Scheme of Evaluation
Internal Assessment Test 2 – Oct.2019

Sub:	Machine Learning							**Code:**	15CS73
Date**:**	14/10/2019	Duration:	90mins	Max Marks:	50	**Sem:**	VII	**Branch:**	ISE A,B

Note: Answer Any Five Questions

Question #		**Description**	**Marks Distribution**		**Max Marks**
1		• Concept of decision tree learning • Decision tree using ID3 algorithm	5M 5M	10M	10 M
2	a)	• Calculating entropy • Calculating information gain	2.5M 2.5M	5 M	10 M
	b)	• Listing the appropriate problems	5M	5 M	
3		• Calculating the overall entropy • Calculating individual attribute gain values along with entropy • Identifying root node and sub nodes • Constructing final decision tree	2M 4M 2M 2M	10M	10 M
4		• Gradient descent algorithm • Deriving the equation	5M 5M	10 M	10 M
5	a)	• Application of neural network with explanation • Diagramatic representation	3M 2M	5 M	10 M
	b)	• Explanation of perceptron concept • Diagrametic representation	3 M 2M	5 M	
6		• Backpropagation algorithm • Deriving the derivatives rule	4M 6M	10 M	10M
7	a)	• Solving perceptron A and B • Justification	2M 2M	4M	10M
	b)	• Issues in decision tree learning 1. Avoiding Overfitting the Data Reduced error pruning Rule post-pruning 2. Incorporating Continuous-Valued Attributes	1.2*5=6M	6M	

		3. Alternative Measures for Selecting Attributes 4. Handling Training Examples with Missing Attribute Values 5. Handling Attributes with Differing Costs			
8)		• Calculating the overall entropy • Calculating individual attribute gain values along with entropy • Identifying root node and sub nodes • Constructing final decision tree	2M 4M 2M 2M	10M	10M

Internal Assessment Test 2 Solutions– Oct.2019

Sub:	Machine Learning							**Code:**	15CS73
Date**:**	14/10/2019	Duration:	90mins	Max Marks:	50	**Sem:**	VII	**Branch:**	ISE A,B

Note: Answer Any Five Questions

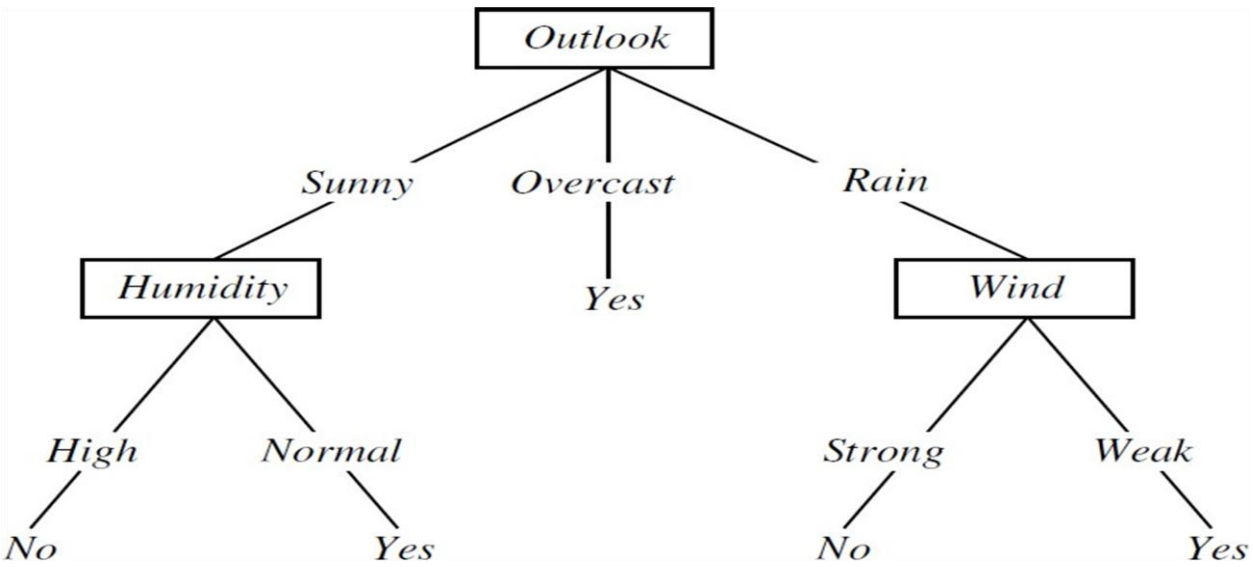
1. Explain the concept of decision tree learning. Discuss the necessary measure required to select the attributes for building a decision tree using ID3 algorithm

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.

Figure 1 illustrates a typical learned decision tree.



This decision tree classifies Saturday mornings according to whether they are suitable for playing tennis. For example, the instance

$$\langle Outlook = Sunny,\ Temperature = Hot,\ Humidity = High,\ Wind = Strong \rangle$$

would be sorted down the left most branch of this decision tree and would therefore be classified as a negative instance (i.e., the tree predicts that *PlayTennis = No*).

In general, decision trees represent a disjunction of conjunctions of constraints on the attribute values of instances. For example, the decision tree shown in Figure 1 corresponds to the expression

$$(Outlook = Sunny \wedge Humidity = Normal)$$
$$\vee \quad (Outlook = Overcast)$$
$$\vee \quad (Outlook = Rain \wedge Wind = Weak)$$

ID3 Algorithm:

ID3 stands for Iterative Dichotomiser 3

- ID3 basic algorithm learns decision trees by constructing them top-down, beginning with the question "which attribute should be tested at the root of the tree?" The best attribute is selected and used as the test at the root node of the tree.
- A descendant of the root node is then created for each possible value of this attribute, and the training examples are sorted to the appropriate descendant node.
- The entire process is then repeated using the training examples associated with each descendant node to select the best attribute to test at that point in the tree.

Which Attribute Is the Best Classifier?

- The central choice in the ID3 algorithm is selecting which attribute to test at each node in the tree.
- ID3 uses ***information gain*** measure to select among the candidate attributes at each step while growing the tree.
- To define information gain, we begin by defining a measure called entropy.
  Entropy measures the impurity of a collection of examples.

$$Entropy\ (S) \equiv -p_{\oplus} \log_2 p_{\oplus} - p_{\ominus} \log_2 p_{\ominus}$$

Where,
p_+ is the proportion of positive examples in S
p_- is the proportion of negative examples in S.

- The entropy is 0 if all members of S belong to the same class
- The entropy is 1 when the collection contains an equal number of positive and negative examples
- If the collection contains unequal numbers of positive and negative examples, the entropy is between 0 and 1

Information gain, is the expected reduction in entropy caused by partitioning the examples according to this attribute.

- The information gain, Gain(S,A) of an attribute A, relative to a collection of examples S, is defined as

$$Gain(S, A) = Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Entropy(S_v)$$

2.a) For the transaction shown in the table compute the following.

i) Entropy of the collection of transaction records of the table with respect to classification.

ii) What are the information gain of a1 and a2 relative to the transactions of the table.

Instance	1	2	3	4	5	6	7	8	9
a1	T	T	T	F	F	F	F	T	F
a2	T	T	F	F	T	T	F	F	T
Target Class	+	+	-	+	-	-	-	+	-

i) S is the given collection ,

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

p_i is the proportion of S belonging to the class i

Entropy(S) = 1 *,when equal number* of positive and negative examples.

ii) **Information gain(S,A2)**

$$Gain(S, A) = Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Entropy(S_v)$$

Gain(S,A2) = 1- { (4/6)*Entropy(T) + (2/6)* Entropy(F)}

=1- {(4/6)*(equal no.of + and -) + (2/6)*(equal no.of + and -)}

= 1-{(4/6)*1 + (2/6)*1}

= 0

2.b) Write the appropriate problems used for decision tree learning & List the applications of decision tree learning.

Appropriate problems for decision tree learning

Although a variety of decision tree learning methods have been developed with somewhat differing capabilities and requirements, decision tree learning is generally best suited to problems with the following characteristics:

- ***Instances are represented by attribute-value pairs***. Instances are described by a fixed set of attributes (e.g., Temperature) and their values (e.g., Hot). The easiest situation for decision tree learning is when each attribute takes on a small number of disjoint possible values (e.g., Hot, Mild, Cold).
- ***The target function has discrete output values***. The decision tree assigns a boolean classification (e.g., yes or no) to each example. Decision tree methods easily extend to learning functions with more than two possible output values.
- ***Disjunctive descriptions may be required***. As noted above, decision trees naturally represent disjunctive expressions.
- ***The training data may contain errors***. Decision tree learning methods are robust to errors, both errors in classifications of the training examples and errors in the attribute values that describe these examples.
- ***The training data may contain missing attribute values***. Decision tree methods can be used even when some training examples have unknown values (e.g., if the Humidity of the day is known for only some of the training examples).

Applications :

Decision tree learning has therefore been applied to problems such as learning to classify medical patients by their disease, equipment malfunctions by their cause, and loan applicants by their likelihood of defaulting on payments.

Such problems, in which the task is to classify examples into one of a discrete set of possible categories are often referred to as classification problems.

3. Create and explain the decision tree for the following transactions using ID3 algorithm.

Tid	Refund	Marital Status	Taxable Income	Cheat
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

Entropy(S) = -(3/10)log (3/10)-(7/10)log (7/10)

= 0.8813

Info.gain(S,Refund) = Entropy(S) – [(3/10)* Entropy(Yes) +(7/10) *Entropy(No)]

Entropy (Yes) = 0 bcz both belongs to –ve class

Entropy (No) = -(3/7) log(3/7) – (4/7) log(4/7)
= 0.9852

Hence
Info.gain(S, Refund) = Entropy(S) – [(3/10)*0 +(7/10) *0.9852]
= 0.8813 - [0 +(7/10)*0.9852]
= 0.19166
Info.gain(S, Marital status) =
Entropy(S) – [(4/10)* Entropy (Single) +(4/10) *Entropy (Married)+(2/10)*Entropy (Divorced)]

Entropy (Single) = 1
Entropy (Married) = 0
Entropy (Divorced) = 1
Info.gain(S, Marital Status) = Entropy(S) – [(4/10)*(1) + (4/10) *(0) +((2/10)*1]
= 0.2813
Info.gain(S, Taxable Income) = Entropy(S) – [(3/10)* Entropy (<80k) + (7/10) *Entropy (>80k)]

Entropy (<80k) = 0
Entropy (>80k) = - (3/7) log (3/7) – (4/7) log (4/7)
= 0.9852
Info.gain(S, Taxable income) = Entropy(S) – [(3/10)*(0) +(7/10) *(0.9852)]
= 0.19166

Among all marital status has highest information gain so it becomes the root node

Marital status=Single: Entropy (Single) =1
Gain(Single=Refund) = E(Single) – [(1/4)*E(Yes)+(3/4)*E(No)]
E(Yes) = 0
E(No) = (-2/3)log(2/3)-(1/3)log(1/3)
=0.9183
Gain(Single=Refund) = 1 – [(1/3)*0 +(3/4)*0.9183]
= 0.31127
Gain(Single=Taxable income) = E(Single) – [(1/4)*E(<80)+(3/4)*E(>80)]
=0.31127
Marital status = Divorced : Entropy(Divorced) =1
Gain(Divorced=Refund) = E(Divorced) – [(1/2)*E(Yes) +(1/2)*E(No)]
E(Yes) = 0
E(No) = 0
Gain = 1
Gain(Divorced=Taxable income) = E(Divorced) – [0+ (2/2)*E(>80k)]
= 0

4. Explain gradient descent algorithm .Derive an equation of gradient descent rule to minimize the error.

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($training\_examples, \eta$)

Each training example is a pair of the form $\langle \vec{x}, t \rangle$, *where* $\vec{x}$ *is the vector of input values, and* t *is the target output value.* η *is the learning rate (e.g., .05).*

- Initialize each w_i to some small random value
- Until the termination condition is met, Do
  - Initialize each Δw_i to zero.
  - For each $\langle \vec{x}, t \rangle$ in *training_examples*, Do
    - Input the instance $\vec{x}$ to the unit and compute the output o
    - For each linear unit weight w_i, Do

$$\Delta w_i \leftarrow \Delta w_i + \eta(t - o)x_i$$

  - For each linear unit weight w_i, Do

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

The delta training rule is best understood by considering the task of training an ***unthresholded*** perceptron; that is, a ***linear unit*** for which the output ***o*** is given by

$$o(\vec{x}) = \vec{w} \cdot \vec{x}$$

Thus, a linear unit corresponds to the first stage of a perceptron, without the threshold.

In order to derive a weight learning rule for linear units, let us begin by specifying a measure for the ***training error*** of a hypothesis (weight vector), relative to the training examples. Although there are many ways to define this error, one common measure that will turn out to be especially convenient is

$$E(\vec{w}) \equiv \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$$

where D is the set of training examples, t_d is the target output for training example *d*, and o_d is the output of the linear unit for training example d.

By this definition, $E(\vec{w})$ is simply half the squared difference between the target output ***td*** and the hear unit output ***od,*** summed over all training examples. Here we characterize ***E*** as a function of $\vec{w}$ **,** because the linear unit output ***o*** depends on this weight vector. Of course ***E*** also depends on the particular set of training examples, but we assume these are fixed during training, so we do not bother to write E as an explicit function of these
. In particular, there we show that under certain conditions the hypothesis that minimizes E is also the most probable hypothesis in H given the training data.

DERIVATION OF THE GRADIENT DESCENT RULE

We can calculate the direction of steepest descent along the error surface.This direction 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 $\nabla E(\vec{w})$.

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

Notice $\nabla E(\vec{w})$is itself a vector, whose components are the partial derivatives of ***E*** with respect to each of the ***wi. When interpreted as a vector in weight space, the gradient specijies the direction that produces the steepest increase in E.*** The negative of this vector therefore gives the direction of steepest decrease.

Since 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

$$\Delta\vec{w} = -\eta \nabla E(\vec{w})$$

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

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i}$$ ----→ **(4.5)**

differentiating Error function E from

$$\frac{\partial E}{\partial w_i} = \frac{\partial}{\partial w_i} \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$$

$$= \frac{1}{2} \sum_{d \in D} \frac{\partial}{\partial w_i} (t_d - o_d)^2$$

$$= \frac{1}{2} \sum_{d \in D} 2(t_d - o_d) \frac{\partial}{\partial w_i} (t_d - o_d)$$

$$= \sum_{d \in D} (t_d - o_d) \frac{\partial}{\partial w_i} (t_d - \vec{w} \cdot \vec{x}_d)$$

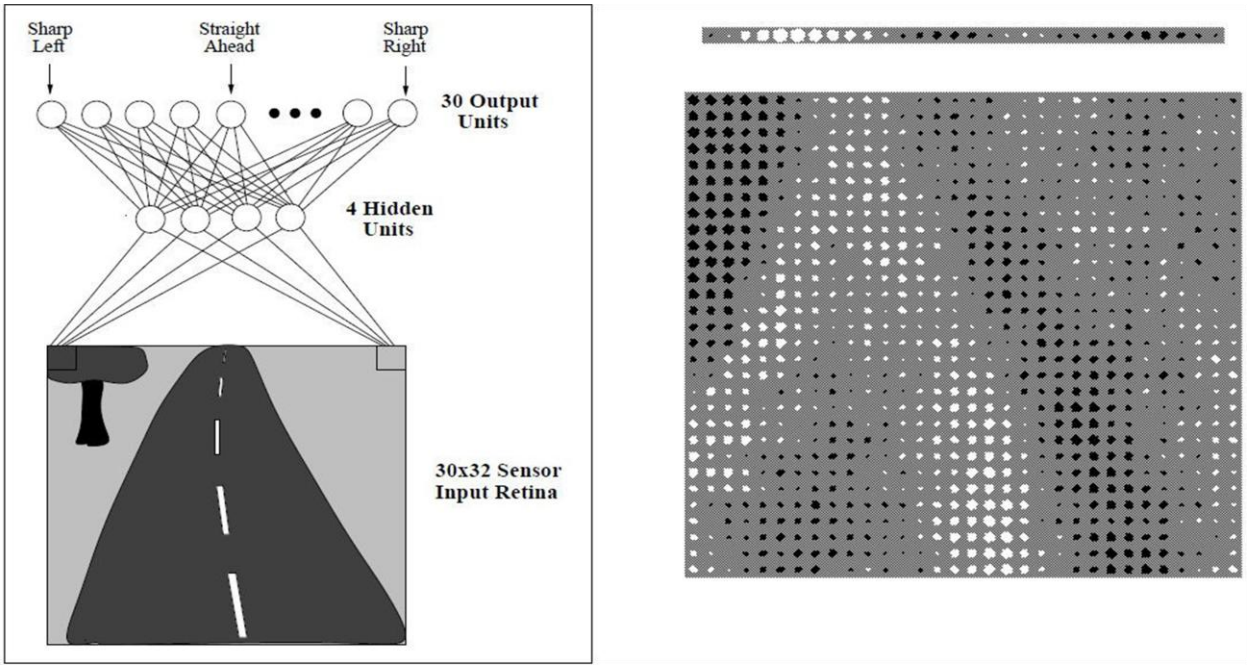
$$\frac{\partial E}{\partial w_i} = \sum_{d \in D} (t_d - o_d)(-x_{id}) \qquad \text{---→ (4.6)}$$

where x_{id} denotes the single input component x_i for training example d. We now have an equation that gives in terms of the linear unit inputs x_{id}, outputs O_d, and target values t_d associated with the training examples. Substituting Equation (4.6) into Equation (4.5) yields the weight update rule for gradient descent

$$\Delta w_i = \eta \sum_{d \in D} (t_d - o_d)\ x_{id}$$

5.a) Discuss the application of neural network which is used for learning to steer an autonomous vehicle

Artificial neural networks (ANNs) provide a general, practical method for learning real-valued, discrete-valued, and vector-valued target functions from examples.



- A prototypical example of ANN learning is provided by Pomerleau's (1993) system ALVINN, which uses a learned ANN to steer an autonomous vehicle driving at normal speeds on public highways.
- The ***input*** to the neural network is a 30x32 grid of pixel intensities obtained from a forward-pointed camera mounted on the vehicle.
- The network ***output*** is the direction in which the vehicle is steered.
- Figure illustrates the neural network representation. The network is shown on the left side of the figure, with the input camera image depicted below it.
- Each node (i.e., circle) in the network diagram corresponds to the output of a single network ***unit,*** and the lines entering the node from below are its *inputs.*
- There are four units that receive inputs directly from all of the 30 x 32 pixels in the image. These are called *"hidden" units* because their output is available only within the network and is not available as part of the global network output. Each of these four hidden units computes a single real-valued output based on a weighted combination of its 960 inputs
- These hidden unit outputs are then used as inputs to a second layer of 30 "output" units.
- Each output unit corresponds to a particular steering direction, and the output values of these units determine which steering direction is recommended most strongly.
- The diagrams on the right side of the figure depict the learned weight values associated with one of the four hidden units in this ANN.
- The large matrix of black and white boxes on the lower right depicts the weights from the 30 x 32 pixel inputs into the hidden unit. Here, a white box indicates a positive weight, a black box a negative weight, and the size of the box indicates the weight magnitude.
- The smaller rectangular diagram directly above the large matrix shows the weights from this hidden unit to each of the 30 output units.

5.b) Explain artificial neural network based on perception concept with diagram

The perceptron *is* the basic processing element. It has inputs that may come from the environment or may be the outputs of other perceptrons. Associated with each input, xj connection weight $x_j \in \Re$ $j = 1, \ldots, d$, is a *connection weight*, or *synaptic weight wj* synapti c weight and the output, $w_j \in \Re$, in the simplest case is a weighted sum of the inputs

$$y = \sum_{j=1}^{d} w_j x_j + w_0$$

*w*0 is the intercept value to make the model more general; it is generally modeled as the weight coming from an extra *bias unit.*

A perceptron takes a vector of real-valued inputs, calculates a linear combination of these inputs, then outputs a

1 if the result is greater than some threshold and -1 otherwise. More precisely, given inputs ***xl*** through ***x,,*** the output ***o(x1,*** . . . , ***x,)*** computed by the perceptron is

$$o(x_1, \ldots, x_n) = \begin{cases} 1 \text{ if } w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_n x_n > 0 \\ -1 \text{ otherwise} \end{cases}$$

where each ***wi*** is a real-valued constant, or weight, that determines the contribution of input ***xi*** to the perceptron output. Notice the quantity ***(-wO)*** is a threshold that the weighted combination of inputs ***w1 x1***+ . . . + ***wnxn*** must surpass in order for the perceptron to output a 1.To simplify notation, we imagine an additional constant input ***xo*** = 1, allowing us to write the above inequality as $\sum_{i=0}^{n} w_i x_i > 0$, or in vector form as $\vec{w} \cdot \vec{x} > 0.$ We will sometimes write the perceptron function as

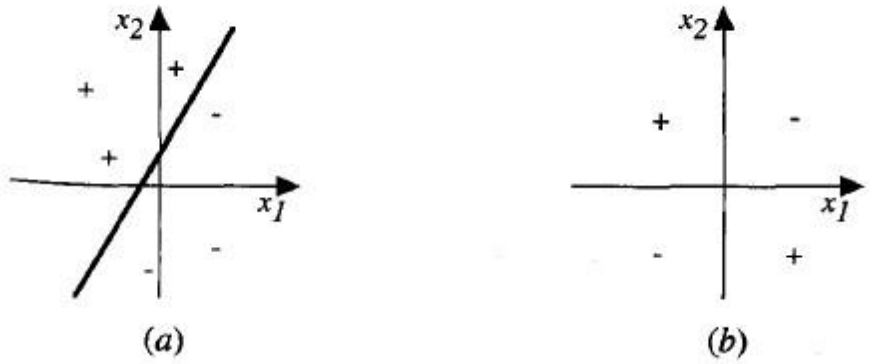
$$o(\vec{x}) = sgn(\vec{w} \cdot \vec{x})$$

where

$$sgn(y) = \begin{cases} 1 \text{ if } y > 0 \\ -1 \text{ otherwise} \end{cases}$$

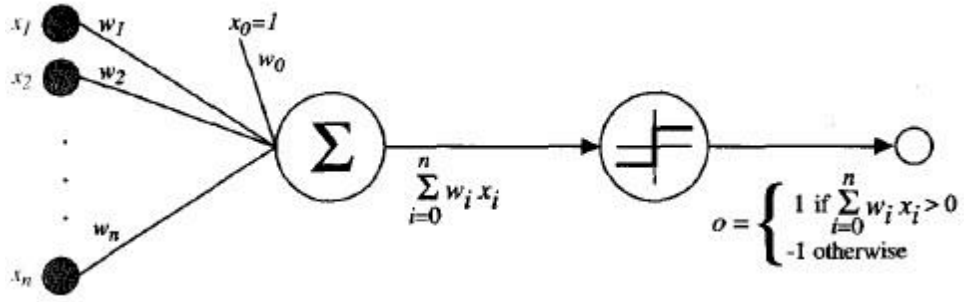
ii) Representational power of Perceptrons

We can view the perceptron as representing a hyperplane decision surface in the n-dimensional space of instances (i.e., points). The perceptron outputs a 1 for instances lying on one side of the hyperplane and outputs a -1 for instances lying on the other side, as illustrated in Figure. The equation for this decision hyperplane is $\vec{w} \cdot \vec{x} = 0$. Of course, some sets of positive and negative examples cannot be separated by any hyperplane. Those that can be separated are called linearly separable sets of examples.



a)A set of training examples that are linearly separable **b) A set of training examples that are not linearly separable**

A single perceptron can be used to represent many boolean functions. For example, if we assume boolean values of 1 (true) and -1 (false), then one way to use a two-input perceptron to implement the AND function is to set the weights w_0 = ***-0.3,*** and $w2 = w2$ =0 *.5*. This perceptron can be made to represent the OR function instead by altering the threshold to w_0 = -0.3. AND and OR can be viewed as special cases of m-of-n functions: that is, functions where at least m of the n inputs to the perceptron must be true. The OR function corresponds to m = 1 and the AND function to m = n. Any m-of-n function is easily represented using a perceptron by setting all input weights to the same value (e.g., 0.5) and then setting the threshold w_0 accordingly. Perceptrons can represent all of the primitive boolean functions AND, OR, NAND (1 AND), and NOR (1 OR). Unfortunately, however, some boolean functions cannot be represented by a single perceptron, such as the XOR function whose value is 1 if and only if **x1 ≠ x2.** Note the set of linearly non separable training examples corresponds to this XOR function.



A Perceptron

6. Write an algorithm for back propagation which uses stochastic gradient descent method. Derive the back propagation rule considering the output layer and training rule for output unit weights.

BACKPROPAGATION (*training_example*, η, n_{in}, n_{out}, n_{hidden})

Each training example is a pair of the form $(\vec{x}, \vec{t})$, *where* $(\vec{x})$ *is the vector of network input values,* $(\vec{t})$ *and is the vector of target network output values.*
η *is the learning rate (e.g., .05).* n_i *is the number of network inputs,* n_{hidden} *the number of units in the hidden layer, and* n_{out} *the number of output units.*
The input from unit i into unit j is denoted x_{ji}, *and the weight from unit i to unit j is denoted* w_{ji}

- Create a feed-forward network with n_i inputs, n_{hidden} hidden units, and n_{out} output units.
- Initialize all network weights to small random numbers
- Until the termination condition is met, Do

  - For each $(\vec{x}, \vec{t})$, 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

$$\delta_k \leftarrow o_k(1 - o_k)(t_k - o_k)$$

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

$$\delta_h \leftarrow o_h(1 - o_h) \sum_{k \in outputs} w_{h,k}\delta_k$$

4. Update each network weight w_{ji}

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

Where

$$\Delta w_{ji} = \eta \delta_j x_{i,j}$$

Derivation of the BACKPROPAGATION Rule

- **For each training example d every weight w_{ji} is updated by adding to it Δw_{ji}**

$$\Delta w_{ji} = -\eta \frac{\partial E_d}{\partial w_{ji}} \qquad \text{.....equ. (1)}$$

where, E_d is the error on training example d, summed over all output units in the network

$$E_d(\vec{w}) \equiv \frac{1}{2} \sum_{k \in output} (t_k - o_k)^2$$

By using chain rule $\frac{\partial E_d}{\partial net_j} x_{ji}$

Case 1: Training Rule for Output Unit Weights. Just as w_{ji} can influence the rest of the network only through net_j, net_j can influence the network only through o_j. Therefore, we can invoke the chain rule again to write

$$\frac{\partial E_d}{\partial net_j} = \frac{\partial E_d}{\partial o_j} \frac{\partial o_j}{\partial net_j} \tag{4.23}$$

To begin, consider just the first term in Equation (4.23)

$$\frac{\partial E_d}{\partial o_j} = \frac{\partial}{\partial o_j} \frac{1}{2} \sum_{k \in outputs} (t_k - o_k)^2$$

The derivatives $\frac{\partial}{\partial o_j}(t_k - o_k)^2$ will be zero for all output units k except when $k = j$. We therefore drop the summation over output units and simply set $k = j$.

$$\begin{aligned} \frac{\partial E_d}{\partial o_j} &= \frac{\partial}{\partial o_j} \frac{1}{2} (t_j - o_j)^2 \\ &= \frac{1}{2} 2(t_j - o_j) \frac{\partial (t_j - o_j)}{\partial o_j} \\ &= -(t_j - o_j) \end{aligned} \tag{4.24}$$

Next consider the second term in Equation (4.23). Since $o_j = \sigma(net_j)$, the derivative $\frac{\partial o_j}{\partial net_j}$ is just the derivative of the sigmoid function, which we have already noted is equal to $\sigma(net_j)(1 - \sigma(net_j))$. Therefore,

$$\frac{\partial o_j}{\partial net_j} = \frac{\partial \sigma(net_j)}{\partial net_j}$$
$$= o_j(1 - o_j) \tag{4.25}$$

Substituting expressions (4.24) and (4.25) into (4.23), we obtain

$$\frac{\partial E_d}{\partial net_j} = -(t_j - o_j)\ o_j(1 - o_j) \tag{4.26}$$

and combining this with Equations (4.21) and (4.22), we have the stochastic gradient descent rule for output units

$$\Delta w_{ji} = -\eta \frac{\partial E_d}{\partial w_{ji}} = \eta\ (t_j - o_j)\ o_j(1 - o_j) x_{ji} \tag{4.27}$$

Case 2: Training Rule for Hidden Unit Weights. In the case where j is an internal, or hidden unit in the network, the derivation of the training rule for w_{ji} must take into account the indirect ways in which w_{ji} can influence the network outputs and hence E_d. For this reason, we will find it useful to refer to the set of all units immediately downstream of unit j in the network (i.e., all units whose direct inputs include the output of unit j). We denote this set of units by $Downstream(j)$. Notice that net_j can influence the network outputs (and therefore E_d) only through the units in $Downstream(j)$. Therefore, we can write

$$\begin{aligned}
\frac{\partial E_d}{\partial net_j} &= \sum_{k \in Downstream(j)} \frac{\partial E_d}{\partial net_k} \frac{\partial net_k}{\partial net_j} \\
&= \sum_{k \in Downstream(j)} -\delta_k \frac{\partial net_k}{\partial net_j} \\
&= \sum_{k \in Downstream(j)} -\delta_k \frac{\partial net_k}{\partial o_j} \frac{\partial o_j}{\partial net_j} \\
&= \sum_{k \in Downstream(j)} -\delta_k\ w_{kj} \frac{\partial o_j}{\partial net_j} \\
&= \sum_{k \in Downstream(j)} -\delta_k\ w_{kj}\ o_j(1 - o_j)
\end{aligned} \tag{4.28}$$

Rearranging terms and using δ_j to denote $-\frac{\partial E_d}{\partial net_j}$, we have

$$\delta_j = o_j(1 - o_j) \sum_{k \in Downstream(j)} \delta_k \, w_{kj}$$

and

$$\Delta w_{ji} = \eta \, \delta_j \, x_{ji}$$

which is precisely the general rule from Equation (4.20) for updating internal unit weights in arbitrary acyclic directed graphs. Notice Equation (T4.4) from Table 4.2 is just a special case of this rule, in which $Downstream(j) = outputs$.

7.a) Consider two perceptrons defined by the threshold expression w0+w1x1+w2x2 >0.
Perceptron A has weight values w0=1, w1=2, w2=1.
Perceptron B has weight values w0=0, w1=2, w2=1.
True or False? Perceptron A has more general than perceptron B.

Solution:

We will say that h_j is (strictly) more-general than h_k (written $h_j >_g h_k$) if and only if $(h_j \geq_g h_k) \wedge (h_k \not\geq_g h_j)$. Finally, we will sometimes find the inverse useful and will say that ***hj*** is ***more specific than*** h_k when h_k is ***more_general-than*** h_j.

X_1	X_2	$w_0+w_1x_1+w_2x_2$ **Perceptron A**	$w_0+w_1x_1+w_2x_2$ **Perceptron B**	**A more general than B** ($A \geq_g B$)
0	0	1+2*0+1*0=1	0+2*0+1*0=0	1
0	1	1+2*0+1*1=2	0+2*0+1*1=1	1
1	0	1+2*1+1*0=3	0+2*1+1*0=2	1
1	1	1+2*1+1*1=4	0+2*1+1*1=3	1

B(<x1,x2>) = 1→ 2x1+x2 > 0 → 1+2x1+x2 > 0 → A(<x1,x2>)) = 1

True.

7.b)Explain the issues in decision tree learning

Solution:

1. **Avoiding Overfitting the Data**

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

Noisy Training Example

<Sunny, Hot, Normal, Strong, ->

- Example is noisy because the correct label is +
- Previously constructed tree misclassifies it

Approaches to avoiding overfitting in decision tree learning

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

1.1. Reduced-error pruning :

- ***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
- Nodes are removed only if the resulting pruned tree performs no worse than-the original over the validation set.
- 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

1.2. Rule Post-Pruning :

Rule post-pruning is successful method for finding high accuracy hypotheses

Rule post-pruning involves the following steps:

1. Infer the decision tree from the training set, growing the tree until the training data is fit as well as possible and allowing overfitting to occur.
2. Convert the learned tree into an equivalent set of rules by creating one rule for each path from the root node to a leaf node.

3. Prune (generalize) each rule by removing any preconditions that result in improving its estimated accuracy.
4. Sort the pruned rules by their estimated accuracy, and consider them in this sequence when classifying subsequent instances.

For example, consider the decision tree above. The leftmost path of the tree in below figure is translated into the rule.

IF (Outlook = Sunny) ^ (Humidity = High) THEN *PlayTennis* = No

Given the above rule, rule post-pruning would consider removing the preconditions (Outlook = Sunny) and (Humidity = High)

It would select whichever of these pruning steps produced the greatest improvement in estimated rule accuracy

2. Incorporating Continuous-Valued Attributes

There are two methods for Handling Continuous Attributes

1. 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}

2. 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

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

- 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. 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. Alternative Measures for Selecting Attributes

One Approach: Use *GainRatio* instead of *Gain*

- The gain ratio measure penalizes attributes by incorporating a split information, that is sensitive to how broadly and uniformly the attribute splits the data

$$GainRatio(S, A) \equiv \frac{Gain(S, A)}{SplitInformation(S, A)}$$

$$SplitInformation(S, A) \equiv -\sum_{i=1}^{c} \frac{|S_i|}{|S|} \log_2 \frac{|S_i|}{|S|}$$

- where S_i is subset of S, for which attribute A has value v_i

4. Handling Training Examples with Missing Attribute Values

The data which is available may contain missing values for some attributes

Example:

Outlook	**Temp**	**Humidity**	**Wind**	**PlayTennis**
Sunny	Hot	High	Light	No
Sunny	Hot	High	Strong	No
Sunny	Mild	???	Light	No

Sunny	Cool	Normal	Light	Yes
Sunny	Mild	Normal	Strong	Yes

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. In this case it is **high**

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 we might describe patients 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.
- In such tasks, we would prefer decision trees that use low-cost attributes where possible, relying on high-cost attributes only when needed to produce reliable classifications.

8) Create and explain the decision tree for the following transactions using ID3 algorithm.

Day	Outlook	Temperature	Humidity	Wind	Play Tennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rainy	Mild	High	Weak	Yes
D5	Rainy	Cool	Normal	Weak	Yes
D6	Rainy	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Weak	Yes
D8	Sunny	Mild	High	Weak	No

Entropy(S) = -(4/8)log (4/8)-(4/8)log (4/8)
= 1 (equal number of positive and negative instances in each group)

Info.gain(S,Outlook) = Entropy(S) – [(4/8)* Entropy(sunny) +(4/8) *Entropy(Rainy)]

Entropy (sunny) = -(2/4) log(2/4) – (2/4) log(2/4)
= 1 (equal number of positive and negative instances in each group)
Entropy (Rainy) = -(2/4) log(2/4) – (2/4) log(2/4)
= 1 (equal number of positive and negative instances in each group)

Hence
Info.gain(S,Outlook) = Entropy(S) – [(4/8)*1 +(4/8) *1]
= 1- [(4/8) +(4/8)]
= 1 -1
= 0
Info.gain(S,Temperature) = Entropy(S) – [(5/8)* Entropy(Hot) +(3/8) *Entropy(Cool)]

Entropy (Hot) = -(4/5) log(4/5) – (1/5) log(1/5)
= -0.2575 -0.4644
= - 0.7219
Entropy (Cool) = -(3/3) log(3/3) -0
= 0 (All instances are in same group)
Info.gain(S,Temperature) = Entropy(S) – [(5/8)*(-0.7219) +(3/8) *(0)]
= 1- 0.4512
= 0.55

Info.gain(S,Humidity) = Entropy(S) – [(5/8)* Entropy(High) +(3/8) *Entropy(Normal)]

Entropy (High) = -(4/5) log(4/5) – (1/5) log(1/5)
= -0.2575 - 0.4644
= - 0.7219
Entropy (Normal) = -(3/3) log(3/3) -0
= 0 (All instances are in same group)
Info.gain(S,Humidity) = Entropy(S) – [(5/8)*(-0.7219) +(3/8) *(0)]
= 1- 0.4512
= 0.55

Info.gain(S,Wind) = Entropy(S) – [(4/8)* Entropy(Strong) +(4/8) *Entropy(Weakl)]

Entropy (Strong) = -(4/4) log(4/4) – 0
= (All instances are in same group)

Entropy (Weak) = 0-(4/4) log(4/4)
= 0 (All instances are in same group)

Info.gain(S,Wind) = Entropy(S) – [(4/8)* 0 +(4/8) *0)]
= 1-0
=1
Info.gain(S,Outlook) = 0
Info.gain(S,Temperature) =0.55
Info.gain(S,Humidity) = 0.55
Info.gain(S,Wind) =1

Since information gain of Wind is more than other attribute,it is selected as **root node** of the decision tree.
Instances - D1,D2,D3,D4 are added as left subtree and D5,D6,D7,D8 are added as right subtree under Root node. Since all the subsets belongs to the same group, they are labeled as Yes and No respectively in the level -2 of the tree