Rainy	Cold	No	Mild	Normal	No
Morning	Sunny	Moderate	Yes	Normal	Normal	Yes
Evening	Sunny	Cold	Yes	High	Strong	Yes

Looking at the data set, we have six attributes and a final attribute that defines the positive or negative example. In this case, yes is a positive example, which means the person will go for a walk.

So now, the general hypothesis is:

h_0 = {‘Morning’, ‘Sunny’, ‘Warm’, ‘Yes’, ‘Mild’, ‘Strong’}

This is our general hypothesis, and now we will consider each example one by one, but only the positive examples.

h_1= {‘Morning’, ‘Sunny’, ‘?’, ‘Yes’, ‘?’, ‘?’}

h_2 = {‘?’, ‘Sunny’, ‘?’, ‘Yes’, ‘?’, ‘?’}

Limitations of Find-S Algorithm

There are a few limitations of the Find-S algorithm listed down below:

1. There is no way to determine if the hypothesis is consistent throughout the data.
2. Inconsistent training sets can actually mislead the Find-S algorithm, since it ignores the negative examples.
3. Find-S algorithm does not provide a backtracking technique to determine the best possible changes that could be done to improve the resulting hypothesis.

Now that we are aware of the limitations of the Find-S algorithm, let us take a look at a practical implementation of the Find-S Algorithm.

Candidate Elimination Algorithm

The candidate elimination algorithm incrementally builds the version space given a hypothesis space H and a set E of examples. The examples are added one by one; each example possibly shrinks the version space by removing the hypotheses that are inconsistent with the example. The candidate elimination algorithm does this by updating the general and specific boundary for each new example.

- You can consider this as an extended form of Find-S algorithm.
- Consider both positive and negative examples.
- Actually, positive examples are used here as Find-S algorithm (Basically they are generalizing from the specification).
- While the negative example is specified from generalize form.

Terms Used:

- **Concept learning:** Concept learning is basically learning task of the machine (Learn by Train data)
- **General Hypothesis:** Not Specifying features to learn the machine.
- **G = {‘?’, ‘?’,’?’,’?’…}:** Number of attributes
- **Specific Hypothesis:** Specifying features to learn machine (Specific feature)

- **S= {'pi','pi','pi'…}:** Number of pi depends on number of attributes.
- **Version Space:** It is intermediate of general hypothesis and Specific hypothesis. It not only just written one hypothesis but a set of all possible hypothesis based on training data-set.

Algorithm:

Step1: Load Data set

Step2: Initialize General Hypothesis and Specific Hypothesis.

Step3: For each training example

Step4: If example is positive example

```
if attribute_value == hypothesis_value:
   Do nothing
else:
   replace attribute value with '?' (Basically generalizing it)
```

Step5: If example is Negative example

Make generalize hypothesis more specific.

Example:

Consider the dataset given below:

Sky	Temperature	Humid	Wind	Water	Forest	Output
sunny	warm	normal	strong	warm	same	yes
sunny	warm	high	strong	warm	same	yes
rainy	cold	high	strong	warm	change	no
sunny	warm	high	strong	cool	change	yes

Algorithmic steps:

Initially : G =[[?, ?, ?, ?, ?, ?], [?, ?, ?, ?, ?, ?], [?, ?, ?, ?, ?, ?],
[?, ?, ?, ?, ?, ?], [?, ?, ?, ?, ?, ?], [?, ?, ?, ?, ?, ?]]

S = [Null, Null, Null, Null, Null, Null]

For instance 1 : <'sunny','warm','normal','strong','warm ','same'> and positive output.

G1 = G

S1 = ['sunny','warm','normal','strong','warm ','same']

For instance 2 :<'sunny','warm','high','strong','warm ','same'> and positive output.

G2 = G

S2 = ['sunny','warm',?,'strong','warm ','same']

For instance 3 :<'rainy','cold','high','strong','warm ','change'> and negative output.

G3 = [['sunny', ?, ?, ?, ?, ?], [?, 'warm', ?, ?, ?, ?], [?, ?, ?, ?, ?, ?],
[?, ?, ?, ?, ?, ?], [?, ?, ?, ?, ?, ?], [?, ?, ?, ?, ?, 'same']]

S3 = S2

For instance 4 :<'sunny','warm','high','strong','cool','change'> and positive output.

G4 = G3

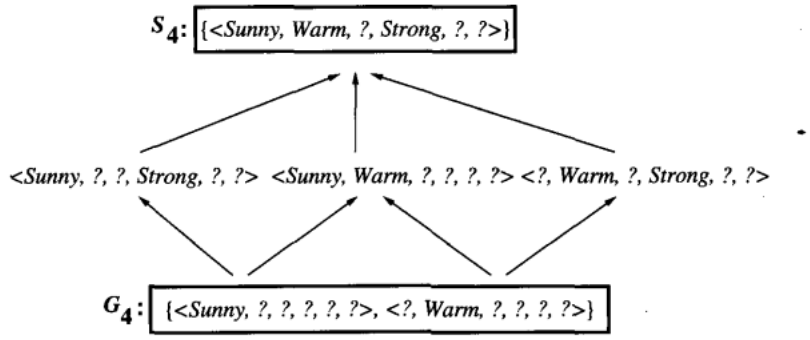
S4 = ['sunny','warm',?,'strong', ?, ?]

At last, by synchronizing the G4 and S4 algorithm produce the output.

Output:

G = [['sunny', ?, ?, ?, ?, ?], [?, 'warm', ?, ?, ?, ?]]

S = ['sunny','warm',?,'strong', ?, ?]



Types of Learning

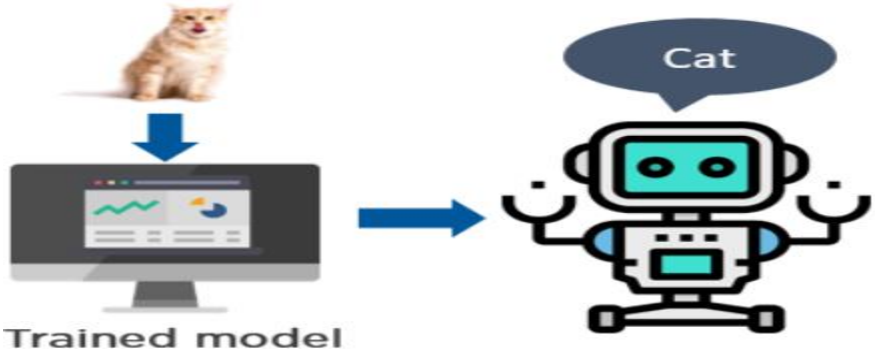
Supervised Learning

Overview:

Supervised learning is a type of machine learning that uses labelled data to train machine learning models. In labelled data, the output is already known. The model just needs to map the inputs to the respective outputs.

An example of supervised learning is to train a system that identifies the image of an animal.

Attached below, you can see that we have our trained model that identifies the picture of a cat.



Algorithms:

Some of the most popularly used supervised learning algorithms are:

- Linear Regression
- Logistic Regression
- Support Vector Machine
- K Nearest Neighbor
- Decision Tree
- Random Forest
- Naive Bayes

Working:

Supervised learning algorithms take labeled inputs and map them to the known outputs, which means you already know the target variable.

Now, let's focus on the training process for the supervised learning method.

Supervised Learning methods need external supervision to train machine learning models. Hence, the name supervised. They need guidance and additional information to return the desired result.

Applications:

Supervised learning algorithms are generally used for solving classification and regression problems.

Few of the top supervised learning applications are weather prediction, sales forecasting, stock price analysis.

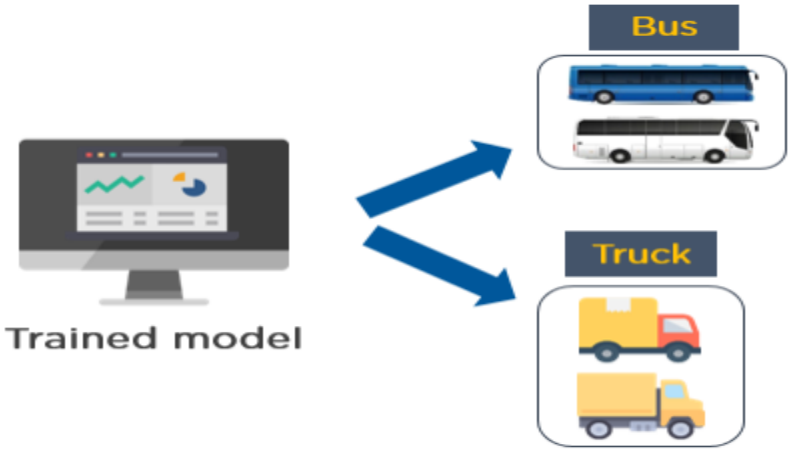
Now that you understand what Supervised learning is, let's see the next type of machine learning.

Unsupervised Learning

Overview:

Unsupervised learning is a type of machine learning that uses unlabeled data to train machines. Unlabeled data doesn't have a fixed output variable. The model learns from the data, discovers the patterns and features in the data, and returns the output.

Depicted below is an example of an unsupervised learning technique that uses the images of vehicles to classify if it's a bus or a truck. The model learns by identifying the parts of a vehicle, such as a length and width of the vehicle, the front, and rear end covers, roof hoods, the types of wheels used, etc. Based on these features, the model classifies if the vehicle is a bus or a truck.



Algorithms:

Selecting the right algorithm depends on the type of problem you are trying to solve. Some of the common examples of unsupervised learning are:

- K Means Clustering
- Hierarchical Clustering
- DBSCAN
- Principal Component Analysis

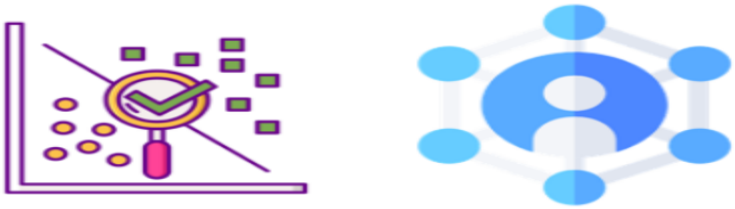
Working:

Unsupervised learning finds patterns and understands the trends in the data to discover the output. So, the model tries to label the data based on the features of the input data.

The training process used in unsupervised learning techniques does not need any supervision to build models. They learn on their own and predict the output.

Applications:

Unsupervised learning is used for solving clustering and association problems.



One of the applications of unsupervised learning is customer segmentation. Based on customer behavior, likes, dislikes, and interests, you can segment and cluster similar customers into a group. Another example where unsupervised learning algorithms are used is used churn rate analysis.

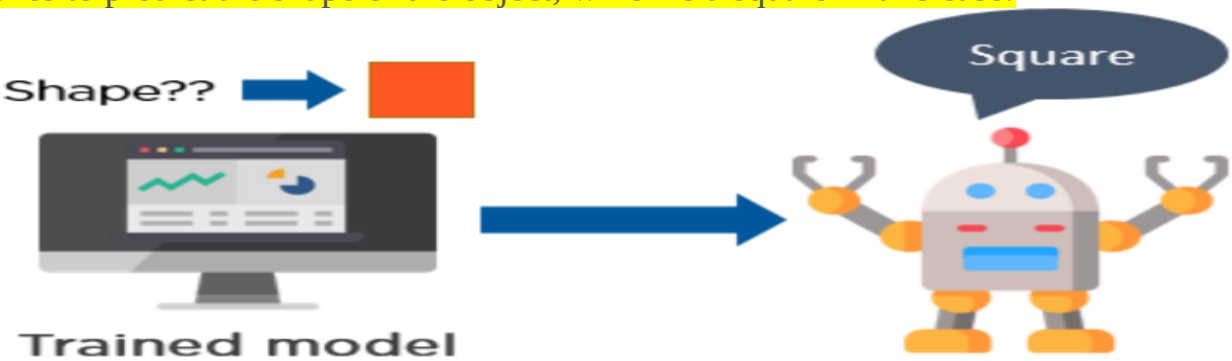
Let's see the third type of machine learning, i.e., reinforcement learning.

Reinforcement Learning

Overview

Reinforcement Learning trains a machine to take suitable actions and maximize its rewards in a particular situation. It uses an agent and an environment to produce actions and rewards. The agent has a start and an end state. But, there might be different paths for reaching the end state, like a maze. In this learning technique, there is no predefined target variable.

An example of reinforcement learning is to train a machine that can identify the shape of an object, given a list of different objects. In the example shown, the model tries to predict the shape of the object, which is a square in this case.



Algorithms

Some of the important reinforcement learning algorithms are:

1. Q-learning
2. Sarsa
3. Monte Carlo
4. Deep Q network

Working

Reinforcement learning follows trial and error methods to get the desired result. After accomplishing a task, the agent receives an award. An example could be to train a dog to catch the ball. If the dog learns to catch a ball, you give it a reward, such as a biscuit.

Reinforcement Learning methods do not need any external supervision to train models.

Reinforcement learning problems are reward-based. For every task or for every step completed, there will be a reward received by the agent. If the task is not achieved correctly, there will be some penalty added.

Now, let's see what applications we have in reinforcement learning.

Applications

Reinforcement learning algorithms are widely used in the gaming industries to build games. It is also used to train robots to do human tasks.

Inductive Learning Algorithm

Inductive Learning Algorithm (ILA) is an iterative and inductive machine learning algorithm which is used for generating a set of a classification rule, 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.

Basic Idea: 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 experts logic in the form of algorithms but this work is very tedious, time taking

and expensive. So we move towards the inductive algorithms which itself generate the strategy for performing a task and need not instruct separately at each step.

Need of ILA in presence of other machine learning algorithms: The ILA is a new algorithm which 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 lack of generalisation of rules.
- The ID3 and AQ used the decision tree production method which was too specific which were difficult to analyse and was 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 overcome the above problems

THE ILA ALGORITHM: General requirements at start of the 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.

Steps in the algorithm:-

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 what is the decision.

Example no.	**Place type**	**weather**	**location**	**decision**
I)	hilly	winter	kullu	Yes
II)	mountain	windy	Mumbai	No
III)	mountain	windy	Shimla	Yes
IV)	beach	windy	Mumbai	No
V)	beach	warm	goa	Yes
VI)	beach	windy	goa	No
VII)	beach	warm	Shimla	Yes

step 1 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

step (2-8) at iteration 1 row 3 & 4 column weather is selected and row 3 & 4 are marked. the rule is added to R IF 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 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 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 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. rule is added to R IF place type is beach AND 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 place type is hilly THEN the decision is yes.
- **Rule 3:** IF location is Shimla THEN the decision is yes.
- **Rule 4:** IF location is Mumbai THEN the decision is no.
- **Rule 5:** IF place type is beach AND the weather is windy THEN the decision is no.

==========

A DECISION TREE

A decision tree is a tree-like graph with nodes representing the place where we pick an attribute and ask a question; edges represent the answers the to the question; and the leaves represent the actual output or class label. They are used in non-linear decision making with simple linear decision surface.

Decision trees classify the examples by sorting them down the tree from the root to some leaf node, with the leaf node providing the classification to the example. Each node in the tree acts as a test case for some attribute, and each edge descending from that node corresponds to one of the possible answers to the test case. This process is recursive in nature and is repeated for every subtree rooted at the new nodes.

Let's illustrate this with help of an example. Let's assume we want to play badminton on a particular day — say Saturday — how will you decide whether to play or not. Let's say you go out and check if it's hot or cold, check the speed of the wind and humidity, how the weather is, i.e. is it sunny, cloudy, or rainy. You take all these factors into account to decide if you want to play or not.

So, you calculate all these factors for the last ten days and form a lookup table like the one below.

Day	Weather	Temperature	Humidity	Wind	Play?
1	Sunny	Hot	High	Weak	No
2	Cloudy	Hot	High	Weak	Yes

Day	Weather	Temperature	Humidity	Wind	Play?
3	Sunny	Mild	Normal	Strong	Yes
4	Cloudy	Mild	High	Strong	Yes
5	Rainy	Mild	High	Strong	No
6	Rainy	Cool	Normal	Strong	No
7	Rainy	Mild	High	Weak	Yes
8	Sunny	Hot	High	Strong	No
9	Cloudy	Hot	Normal	Weak	Yes
10	Rainy	Mild	High	Strong	No

Table 1. Obeservations of the last ten days.

Now, you may use this table to decide whether to play or not. But, what if the weather pattern on Saturday does not match with any of rows in the table? This may be a problem. A decision tree would be a great way to represent data like this because it takes into account all the possible paths that can lead to the final decision by following a tree-like structure.

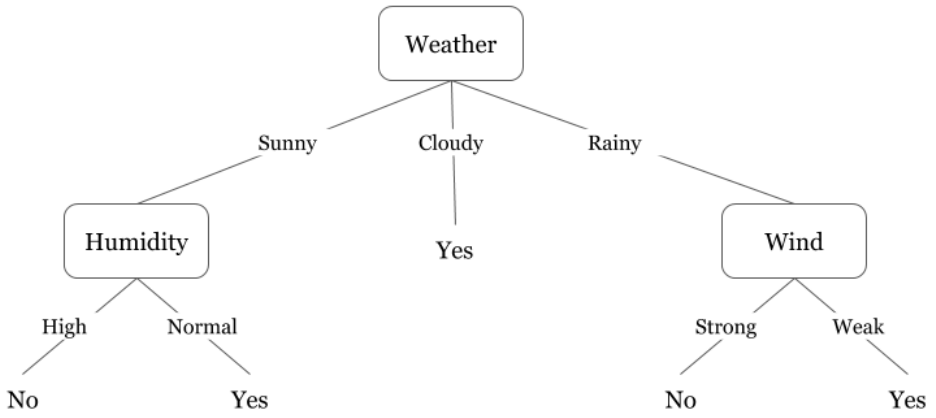


Fig 1. A decision tree for the concept Play Badminton

Fig 1. illustrates a learned decision tree. We can see that each node represents an attribute or feature and the branch from each node represents the outcome of that node. Finally, its the leaves of the tree where the final decision is made. If features are continuous, internal nodes can test the value of a feature against a threshold (see Fig. 2).

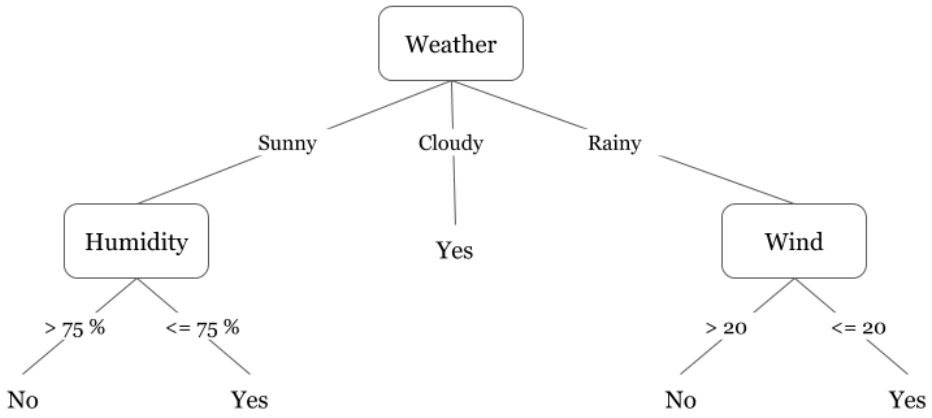


Fig 2. A decision tree for the concept Play Badminton (when attributes are continuous)

A general algorithm for a decision tree can be described as follows:

1. Pick the best attribute/feature. The best attribute is one which best splits or separates the data.
2. Ask the relevant question.
3. Follow the answer path.
4. Go to step 1 until you arrive to the answer.

The best split is one which separates two different labels into two sets

---

What is Heuristics?

A heuristic is a technique that is used to solve a problem faster than the classic methods. These techniques are used to find the approximate solution of a problem when classical methods do not. Heuristics are said to be the problem-solving techniques that result in practical and quick solutions.

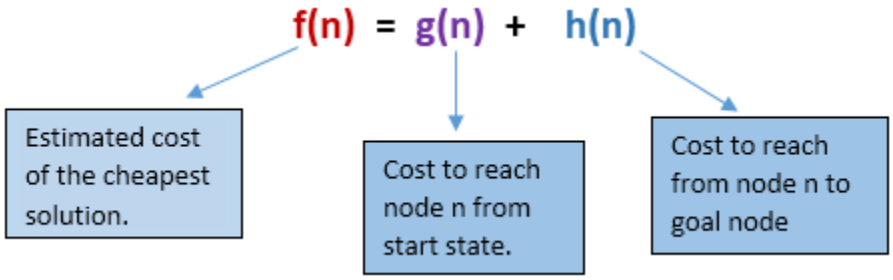
Heuristics are strategies that are derived from past experience with similar problems. Heuristics use practical methods and shortcuts used to produce the solutions that

may or may not be optimal, but those solutions are sufficient in a given limited timeframe.

1.) A* Search Algorithm:

A* search is the most commonly known form of best-first search. It uses heuristic function h(n), and cost to reach the node n from the start state g(n). It has combined features of UCS and greedy best-first search, by which it solve the problem efficiently. A* search algorithm finds the shortest path through the search space using the heuristic function. This search algorithm expands less search tree and provides optimal result faster. A* algorithm is similar to UCS except that it uses g(n)+h(n) instead of g(n).

In A* search algorithm, we use search heuristic as well as the cost to reach the node. Hence we can combine both costs as following, and this sum is called as a **fitness number**.



> At each point in the search space, only those node is expanded which have the lowest value of f(n), and the algorithm terminates when the goal node is found.

Algorithm of A* search:

Step1: Place the starting node in the OPEN list.

Step 2: Check if the OPEN list is empty or not, if the list is empty then return failure and stops.

Step 3: Select the node from the OPEN list which has the smallest value of evaluation function (g+h), if node n is goal node then return success and stop, otherwise

Step 4: Expand node n and generate all of its successors, and put n into the closed list. For each successor n', check whether n' is already in the OPEN or CLOSED list, if not then compute evaluation function for n' and place into Open list.

Step 5: Else if node n' is already in OPEN and CLOSED, then it should be attached to the back pointer which reflects the lowest g(n') value.

Step 6: Return to **Step 2**.

Advantages:

- A* search algorithm is the best algorithm than other search algorithms.
- A* search algorithm is optimal and complete.
- This algorithm can solve very complex problems.

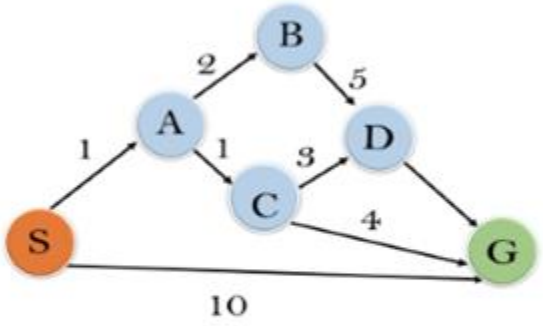
Disadvantages:

- It does not always produce the shortest path as it mostly based on heuristics and approximation.
- A* search algorithm has some complexity issues.
- The main drawback of A* is memory requirement as it keeps all generated nodes in the memory, so it is not practical for various large-scale problems.

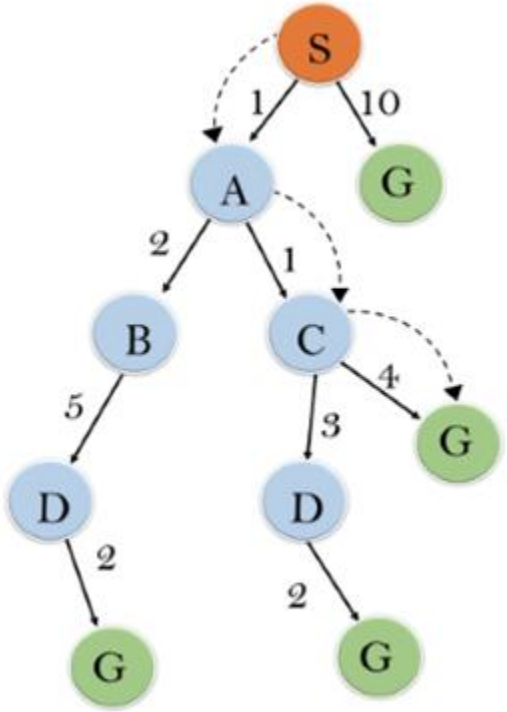
Example:

In this example, we will traverse the given graph using the A* algorithm. The heuristic value of all states is given in the below table so we will calculate the f(n) of each state using the formula f(n)= g(n) + h(n), where g(n) is the cost to reach any node from start state.

Here we will use OPEN and CLOSED list.



State	h(n)
S	5
A	3
B	4
C	2
D	6
G	0



Solution:

Initialization: {(S, 5)}

Iteration1: {(S--> A, 4), (S-->G, 10)}

Iteration2: {(S--> A-->C, 4), (S--> A-->B, 7), (S-->G, 10)}

Iteration3: {(S--> A-->C--->G, 6), (S--> A-->C--->D, 11), (S--> A-->B, 7), (S-->G, 10)}

Iteration 4 will give the final result, as **S--->A--->C--->G** it provides the optimal path with cost 6.

Points to remember:

- A* algorithm returns the path which occurred first, and it does not search for all remaining paths.
- The efficiency of A* algorithm depends on the quality of heuristic.
- A* algorithm expands all nodes which satisfy the condition f(n)<="" li="">

Complete: A* algorithm is complete as long as:

- Branching factor is finite.
- Cost at every action is fixed.

Optimal: A* search algorithm is optimal if it follows below two conditions:

- **Admissible:** the first condition requires for optimality is that h(n) should be an admissible heuristic for A* tree search. An admissible heuristic is optimistic in nature.
- **Consistency:** Second required condition is consistency for only A* graph-search.

If the heuristic function is admissible, then A* tree search will always find the least cost path.

Time Complexity: The time complexity of A* search algorithm depends on heuristic function, and the number of nodes expanded is exponential to the depth of solution d. So the time complexity is O(b^d), where b is the branching factor.

Space Complexity: The space complexity of A* search algorithm is **O(b^d)**

2)Hill Climbing Algorithm in Artificial Intelligence

- Hill climbing algorithm is a local search algorithm which continuously moves in the direction of increasing elevation/value to find the peak of the mountain or best solution to the problem. It terminates when it reaches a peak value where no neighbor has a higher value.
- Hill climbing algorithm is a technique which is used for optimizing the mathematical problems. One of the widely discussed examples of Hill climbing algorithm is Traveling-salesman Problem in which we need to minimize the distance traveled by the salesman.
- It is also called greedy local search as it only looks to its good immediate neighbor state and not beyond that.
- A node of hill climbing algorithm has two components which are state and value.
- Hill Climbing is mostly used when a good heuristic is available.
- In this algorithm, we don't need to maintain and handle the search tree or graph as it only keeps a single current state.

Features of Hill Climbing:

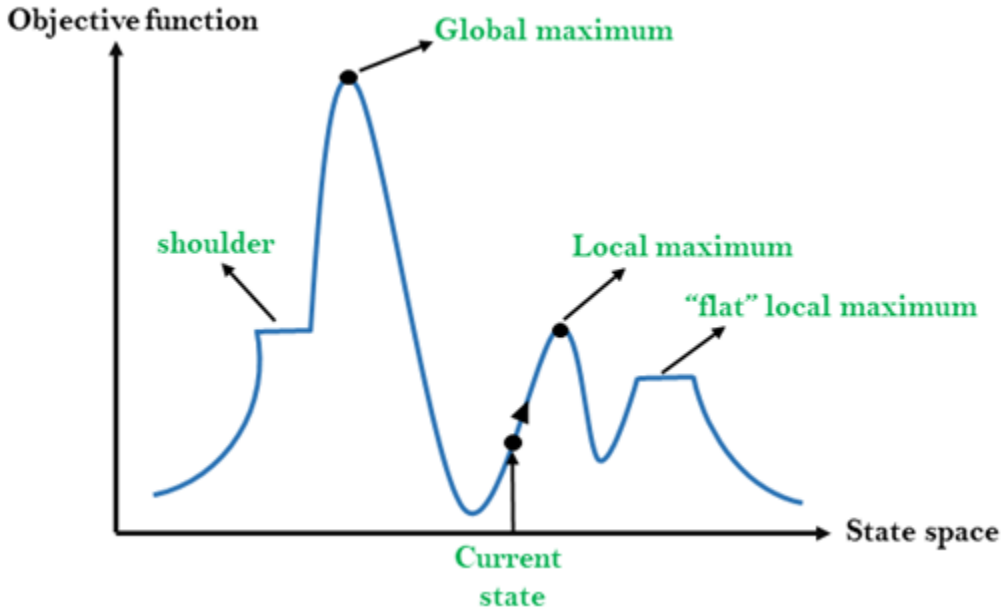
Following are some main features of Hill Climbing Algorithm:

- **Generate and Test variant:** Hill Climbing is the variant of Generate and Test method. The Generate and Test method produce feedback which helps to decide which direction to move in the search space.
- **Greedy approach:** Hill-climbing algorithm search moves in the direction which optimizes the cost.
- **No backtracking:** It does not backtrack the search space, as it does not remember the previous states.

State-space Diagram for Hill Climbing:

The state-space landscape is a graphical representation of the hill-climbing algorithm which is showing a graph between various states of algorithm and Objective function/Cost.

On Y-axis we have taken the function which can be an objective function or cost function, and state-space on the x-axis. If the function on Y-axis is cost then, the goal of search is to find the global minimum and local minimum. If the function of Y-axis is Objective function, then the goal of the search is to find the global maximum and local maximum.



Different regions in the state space landscape:

Local Maximum: Local maximum is a state which is better than its neighbor states, but there is also another state which is higher than it.

Global Maximum: Global maximum is the best possible state of state space landscape. It has the highest value of objective function.

Current state: It is a state in a landscape diagram where an agent is currently present.

Flat local maximum: It is a flat space in the landscape where all the neighbor states of current states have the same value.

Shoulder: It is a plateau region which has an uphill edge.

Types of Hill Climbing Algorithm:

- Simple hill Climbing:

- Steepest-Ascent hill-climbing:
- Stochastic hill Climbing:

1. Simple Hill Climbing:

Simple hill climbing is the simplest way to implement a hill climbing algorithm. **It only evaluates the neighbor node state at a time and selects the first one which optimizes current cost and set it as a current state**. It only checks it's one successor state, and if it finds better than the current state, then move else be in the same state. This algorithm has the following features:

- Less time consuming
- Less optimal solution and the solution is not guaranteed

Algorithm for Simple Hill Climbing:

- **Step 1:** Evaluate the initial state, if it is goal state then return success and Stop.
- **Step 2:** Loop Until a solution is found or there is no new operator left to apply.
- **Step 3:** Select and apply an operator to the current state.
- **Step 4:** Check new state:
    1. If it is goal state, then return success and quit.
    2. Else if it is better than the current state then assign new state as a current state.
    3. Else if not better than the current state, then return to step2.
- **Step 5:** Exit.

2. Steepest-Ascent hill climbing:

The steepest-Ascent algorithm is a variation of simple hill climbing algorithm. This algorithm examines all the neighboring nodes of the current state and selects one neighbor node which is closest to the goal state. This algorithm consumes more time as it searches for multiple neighbors

Algorithm for Steepest-Ascent hill climbing:

- **Step 1:** Evaluate the initial state, if it is goal state then return success and stop, else make current state as initial state.
- **Step 2:** Loop until a solution is found or the current state does not change.

1. Let SUCC be a state such that any successor of the current state will be better than it.
2. For each operator that applies to the current state:
    I. Apply the new operator and generate a new state.
    II. Evaluate the new state.
    III. If it is goal state, then return it and quit, else compare it to the SUCC.
    IV. If it is better than SUCC, then set new state as SUCC.
    V. If the SUCC is better than the current state, then set current state to SUCC.

- **Step 5:** Exit.

3. Stochastic hill climbing:

Stochastic hill climbing does not examine for all its neighbour before moving. Rather, this search algorithm selects one neighbour node at random and decides whether to choose it as a current state or examine another state.

==========

Important Questions

1. What are different types of ML Algorithm?
2. What is Confusion Matrix?
3. What is Cross Validation?
4. What is Ensemble Learning?
5. What is Heuristic Search?
6. Define Supervised Learning
7. What is Unsupervised Learning?
8. DefineReinforcement Learning?

9. Differentiate between Classification and Regression.
10. Differentiate between ML and DL.
11. Give a brief note on false positive and false negative in Confusion Matrix.
12. Define Precision & Recall.
13. What is Bias in ML?
14. Give a brief note on Decision tree Classification.
15. What is Training set & Test set in ML model?

- ✧ Discuss about the applications of ML.
- ✧ Define ML. Discuss with example why ML is important.
- ✧ Write short notes on Concept Learning.
- ✧ Discuss the Perspectives and issues in ML.
- ✧ Write short notes on Inductive Learning Algorithm.
- ✧ Write short notes on Supervised Learning.
- ✧ Give a brief note on Unsupervised Learning.
- ✧ Write short notes on Reinforcement Learning.
- ✧ Discuss about Hill Climbing Algorithm.
- ✧ Discuss about A* Algorithm.

A. Explain Candidate Elimination Algorithm.
B. Describe in detail about all the steps involved in Designing a Learning System.
C. Explain about Decision tree representation with example.
D. Briefly explain about Heuristic Search.
E. Write short notes on
    i) Supervised Learning
    ii) Unsupervised Learning
    iii) Reinforcement Learning