CMR INSTITUTE OF TECHNOLOGY

USN | | | | | | | | | | |

Internal Assesment Test –IV, February 2022

Sub:	MACHINE LEARNING							Code:	18MCA53
Date:	02-02-2022	Duration:	90 mins	Max Marks:	50	Sem:	V	Branch:	MCA

Note : Answer FIVE FULL Questions, choosing ONE full question from each Module

		Marks	OBE CO	OBE RBT
	Part-I			
Q1	Define a Well-Posed Learning Problem. Quote some successful applications of machine learning.	10	CO1	L2
	or			
Q2	Elaborate the design choice of choosing the training experience and choosing the Target Function while designing a learning system.	10	CO1	L2
	Part-II			
Q3	Write FIND-S algorithm and discuss the issues with the algorithm for Enjoy Sport problem.	10	CO3	L2
	or			
Q4	Consider the following training example. **Example** | **Sky** | **AirTemp** | **Humidity** | **Wind** | **Water** | **Forecast** | **EnjoySport** **1** | Sunny | Warm | Normal | Strong | Warm | Same | Yes **2** | Sunny | Warm | High | Strong | Warm | Same | Yes **3** | Rainy | Cold | High | Strong | Warm | Change | No **4** | Sunny | Warm | High | Strong | Cool | Change | Yes Show the general and specific boundaries of the version space after applying candidate elimination algorithm.	10	CO3	L2
	Part-III			
Q5	Draw decision tree for the given dataset and calculate the entropy and information gain. **Instance** | **Classification** | **a1** | **a2** 1 | + | T | T 2 | + | T | T 3 | - | T | F 4 | + | F | F 5 | - | F | T 6 | - | F | T	10	CO3	L2
	or			
Q 6	Design the decision tree for the following dataset and predict whether Golf will be played on the day. Day | Outlook | Temperature | Humidity | Wind | Play Golf D1 | Sunny | Hot | High | Weak | No D2 | Sunny | Hot | High | Strong | No D3 | Overcast | Hot | High | Weak | Yes D4 | Rain | Mild | High | Weak | Yes D5 | Rain | Cool | Normal | Weak | Yes D6 | Rain | Cool | Normal | Strong | No D7 | Overcast | Cool | Normal | Strong | Yes D8 | Sunny | Mild | High | Weak | No D9 | Sunny | Cool | Normal | Weak | Yes D10 | Rain | Mild | Normal | Weak | Yes D11 | Sunny | Mild | Normal | Strong | Yes D12 | Overcast | Mild | High | Strong | Yes D13 | Overcast | Hot | Normal | Weak | Yes D14 | Rain | Mild | High | Strong | No	10	CO4	L3

CMR INSTITUTE OF TECHNOLOGY

USN | | | | | | | | | | |

Internal Assesment Test –IV, February 2022

Sub:	MACHINE LEARNING						Code:	18MCA53
Date:	02-02-2022	Duration:	90 mins	Max Marks:	50	Sem: V	Branch:	MCA

	Note : Answer FIVE FULL Questions, choosing ONE full question from each Module	Marks	OBE	
			CO	RBT
Q7	**Part-IV** Write the ID3 algorithm. Explain the appropriate problems for decision tree learning.	10	CO3	L3
Q8	**or** What is linearly in separable problem? Design a two-layer Back propagation network for feed forward network.	10	CO3	L2
Q9	**Part-V** What is the perceptron training rule. Write the Gradient decent Algorithm and visualize the Hypothesis space for gradient decent rule.	10	CO3	L3
Q10	**or** Write the Back propagation algorithm for feed forward network with example.	10	CO3	L2

Answer Key

Part-I

Q1 Define a Well-Posed Learning Problem. Quote some successful applications of machine learning.

Well-Posed Learning Definition: 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 in T, as measured by P, improves with experience E.

Examples:

Checkers Game: A computer program that learns to play checkers might improve its performance as measured by its ability to win at the class of tasks involving playing checkers game, through experience obtained by playing games against itself:

***checkers learning problem*:**

- Task T: playing checkers
- Performance measure P: percent of games won against opponents
- Training experience E: playing practice games against itself

A handwriting recognition learning problem:

- Task T: recognizing and classifying handwritten words within images
- Performance measure P: percent of words correctly classified
- Training experience E: a database of handwritten words with given classifications

***A robot driving learning problem*:**

- Task T: driving on public four-lane highways using vision sensors
- Performance measure P: average distance travelled before an error (as judged by human overseer)
- Training experience E: a sequence of images and steering commands recorded while observing a human driver

Q2 **DESIGNING A LEARNING SYSTEM**

The basic design issues and approaches to machine learning are illustrated by designing a program to learn to play checkers, with the goal of entering it in the world checkers tournament

1. Choosing the Training Experience
2. Choosing the Target Function
3. Choosing a Function Approximation Algorithm
   1. Estimating training values
   2. Adjusting the weights

1. Choosing the Training Experience

- The first design choice is to choose the type of training experience from which the system will learn.
- The type of training experience available can have a significant impact on success or failure of the learner.

<u>There are three attributes which impact on success or failure of the learner</u>

1. Whether the training experience provides ***direct or indirect feedback*** regarding the choices made by the performance system.

   <u>For example, in checkers game</u>:

   In learning to play checkers, the system might learn from ***direct training examples*** consisting of ***individual checkers board states*** and ***the correct move for each***.

   Indirect training examples consisting of the ***move sequences*** and ***final outcomes*** of various games played. The information about the correctness of specific moves early in the game must be inferred indirectly from the fact that the game was eventually won or lost.

   Here the learner faces an additional problem of ***credit assignment***, or determining the degree to which each move in the sequence deserves credit or blame for the final outcome.

2. The degree to which the ***learner controls the sequence of training examples***

   For example, in checkers game:
   The learner might depends on the ***teacher*** to select informative board states and to provide the correct move for each.

   Alternatively, the learner might itself propose board states that it finds particularly confusing and ask the teacher for the correct move.

   The learner may have complete control over both the board states and (indirect) training classifications, as it does when it learns by playing against itself with ***no teacher present***.

3. How well it represents the ***distribution of examples*** over which the final system performance P must be measured

   For example, in checkers game:
   In checkers learning scenario, the performance metric P is the percent of games the system wins in the world tournament.

   If its training experience E consists only of games played against itself, there is a danger that this training experience might not be fully representative of the distribution of situations over which it will later be tested.
   It is necessary to learn from a distribution of examples that is different from those on which the final system will be evaluated.

2. Choosing the Target Function

The next design choice is to determine exactly what type of knowledge will be learned and how this will be used by the performance program.

Let's consider a checkers-playing program that can generate the legal moves from any board state.
The program needs only to learn how to choose the best move from among these legal moves. We must learn to choose among the legal moves, the most obvious choice for the type of information to be learned is a program, or function, that chooses the best move for any given board state.

1. Let ***ChooseMove*** be the target function and the notation is

   $$ChooseMove : B \rightarrow M$$

   which indicate that this function accepts as input any board from the set of legal board states B and produces as output some move from the set of legal moves M.
   ChooseMove is a choice for the target function in checkers example, but this function will turn out to be very difficult to learn given the kind of indirect training experience available to our system

2. An alternative target function is an ***evaluation function*** that assigns a ***numerical score*** to any given board state
   Let the target function V and the notation

   $$V:B \rightarrow R$$

   which denote that V maps any legal board state from the set B to some real value. Intend for this target function V to assign higher scores to better board states. If the system can successfully learn such a target function V, then it can easily use it to select the best move from any current board position.

Let us define the target value V(b) for an arbitrary board state b in B, as follows:

- If b is a final board state that is won, then V(b) = 100
- If b is a final board state that is lost, then V(b) = -100
- If b is a final board state that is drawn, then V(b) = 0
- 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 Function Approximation Algorithm

In order to learn the target function **f** we require a set of training examples, each describing a specific board state b and the training value Vtrain(b) for b.

Each training example is an ordered pair of the form (b, Vtrain(b)).

For instance, the following training example describes a board state b in which black has won the game (note x2 = 0 indicates that red has no remaining pieces) and for which the target function value Vtrain(b) is therefore +100.

((x1=3, x2=0, x3=1, x4=0, x5=0, x6=0), +100)

Function Approximation Procedure

1. Derive training examples from the indirect training experience available to the learner
2. Adjusts the weights wi to best fit these training examples

1. Estimating training values

A simple approach for estimating training values for intermediate board states is to assign the training value of Vtrain(b) for any intermediate board state b to be V(Successor(b))

Where ,

$\hat{V}$

V is the learner's current approximation to V

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

Rule for estimating training values

Vtrain(b) ← V (Successor(b))

2. Adjusting the weights

Specify the learning algorithm for choosing the weights wi to best fit the set of training examples {(b, Vtrain(b))}

$$E \equiv \sum_{\langle b, V_{train}(b)\rangle \in \text{ training examples}} (V_{train}(b) - \hat{V}(b))^2$$

A first step is to define what we mean by the bestfit to the training data.

One common approach is to define the best hypothesis, or set of weights, as that which minimizes the squared error E between the training values and the values predicted by the hypothesis.

Several algorithms are known for finding weights of a linear function that minimize E. One such algorithm is called the ***least mean squares, or LMS training rule***. For each observed training example it adjusts the weights a small amount in the direction that reduces the error on this training example

LMS weight update rule :- For each training example (b, Vtrain(b))

Use the current weights to calculate $\hat{V}$(b)
For each weight wi, update it as

$$wi \leftarrow wi + \eta\ (Vtrain\ (b) - \hat{V}(b))\ xi$$

Here η is a small constant (e.g., 0.1) that moderates the size of the weight update.

Working of weight update rule

- When the error (Vtrain(b)- $\hat{V}$(b)) is zero, no weights are changed.
- When (Vtrain(b) - $\hat{V}$(b)) is positive (i.e., when $\hat{V}$(b) is too low), then each weight is increased in proportion to the value of its corresponding feature. This will raise the value of $\hat{V}$(b), reducing the error.

If the value of some feature xi is zero, then its weight is not altered regardless of the error, so that the only weights updated are those whose features actually occur on the training example board.

Part-II

Q3 Write FIND-S algorithm and discuss the issues with the algorithm for Enjoy Sport problem.

FIND-S Algorithm

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

For each attribute constraint a_i in *h*

If the constraint a_i is satisfied by *x*

Then do nothing

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

Unanswered by FIND-S

1. Has the learner converged to the correct target concept?
2. Why prefer the most specific hypothesis?
3. Are the training examples consistent?
4. What if there are several maximally specific consistent hypotheses?

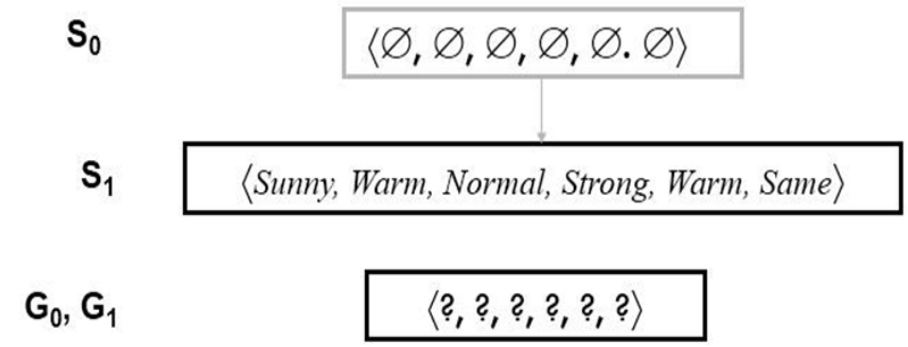
Q4 CANDIDATE-ELIMINTION algorithm begins by initializing the version space to the set of all hypotheses in H;

Initializing the G boundary set to contain the most general hypothesis in H G0 □?, ?, ?, ?, ?, ?□

Initializing the S boundary set to contain the most specific (least general) hypothesis S0 □□, □, □, □, □, □□

For training example d,

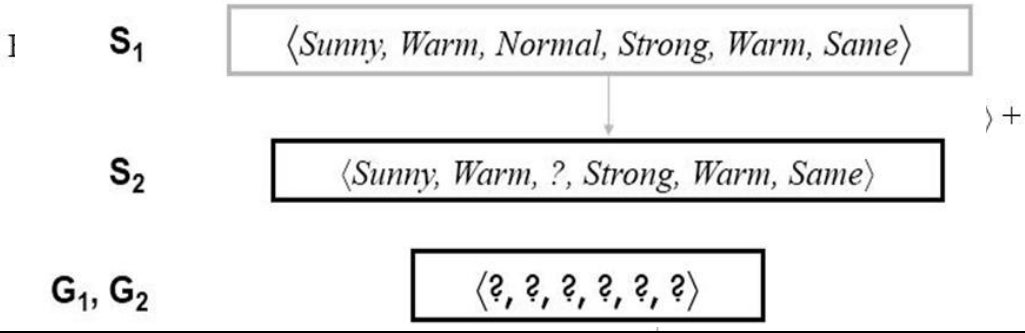
⟨Sunny, Warm, Normal, Strong, Warm, Same ⟩ +

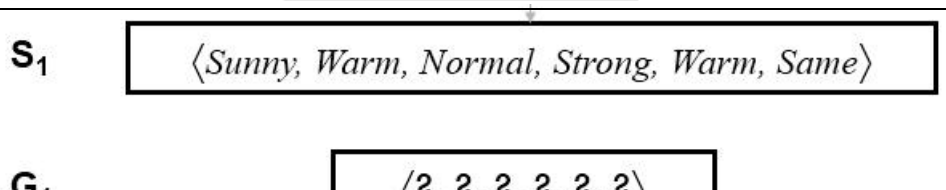


□ When the second training example is observed, it has a similar effect of generalizing S further to S2, leaving G again unchanged i.e., G2 = G1 = G0

For training example d,

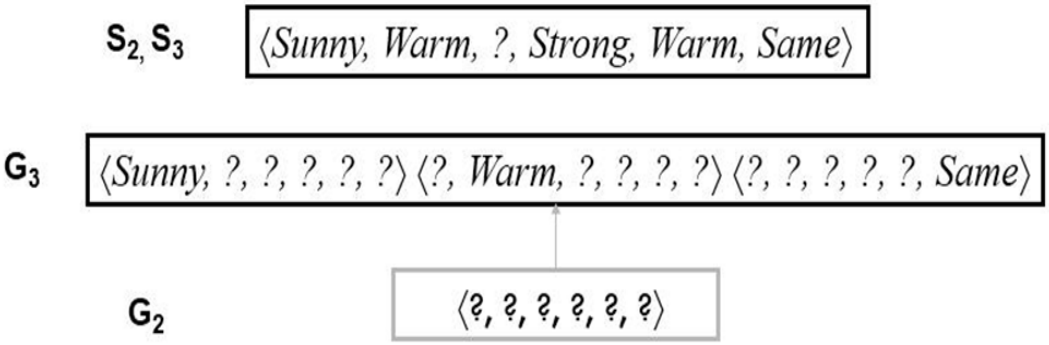
⟨Sunny, Warm, High, Strong, Warm, Same⟩ +





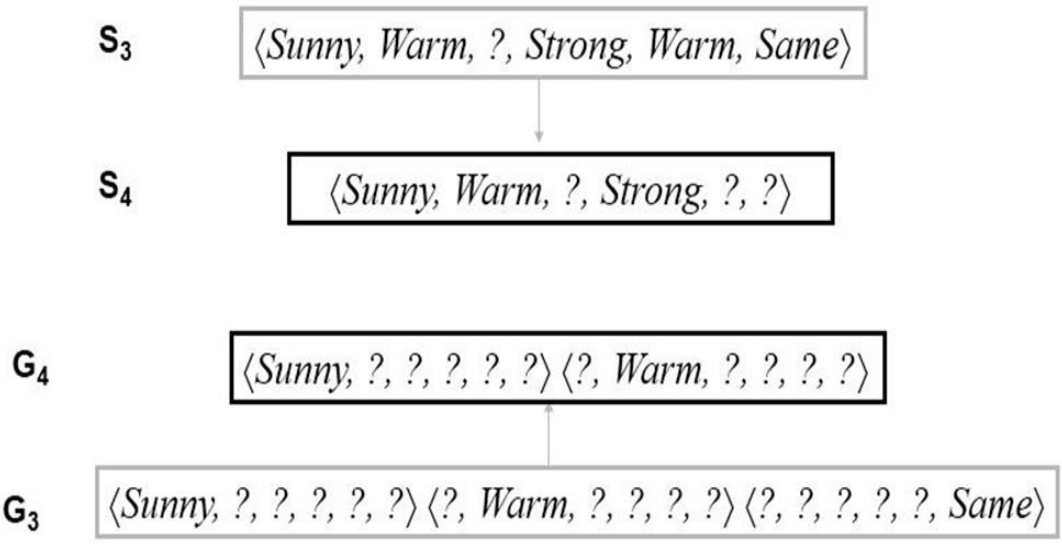
Consider the third training example.

For training example d, ⟨Rainy, Cold, High, Strong, Warm, Change⟩ –

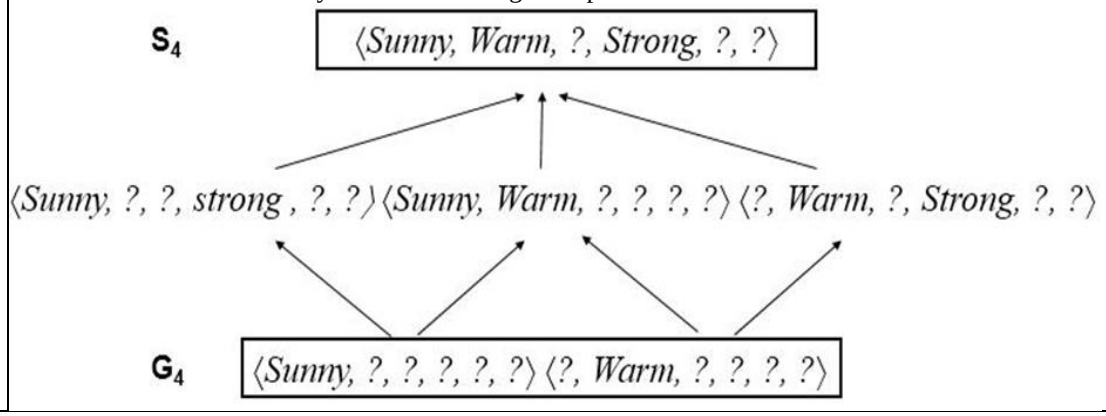


Consider the fourth training example

For training example d, ⟨Sunny, Warm, High, Strong, Cool Change⟩ +



After processing these four examples, the boundary sets S4 and G4 delimit the version space of all hypotheses consistent with the set of incrementally observed training examples.



Q5

Part-III

Draw decision tree for the given dataset and calculate the entropy and information gain.

Instance	Classification	a1	a2
1	+	T	T
2	+	T	T
3	-	T	F
4	+	F	F
5	-	F	T
6	-	F	T

Attribute: a1

$Values\ (a1) = T, F$

$S = [3+, 3-]$ $\quad Entropy(S) = 1.0$

$S_T = [2+, 1-]$ $\quad Entropy(S_T) = -\frac{2}{3} log_2 \frac{2}{3} - \frac{1}{3} log_2 \frac{1}{3} = 0.9183$

$S_F \leftarrow [1+, 2-]$ $\quad Entropy(S_F) = -\frac{1}{3} log_2 \frac{1}{3} - \frac{2}{3} log_2 \frac{2}{3} = 0.9183$

$Gain\ (S, a1)$ =Entropy (S) $\sum_{v \in \{T,F\}} \frac{|S_v|}{|S|} Entropy(S_v)$

$Gain(S, a1) = Entropy(S) - \frac{3}{6} Entropy(S_T) - \frac{3}{6} Entropy(S_F)$

$Gain(S, a1) = 1.0 - \frac{3}{6} * 0.9183 - \frac{3}{6} * 0.9183 = 0.0817$

Attribute: a2

$Values\ (a2) = T, F$

$S = [3+, 3-]$ $\quad Entropy(S) = 1.0$

$S_T = [2+, 2-]$ $\quad Entropy(S_T) = 1.0$

$S_F \leftarrow [1+, 1-]$ $\quad Entropy(S_F) = 1.0$

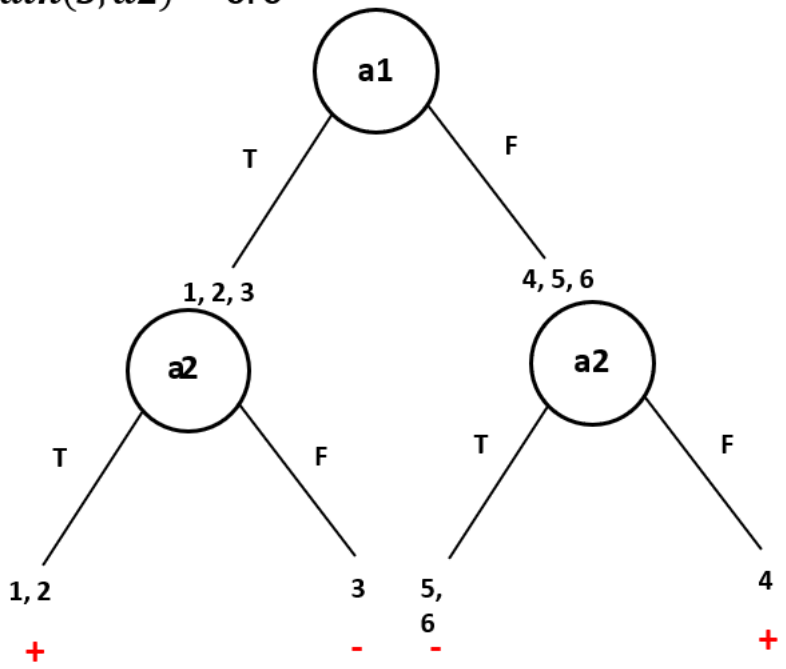
$Gain\ (S, a2) = Entropy(S) - \sum_{v \in \{T,F\}} \frac{|S_v|}{|S|} Entropy(S_v)$

$Gain(S, a2) = Entropy(S) - \frac{4}{6} Entropy(S_T) - \frac{2}{6} Entropy(S_F)$

$Gain(S, a2) = 1.0 - \frac{4}{6} * 1.0 - \frac{2}{6} * 1.0 = 0.0$

$Gain(S, a1) = 0.0817 \; - \; Maximum\ Gain$

$Gain(S, a2) = 0.0$



or

Q 6 Design the decision tree for the following dataset and predict whether Golf will be played on the day.

Day	Outlook	Temperature	Humidity	Wind	Play Golf
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

Attribute: Outlook

$Values(Outlook) = Sunny, Overcast, Rain$

$S = [9+, 5-]$ $\quad Entropy(S) = -\frac{9}{14}log_2\frac{9}{14} - \frac{5}{14}log_2\frac{5}{14} = 0.94$

$S_{Sunny} \leftarrow [2+, 3-]$ $\quad Entropy(S_{Sunny}) = -\frac{2}{5}log_2\frac{2}{5} - \frac{3}{5}log_2\frac{3}{5} = 0.971$

$S_{Overcast} \leftarrow [4+, 0-]$ $\quad Entropy(S_{Overcast}) = -\frac{4}{4}log_2\frac{4}{4} - \frac{0}{4}log_2\frac{0}{4} = 0$

$S_{Rain} \leftarrow [3+, 2-]$ $\quad Entropy(S_{Rain}) = -\frac{3}{5}log_2\frac{3}{5} - \frac{2}{5}log_2\frac{2}{5} = 0.971$

$$Gain(S, Outlook) = Entropy(S) - \sum_{v \in \{Sunny, Overcast, Rain\}} \frac{|S_v|}{|S|} Entropy(S_v)$$

$Gain(S, Outlook)$

$$= Entropy(S) - \frac{5}{14}Entropy(S_{Sunny}) - \frac{4}{14}Entropy(S_{Overcast}) - \frac{5}{14}Entropy(S_{Rain})$$

$$Gain(S, Outlook) = 0.94 - \frac{5}{14}0.971 - \frac{4}{14}0 - \frac{5}{14}0.971 = 0.2464$$

Attribute: Temp

$Values(Temp) = Hot, Mild, Cool$

$S = [9+, 5-]$ $\quad Entropy(S) = -\frac{9}{14}log_2\frac{9}{14} - \frac{5}{14}log_2\frac{5}{14} = 0.94$

$S_{Hot} \leftarrow [2+, 2-]$ $\quad Entropy(S_{Hot}) = -\frac{2}{4}log_2\frac{2}{4} - \frac{2}{4}log_2\frac{2}{4} = 1.0$

$S_{Mild} \leftarrow [4+, 2-]$ $\quad Entropy(S_{Mild}) = -\frac{4}{6}log_2\frac{4}{6} - \frac{2}{6}log_2\frac{2}{6} = 0.9183$

$S_{Cool} \leftarrow [3+, 1-]$ $\quad Entropy(S_{Cool}) = -\frac{3}{4}log_2\frac{3}{4} - \frac{1}{4}log_2\frac{1}{4} = 0.8113$

$$Gain(S, Temp) = Entropy(S) - \sum_{v \in \{Hot, Mild, Cool\}} \frac{|S_v|}{|S|} Entropy(S_v)$$

$Gain(S, Temp)$

$$= Entropy(S) - \frac{4}{14}Entropy(S_{Hot}) - \frac{6}{14}Entropy(S_{Mild}) - \frac{4}{14}Entropy(S_{Cool})$$

$$Gain(S, Temp) = 0.94 - \frac{4}{14}1.0 - \frac{6}{14}0.9183 - \frac{4}{14}0.8113 = 0.0289$$

Attribute: Humidity

$Values\ (Humidity) = High, Normal$

$S = [9+, 5-]$ $\quad Entropy(S) = -\frac{9}{14} log_2 \frac{9}{14} - \frac{5}{14} log_2 \frac{5}{14} = 0.94$

$S_{High} \leftarrow [3+, 4-]$ $\quad Entropy(S_{High}) = -\frac{3}{7} log_2 \frac{3}{7} - \frac{4}{7} log_2 \frac{4}{7} = 0.9852$

$S_{Normal} \leftarrow [6+, 1-]$ $\quad Entropy(S_{Normal}) = -\frac{6}{7} log_2 \frac{6}{7} - \frac{1}{7} log_2 \frac{1}{7} = 0.5916$

$$Gain\ (S, Humidity) = Entropy(S) - \sum_{v \in \{High, Normal\}} \frac{|S_v|}{|S|} Entropy(S_v)$$

$$Gain(S, Humidity) = Entropy(S) - \frac{7}{14} Entropy(S_{High}) - \frac{7}{14} Entropy(S_{Normal})$$

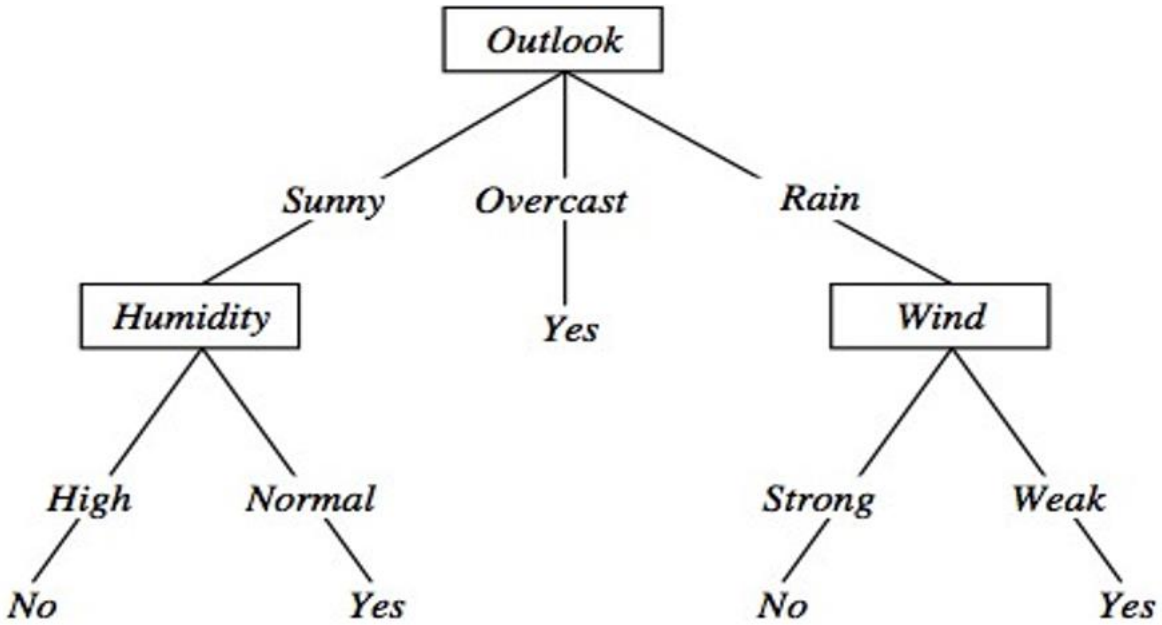
$$Gain(S, Humidity) = 0.94 - \frac{7}{14} 0.9852 - \frac{7}{14} 0.5916 = 0.1516$$

$Gain\ (S, Outlook) = 0.2464$

$Gain(S, Temp) = 0.0289$

$Gain(S, Humidity) = 0.1516$

$Gain(S, Wind) = 0.0478$



Q7

Issues in learning decision trees include

1 Avoiding Overfitting the Data

2 Reduced error pruning

3 Rule post-pruning
4 Incorporating Continuous-Valued Attributes
5 Alternative Measures for Selecting Attributes
6 Handling Training Examples with Missing Attribute Values
7 Handling Attributes with Differing Costs

Avoiding Overfitting the Data

8 The ID3 algorithm grows each branch of the tree just deeply enough to perfectly classify the training examples but it can lead to difficulties when there is noise in the data, or when the number of training examples is too small to produce a representative sample of the true target function. This algorithm can produce trees that overfit the training examples.

How can it be possible for tree h to fit the training examples better than h', but for it to perform more poorly over subsequent examples?

9 Overfitting can occur when the training examples contain random errors or noise
10 When small numbers of examples are associated with leaf nodes.

Approaches to avoiding overfitting in decision tree learning

11 Pre-pruning (avoidance): Stop growing the tree earlier, before it reaches the point where it perfectly classifies the training data
12 Post-pruning (recovery): Allow the tree to overfit the data, and then post-prune the tree

Criterion used to determine the correct final tree size

13 Use a separate set of examples, distinct from the training examples, to evaluate the utility of post-pruning nodes from the tree
14 Use all the available data for training, but apply a statistical test to estimate whether expanding (or pruning) a particular node is likely to produce an improvement beyond the training set
15 Use measure of the complexity for encoding the training examples and the decision tree, halting growth of the tree when this encoding size is minimized. This approach is called the Minimum Description Length

MDL – Minimize : size(tree) + size (misclassifications(tree))

Reduced-Error Pruning

16 Reduced-error pruning, is to consider each of the decision nodes in the tree to be candidates for pruning
17 ***Pruning*** a decision node consists of removing the subtree rooted at that node, making it a leaf node, and assigning it the most common classification of the training examples affiliated with that node
18 Nodes are removed only if the resulting pruned tree performs no worse than-the original over the validation set.
19 Reduced error pruning has the effect that any leaf node added due to coincidental regularities in the training set is likely to be pruned because these same coincidences are unlikely to occur in the validation set

2. Incorporating Continuous-Valued Attributes

Continuous-valued decision attributes can be incorporated into the learned tree.

There are two methods for Handling Continuous Attributes

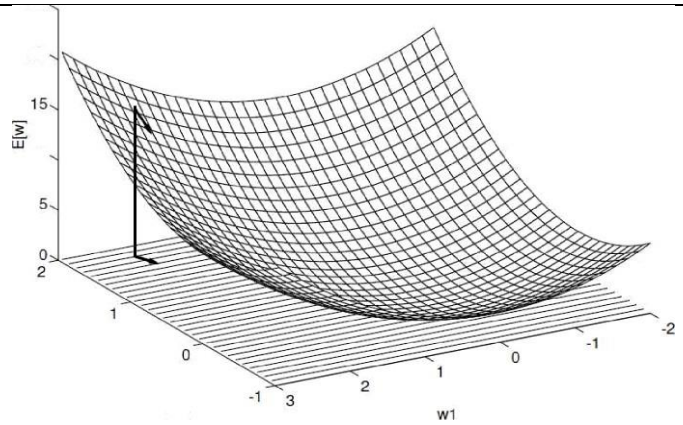
20 Define new discrete valued attributes that partition the continuous attribute value into a discrete set of intervals.
E.g., {high ≡ Temp > 35º C, med ≡ 10º C < Temp ≤ 35º C, low ≡ Temp ≤ 10º C}

21 Using thresholds for splitting nodes
e.g., A ≤ a produces subsets A ≤ a and A > a
What threshold-based Boolean attribute should be defined based on Temperature?

Temperature:	40	48	60	72	80	90
PlayTennis:	No	No	Yes	Yes	Yes	No

22 Pick a threshold, c, that produces the greatest information gain

	23 In the current example, there are two candidate thresholds, corresponding to the values of Temperature at which the value of PlayTennis changes: (48 + 60)/2, and (80 + 90)/2. 24 The information gain can then be computed for each of the candidate attributes, Temperature >54, and Temperature >85 and the best can be selected (Temperature >54) **2. Alternative Measures for Selecting Attributes** • The problem is if attributes with many values, Gain will select it ? • Example: consider the attribute Date, which has a very large number of possible values. (e.g., March 4, 1979). • If this attribute is added to the PlayTennis data, it would have the highest information gain of any of the attributes. This is because Date alone perfectly predicts the target attribute over the training data. Thus, it would be selected as the decision attribute for the root node of the tree and lead to a tree of depth one, which perfectly classifies the training data. • This decision tree with root node Date is not a useful predictor because it perfectly separates the training data, but poorly predict on subsequent examples. **3. Handling Training Examples with Missing Attribute Values** The data which is available may contain missing values for some attributes Example: Medical diagnosis □ <Fever = true, Blood-Pressure = normal, …, Blood-Test = ?, …> □ Sometimes values truly unknown, sometimes low priority (or cost too high) *Strategies for dealing with the missing attribute value* □ If node n test A, assign most common value of A among other training examples sorted to node n □ Assign most common value of A among other training examples with same target value □ Assign a probability pi to each of the possible values vi of A rather than simply assigning the most common value to A(x) **5. Handling Attributes with Differing Costs** □ In some learning tasks the instance attributes may have associated costs. □ For example: In learning to classify medical diseases, the patients described in terms of attributes such as Temperature, BiopsyResult, Pulse, BloodTestResults, etc. □ These attributes vary significantly in their costs, both in terms of monetary cost and cost to patient comfort Decision trees use low-cost attributes where possible, depends only on high-cost attributes only when needed to produce reliable classifications.
	or
Q8	What is linearly in separable problem? Design a two-layer Back propagation network for feed forward network.
Q9	• Perceptron learning converges to a consistent model if D (training set) is linearly separable. • If the data is not linearly separable than this will not converge. • If the training examples are not linearly separable, the delta rule converges toward a best-fit approximation to the target concept. • The key idea behind the *delta rule* is to use *gradient descent* to search the hypothesis space of possible weight vectors to find the weights that best fit the training examples.



- Gradient descent search determines a weight vector that minimizes E by starting with an arbitrary initial weight vector, then repeatedly modifying it in small steps.
- At each step, the weight vector is altered in the direction that produces the steepest descent along the error surface depicted in above figure. This process continues until the global minimum error is reached.

Derivation of the Gradient Descent Rule

How to calculate the direction of steepest descent along the error surface?

The direction of steepest can be found by computing the derivative of E with respect to each component of the vector $\vec{w}$. This vector derivative is called the gradient of E with respect to $\vec{w}$, written as

$$\nabla E[\vec{w}] \equiv \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \cdots \frac{\partial E}{\partial w_n}\right] \qquad \text{equ. (3)}$$

The gradient specifies the direction of steepest increase of E, the training rule for gradient descent is

$$\vec{w} \leftarrow \vec{w} + \Delta\vec{w}$$

Where,

$$\Delta\vec{w} = -\eta\nabla E(\vec{w}) \qquad \text{equ. (4)}$$

- Here η is a positive constant called the learning rate, which determines the step size in the gradient descent search.
- The negative sign is present because we want to move the weight vector in the direction that decreases E.

This training rule can also be written in its component form

$$w_i \leftarrow w_i + \Delta w_i$$

Where,

$$\Delta w_i = -\eta\frac{\partial E}{\partial w_i} \qquad \text{equ. (5)}$$

$$\Delta w_i = \eta \sum_{d \in D} (t_d - o_d)\, x_{id} \qquad \text{equ. (7)}$$

$$\frac{\partial}{\partial w_i} \qquad \frac{\partial}{\partial w_i} 2 \sum_d$$

$$= \frac{1}{2} \sum_d \frac{\partial}{\partial w_i} (t_d - o_d)^2$$

$$= \frac{1}{2} \sum_d 2(t_d - o_d) \frac{\partial}{\partial w_i} (t_d - o_d)$$

$$= \sum_d (t_d - o_d) \frac{\partial}{\partial w_i} (t_d - \vec{w} \cdot \vec{x}_d)$$

$$\frac{\partial E}{\partial w_i} = \sum_d (t_d - o_d)(-x_{i,d}) \qquad \text{equ. (6)}$$

Substituting Equation (6) into Equation (5) yields the weight update rule for gradient descent

or

Q10 Write the Back propagation algorithm for feed forward network with example.

Create a feed-forward network with n_{in} inputs, n_{hidden} hidden units, and n_{out} output units.

Initialize all network weights to small random numbers (e.g., between −.05 and .05).

Until the termination condition is met, Do

- **For each $\langle \vec{x}, \vec{t} \rangle$ in *training_examples*, Do**

  Propagate the input forward through the network:

  1. **Input the instance $\vec{x}$ to the network and compute the output o_u of every unit u in the network.**

  Propagate the errors backward through the network:

  2. **For each network output unit k, calculate its error term δ_k**

$$\delta_k \leftarrow o_k(1 - o_k)(t_k - o_k) \qquad \text{(T4.3)}$$

  3. **For each hidden unit h, calculate its error term δ_h**

$$\delta_h \leftarrow o_h(1 - o_h) \sum_{k \in outputs} w_{kh} \delta_k \qquad \text{(T4.4)}$$

  4. **Update each network weight w_{ji}**

$$w_{ji} \leftarrow w_{ji} + \Delta w_{ji}$$

  where

$$\Delta w_{ji} = \eta\, \delta_j\, x_{ji} \qquad \text{(T4.5)}$$

Back Propagation Algorithm