CMR INSTITUTE OF TECHNOLOGY

USN | | | | | | | | | |

Internal Assesment Test –I, September 2020

Sub:	MACHINE LEARNING							Code:	18MCA53
Date:	15-09-2020	Duration:	90 mins	Max Marks:	50	Sem:	V	Branch:	MCA

Answer Any 5 **QUESTION**s

		Marks	OBE CO	OBE RBT
1a)	Define well-posed learning problem and explain with the help of an example.	4	CO1	L1
1b)	Discuss the following with respect to the learning task for "A checkers learning problem'. i) Choosing the training experience. ii) Choosing the target function. iii) Choosing a function approximation algorithm.	6	CO1	L2
2)	Write FIND-S algorithm and discuss the issues with the algorithm	10	CO1	L2
3)	Consider the given below following training example. (table below) Show the general and specific boundaries of the version space after applying candidate elimination algorithm.	10	CO1	L5
4	Explain the various stages involved in designing a learning system in brief.	10	CO1	L2
5	What is decision tree and discuss the use of decision tree for classification with an example.	10	CO2	L2
6)	Summarize the practical issues of decision tree learning.	10	CO2	L3
7)	Design the decision tree for the following dataset and predict whether Golf will be played on the day. (table below)	10	CO2	L5
8)	What do you mean by gain and entropy? How it is used to build decision tree. Explain it with the help of an example.	10	CO2	L2

Table for Question 3:

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

Table for Question 7:

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

CMR INSTITUTE OF TECHNOLOGY

USN | | | | | | | | | |

Internal Assessment Test 1 – Sep. 2020

Sub:	MACHINE LEARNING								**Sub Code:**	18MCA 53
Date:	15-09-2020	Duration:	90 min's	Max Marks:	50	**Sem**	5^{th}		**Branch:**	**MCA**

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

	PART I	MARKS	OBE CO	OBE RBT
1a)	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	4	CO1	L1

1b)	**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
		6	CO1	L2

system will learn.

- The type of training experience available can have a significant impact on success or failure of the learner.

There are three attributes which impact on success or failure of the learner

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

   For example, in checkers game:

   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)

<u>Function Approximation Procedure</u>

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. <u>Estimating training values</u>

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 ,

- $\hat{V}$ 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) ← $\hat{V}$ (Successor(b))

2. <u>Adjusting the weights</u>

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

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

$$w_i \leftarrow w_i + \eta\,(V_{train}(b) - \hat{V}(b))\,x_i$$

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

<u>Working of weight update rule</u>

- 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.

2) **FIND-S: FINDING A MAXIMALLY SPECIFIC HYPOTHESIS**

<u>**FIND-S Algorithm**</u>

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*

10 CO1 L2

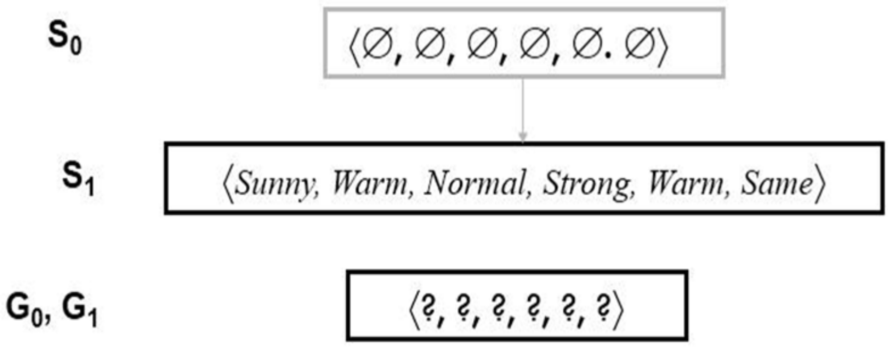
<u>**Unanswered by FIND-S**</u>

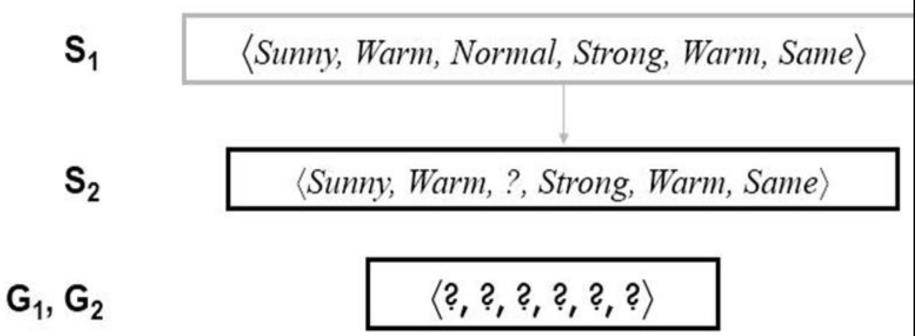
1. Has the learner converged to the correct target concept?
2. Why prefer the most specific hypothesis?
3. Are the training examples consistent?
4. What if there are several maximally specific consistent hypotheses?

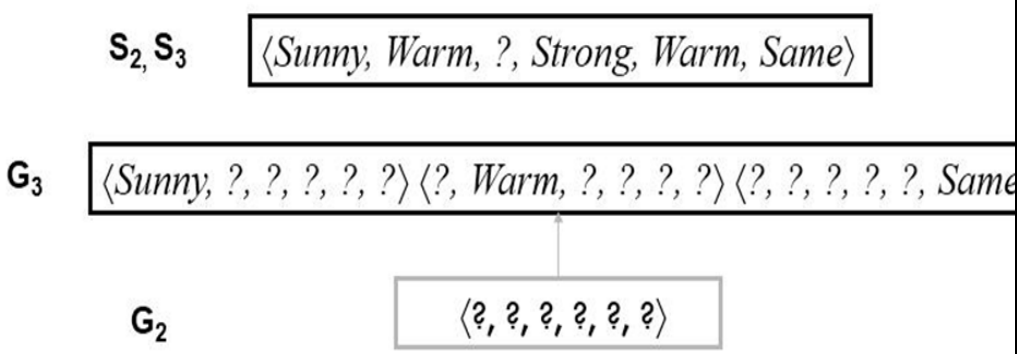
<table>
<tr>
<td>3)</td>
<td>
CANDIDATE-ELIMINTION algorithm begins by initializing the version space to the set of all hypotheses in H;

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 ⟩ +


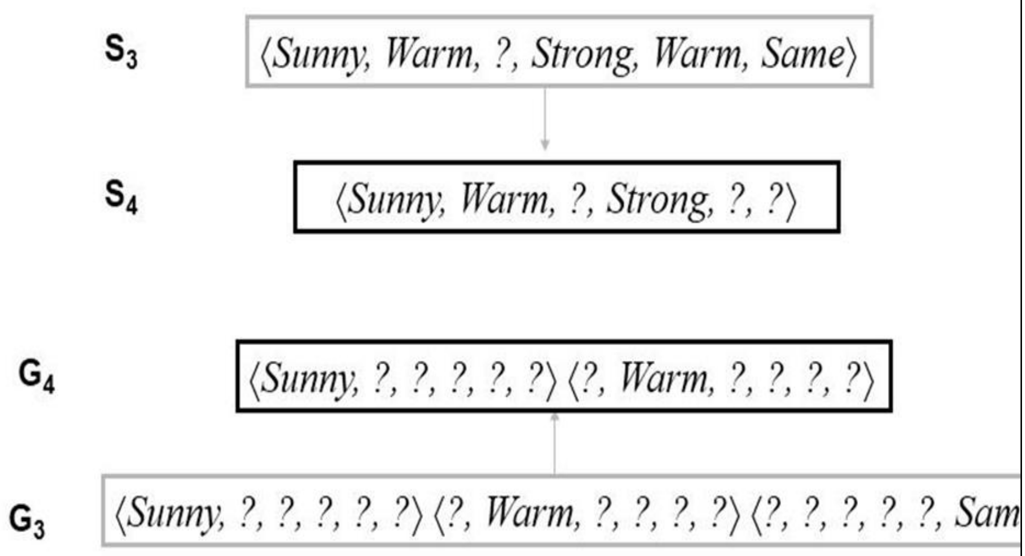
□ <u>When the second training example is observed,</u> it has a similar effect of generalizing S further to S2, leaving G again unchanged i.e., G2 = G1 = G0
For training example d,
⟨Sunny, Warm, High, Strong, Warm, Same⟩ +


<u>Consider the third training example</u>.
For training example d,
⟨Rainy, Cold, High, Strong, Warm, Change ⟩ –


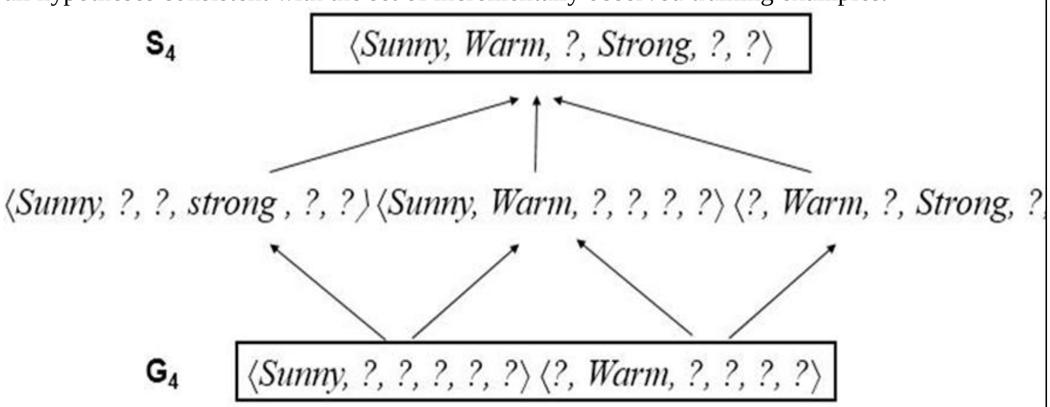
<u>Consider the fourth training example</u>
</td>
<td>10</td>
<td>CO1</td>
<td>L5</td>
</tr>
</table>

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.



4)

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

4. Choosing the Training Experience
5. Choosing the Target Function
6. Choosing a Representation for the Target Function
7. Choosing a Function Approximation Algorithm
    1. Estimating training values
    2. Adjusting the weights
8. The Final Design

10 CO1 L2

2. 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>

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

For example, in checkers game:

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.

4. 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***.

5. 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.

3. 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

3. 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

$$V{:}B \qquad \rightarrow R$$

   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 Representation for the Target Function

Let's choose a simple representation - for any given board state, the function ***c*** will be calculated as a linear combination of the following board features:

- xl: the number of black pieces on the board
- x2: the number of red pieces on the board
- x3: the number of black kings on the board
- x4: the number of red kings on the board
- x5: the number of black pieces threatened by red (i.e., which can be captured on red's next turn)
- x6: the number of red pieces threatened by black

Thus, learning program will represent as a linear function of the form

$$\hat{V}(b) = w_0 + w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + w_5x_5 + w_6x_6$$

Where,

- w0 through w6 are numerical coefficients, or weights, to be chosen by the learning algorithm.
- Learned values for the weights w1 through w6 will determine the relative importance of the various board features in determining the value of the board
- The weight w0 will provide an additive constant to the board value

4. 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

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

2. 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 ,

- $\hat{V}$ 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

$$\textbf{Vtrain(b)} \leftarrow \hat{\textbf{V}} \textbf{ (Successor(b))}$$

3. Adjusting the weights

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

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

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

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.

The Final Design

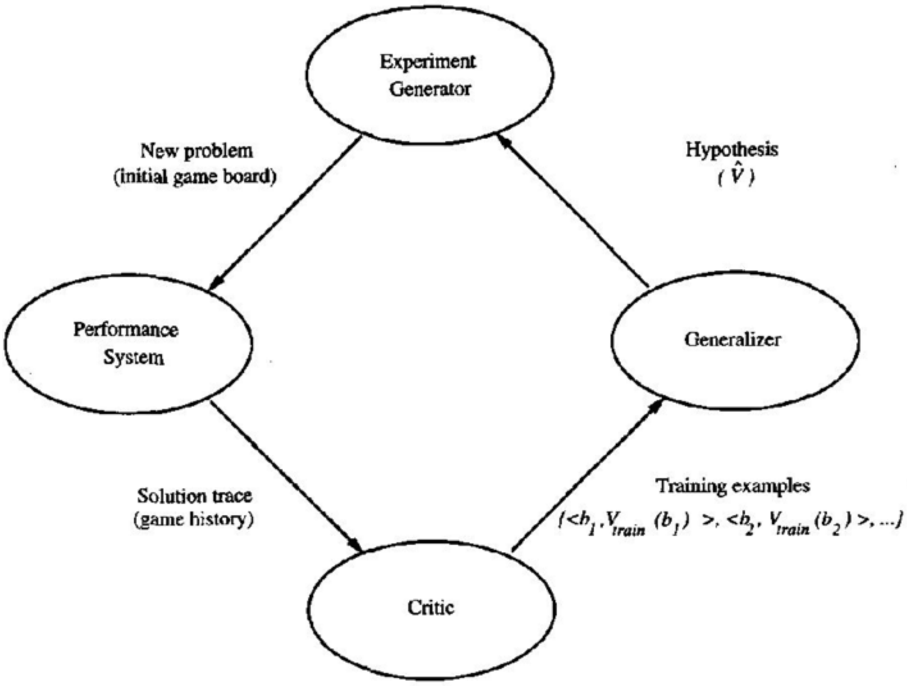
<table>
<tr>
<td></td>
<td>
The final design of checkers learning system can be described by four distinct program modules that represent the central components in many learning systems

1. **The Performance System** is the module that must solve the given performance task by using the learned target function(s). It takes an instance of a new problem (new game) as input and produces a trace of its solution (game history) as output.

2. **The Critic** takes as input the history or trace of the game and produces as output a set of training examples of the target function

3. **The Generalizer** takes as input the training examples and produces an output hypothesis that is its estimate of the target function. It generalizes from the specific training examples, hypothesizing a general function that covers these examples and other cases beyond the training examples.

4. **The Experiment Generator** takes as input the current hypothesis and outputs a new problem (i.e., initial board state) for the Performance System to explore. Its role is to pick new practice problems that will maximize the learning rate of the overall system.



</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5)</td>
<td>
Decision tree learning is a method for approximating discrete-valued target functions, in which the learned function is represented by a decision tree.

DECISION TREE REPRESENTATION

- Decision trees classify instances by sorting them down the tree from the root to some leaf node, which provides the classification of the instance.
- Each node in the tree specifies a test of some attribute of the instance, and each branch descending from that node corresponds to one of the possible values for this attribute.
- An instance is classified by starting at the root node of the tree, testing the attribute specified by this node, then moving down the tree branch corresponding to the value of the attribute in the given example. This process is then repeated for the subtree rooted at the new node.
</td>
<td>10</td>
<td>CO2</td>
<td>L2</td>
</tr>
</table>

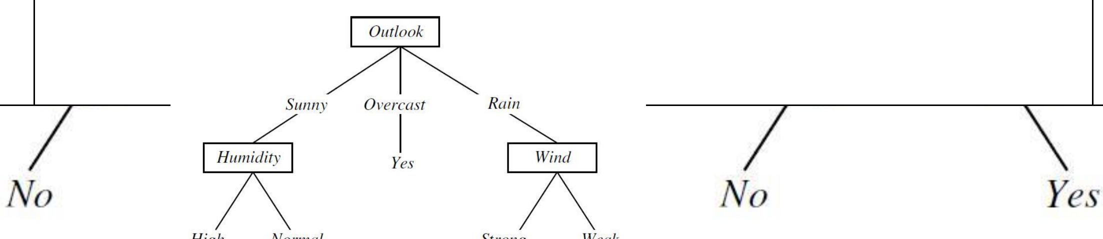


	FIGURE: A decision tree for the concept ***PlayTennis***. An example is classified by sorting it through the tree to the appropriate leaf node, then returning the classification associated with this leaf. **Example:-** Let's say we have a sample of 50 students with three variables Gender (Boy/ Girl), Class(X/ XI) and Height (5 to 6 ft). 20 out of these 50 play cricket in rest time. Suppose you want to find on unknown dataset which contains all the features(Gender, class, height) that he/she will play or not in rest time. This is where decision tree supports, it will separate the students based on all values of three variable and identify the variable, which creates the best uniform sets of students			
6)	<u>Issues in learning decision trees include</u> 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 **<u>Avoiding Overfitting the Data</u>** 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	10	CO2	L3

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. <u>Incorporating Continuous-Valued Attributes</u>

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

<u>*There are two methods for Handling Continuous Attributes*</u>

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)

3. <u>Alternative Measures for Selecting Attributes</u>

- 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.

4. <u>Handling Training Examples with Missing Attribute Values</u>

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)

<u>*Strategies for dealing with the missing attribute value*</u>

- 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

<table>
<tr><td></td><td>
• 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.
</td><td></td><td></td><td></td></tr>
<tr><td>7)</td><td>
Step 1:
Total – 14 Yes(p) - 9 No(n) – 5
Attributes: Outlook ={Sunny, Overcast, Rain} Temperature = {Hot, Mild, Cool}
Humidity = {High, Normal} Wind ={Weak, Strong}
Step 2: Calculate the entropy of the dataset
Entropy(S) = $- p_+ \log_2 p_+ - p_- \log_2 p_-$
$= - \frac{p}{p+n} log\left(\frac{p}{p+n}\right) - \frac{n}{p+n} log\left(\frac{n}{p+n}\right)$
=- (9/(9+5)) log(9/(9+5)) –(5/(9+5)) log(5/(9+5))
=-(9/14) log(9/14) – (5/14) log(5/14)
= (-0.643)(-0.637) – (0.357)(-1.486)
=0.940
Step 3:
i) Select Outlook attribute
Outlook ={Sunny, Overcast, Rain}
Sunny : Yes(p)- 2 No(n)- 3
Overcast: Yes(p)- 4 No(n)- 0
Rain: Yes(p)- 3 No(n)- 2

a) Entropy of Outlook attribute
$= - \frac{p}{p+n} log\left(\frac{p}{p+n}\right) - \frac{n}{p+n} log\left(\frac{n}{p+n}\right)$
Entropy(Outlook = Sunny) = -(2/5)log(2/5) – (3/5)log(3/5)
= -(0.4)(-1.322) – (0.6)(-0.737) =0.971
Entropy(Outlook = Overcast) = -(4/4)log(4/4) – 0= 0
Entropy(Outlook = Rain) = -(3/5)log(3/5) – (2/5)log(2/5)
=-(0.6)(-0.737) – (0.4)(-1.322) = 0.971

b) Average Information Entropy(I)
I(Outlook) = ((2+3)/(9+5))*0.971 + ((3+2)/(9+5))*0.971 + 0
= (5/14)*0.971 + 0.3571*0.971
= 0.693

c) Information Gain(Outlook) = Entropy(S) –I(Outlook)
= 0.940 - 0.940 = 0.247
ii) Select Temperature attribute
Temperature = {Hot, Mild, Cool}
Hot : p=2 n= 2
Mild: p=4 n=2
Cool: p=3 n=1

a. Calculate the entropy for Temperature
$= - \frac{p}{p+n} log\left(\frac{p}{p+n}\right) - \frac{n}{p+n} log\left(\frac{n}{p+n}\right)$
Entropy(Temperature= Hot)= -(2/4)log(2/4) – (2/4)log(2/4)
= -(0.5)(-1) – (0.5)(-1)
= 1
</td><td>10</td><td>CO2</td><td>L5</td></tr>
</table>

Entropy(Temperature = Mild) = -(4/6)log(4/6) – (2/6)log(2/6)
= -(0.66)(-0.599) – (0.33)(-1.599)
= 0.923

Entropy(Temperature = Cool) = -(3/4)log(3/4) – (1/4)log(1/4)
= -(0.75)(-0.415) – (0.25)(-2)
= 0.811

b. Average Information Entropy(I)

I(Temperature) = (4/14)*1 + (6/14)* 0.923 + (4/14)* 0.811 = 0.913

c. Information Gain(Temperature)

IG(Temperature) = Entropy(S) – I(Temperature)
= *0.940* - 0.913
= 0.027

iii) **Select Humidity attribute**

Humidity = {High, Normal}

High: p: 3 n:4

Normal: p: 6 n: 1

a. Calculate the entropy for Temperature

$$=-\frac{p}{p+n}\log\left(\frac{p}{p+n}\right) - \frac{n}{p+n}\log\left(\frac{n}{p+n}\right)$$

Entropy(Humidity = High) = - (3/7)log(3/7) – (4/7)log(4/7)
= -(0.4286)(-1.2223) – (0.5714)(-0.8074)
= 0.985

Entropy(Humidity = Normal) = -(6/7)log(6/7) – (1/7)log(1/7)
= -(0.8571)(-0.2225)-(0.1429)(-2.8069) = 0.591

b. Average Information Entropy

c. Average Information Entropy(I)

I(Humidity) = (7/14)*0.985 + (7/14)*0.591
= 0.788

d. Information Gain(Humidity)

IG(Humidity) = Entropy(S) – I(Humidity)
= *0.940 – 0.788*
= 0.152

iv) **Select Windy attribute**

Wind ={Weak, Strong}

Weak: p:6 n:2

Strong: p:3 n:3

a. Calculate the entropy for Windy

$$=-\frac{p}{p+n}\log\left(\frac{p}{p+n}\right) - \frac{n}{p+n}\log\left(\frac{n}{p+n}\right)$$

Entropy(Windy = Weak) = -(6/8)log(6/8) – (2/8)log(2/8)
= 0.811

Entropy(Windy = Strong) = -(3/6)log (3/6) – (3/6)log(3/6)
= 1

b. Average Information Entropy(I)

I(Windy) = (8/14)* 0.811 + (6/14)*1
= 0.892

c. Information Gain(Windy)

IG(Windy) = Entropy(S) – I(Windy)
= *0.940 – 0.892 = 0.048*

IG(Outlook) = *0.247*

IG(Temperature) = 0.27

IG(Humidity) = 0.152

IG(Windy) = *0.048*

Highest Information Gain is 0.247 -> Outlook

P:2 N:3 Total:5

Temperature= {hot, cool, mild}

Hot:p:0 n:2

Cool: p:1 n:0

Mild: p:1 n:1

Humidity={High, Normal}

High: p:0 n:3

Normal:p:2 n:0

Windy:{Weak, Strong}

Weak: p:1 n:2

Strong: p:1 n:1

1. **Calculate the entropy of Dataset(S)**

   $=-\frac{p}{p+n}\log\left(\frac{p}{p+n}\right) - \frac{n}{p+n}\log\left(\frac{n}{p+n}\right)$

   Entropy = -(2/5)log(2/5) – (3/5)log(3/5) = 0.971

2. **Calculate the Information Gain**

   a. **Calculate entropy of humidity**

      Entropy(Humidity = High) = 0 – (3/3)log(3/3) = 0

      Entropy(Humidity = Normal) = 0

   b. **Calculate Average information entropy(I) of humidity**

      I(Humidity) = 0

   c. **Information gain of humidity**

      IG(Humidity) = Entropy(S) – I(Humidity)

      =0.971 – 0 = 0.971

   d. **Calculate entropy of Windy**

      Windy:{Weak, Strong}

      Weak: p:1 n:2

      Strong: p:1 n:1

      $=-\frac{p}{p+n}\log\left(\frac{p}{p+n}\right) - \frac{n}{p+n}\log\left(\frac{n}{p+n}\right)$

      Entropy(Windy = Weak) = -(1/3)log(1/3) – (2/3)log(2/3) = 0.918

      Entropy(wind = Strong) = -(1/2)log(1/2) – (1/2) log(1/2) =1

   e. **Calculate Average information entropy of windy**

      I(Windy) = (3/5) *0.918 + (2/5)*1 = 0.951

   f. **Information gain of windy**

      IG(Windy) = Entropy(S) - I(Windy)

      = 0.971- 0.951 = 0.020

g. **Calculate entropy of temperature**

Temperature= {hot, cool, mild}

Hot:p:0 n:2

Cool: p:1 n:0

Mild: p:1 n:1

$$=-\frac{p}{p+n}log\left(\frac{p}{p+n}\right) - \frac{n}{p+n}log\left(\frac{n}{p+n}\right)$$

Entropy (Temperature = hot) = 0

Entropy(Temperature = Cool) = 0

Entropy(Temperature = mild) = -(1/2)log(1/2) – (1/2)log(1/2) = 1

h. **Calculate Average information entropy of temperature**

I(temperature) = (2/5) * 0 + (1/5)*0 + (2/5)*1= 0.4

i. **Information gain of temperature**

IG(Temperature) =0.971 - 0.4 = 0.571

3. **Select the attribute with highest information gain**

IG(Temperature) =0.971 - 0.4 = 0.571

IG(Windy) = 0.020

IG(Humidity) = 0.971

Total = 5

P=3

N= 2

1. Calculate the entropy of the dataset(S)

$$\frac{p}{p+n}log\left(\frac{p}{p+n}\right) - \frac{n}{p+n}log\left(\frac{n}{p+n}\right)$$

Entropy = - (3/5)log(3/5) – (2/5)log(2/5) = 0.971

2. **Calculate the information gain**

a. **Calculate entropy of temperature**

Temperature ={mild, cool}

Mild:p:2 n:1

Cool:p:1 n:1

Entropy(temperature = mild) = -(2/3)log(2/3) – (1/3)log(1/3) = 0.918

Entropy(temperature = cool) =-(1/2)log(1/2) – (1/2)log(1/2) = 1

b. **Calculate average information entropy of temperature**

I(Temperature) = 0.951

c. **Information gain of temperature**

0.971– 0.951 = 0.20

d. **Calculate entropy of Humidity**

Entropy(Humidity= High) = 1

Entropy(Humidity = Normal) = 0.918

e. **Calculate average information entropy of humidity**

I(Humidity) = 0.951

f. **Information gain of humidity**

Gain = 0.971 – 0.951 = 0.020

g. **Calculate entropy of Windy**

Entropy(Windy = Strong) = 0

Entropy(Windy = Weak) = 0

h. **Calculate average information entropy of Windy**

I(Windy) = 0

i. **Information gain of Windy**

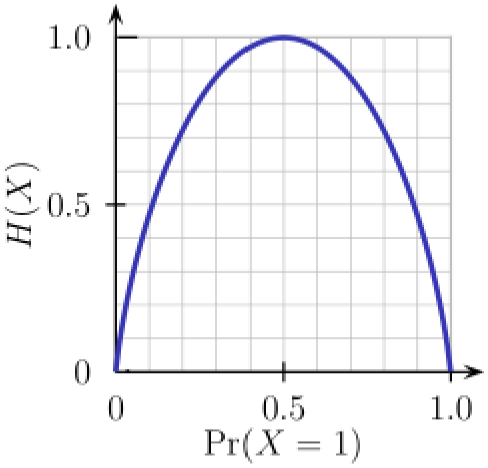
Gain = 0.971 – 0 = 0.971

3. Select the attribute with highest information gain

Select Windy.

8) Entropy

Entropy is a measure of the randomness in the information being processed. The higher the entropy, the harder it is to draw any conclusions from that information. Flipping a coin is an example of an action that provides information that is random.



From the above graph, it is quite evident that the entropy H(X) is zero when the probability is either 0 or 1. The Entropy is maximum when the probability is 0.5 because it projects perfect randomness in the data and there is no chance if perfectly determining the outcome.

ID3 follows the rule — A branch with an entropy of zero is a leaf node and A brach with entropy more than zero needs further splitting.

Mathematically Entropy for 1 attribute is represented as:

$$E(S) = \sum_{i=1}^{c} - p_i \log_2 p_i$$

Play Golf	
Yes	No
9	5

Entropy(PlayGolf) = Entropy (5,9)

= Entropy (0.36, 0.64)

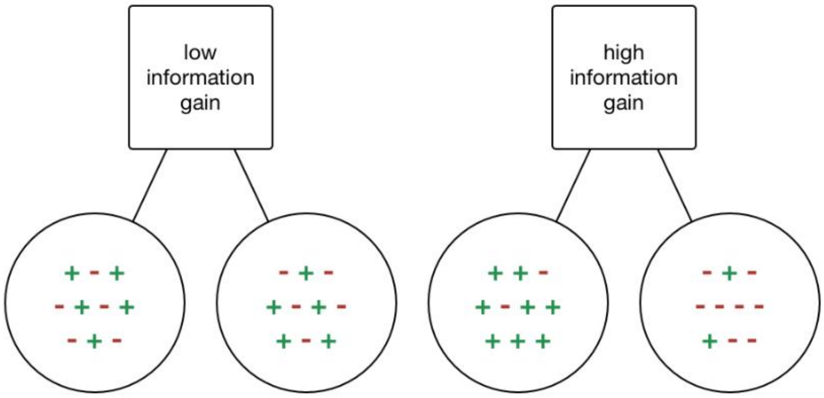
= - (0.36 $\log_2$ 0.36) - (0.64 $\log_2$ 0.64)

= 0.94

Information Gain

Information gain or **IG** is a statistical property that measures how well a given attribute separates the training examples according to their target classification. Constructing a decision tree is all about finding an attribute that returns the highest information gain and the smallest entropy.

Marks	CO	Level
10	CO2	L2



Information gain is a decrease in entropy. It computes the difference between entropy before split and average entropy after split of the dataset based on given attribute values. ID3 (Iterative Dichotomiser) decision tree algorithm uses information gain.

Mathematically, IG is represented as:

$$\text{Information Gain}(T,X) = \text{Entropy}(T) - \text{Entropy}(T, X)$$

$$\begin{aligned} IG(PlayGolf, Outlook) &= E(PlayGolf) - E(PlayGolf, Outlook) \\ &= 0.940 - 0.693 \\ &= 0.247 \end{aligned}$$