VTU Solution – Jan/Feb 2023

Sub:	Artificial Intelligence & Machine Learning							Code:	18CS71
Date:	16/1/2023	Duration:	3 Hours	Max Marks:	100	Sem:	VII	Branch:	ISE

Q.n	Solutions	Marks
1a	The Water Jug Problem states: You are provided with two jugs, one with 4-gallons of capacity and the other with one with 3-gallons of capacity. Neither have any measuring markers on it. How can we get exactly two gallons of water in the 4-gallon jug • Write down the production rules for the above problem • Write any one solution to the above problem Problem is "You are given two jugs, a 4-litre one and a 3-litre one. One neither has any measuring markers on it. There is a pump that can be used to fill the jugs with water. How can you get exactly 2 litres of water into 4-litre jug?" Solution: The state space for the problem can be described as a set of states, where each state represents the number of gallons in each state. The game start with the initial state described as a set of ordered pairs of integers: • State: (x, y) – x = number of lts in 4 lts jug – y = number of lts in 3 lts jug x = 0, 1, 2, 3, or 4 y = 0, 1, 2, 3 • Start state: (0, 0) i.e., 4-litre and 3-litre jugs is empty initially. • Goal state: (2, n) for any n that is 4-litre jug has 2 litres of water and 3-litre jug has any value from 0-3 since it is not specified. • Attempting to end up in a goal state. Production Rules: These rules are used as operators to solve the problem. They are represented as rules whose left sides are used to describe new state that result from approaching the rule.	08

	1	(x, y) if $x < 4$	$\rightarrow$	$(4, y)$	Fill the 4-gallon jug
	2	(x, y) if $y < 3$	$\rightarrow$	$(x, 3)$	Fill the 3-gallon jug
	3	(x, y) if $x > 0$	$\rightarrow$	$(x - d, y)$	Pour some water out of the 4-gallon jug
	4	(x, y) if $y > 0$	$\rightarrow$	$(x, y - d)$	Pour some water out of the 3-gallon jug
	5	(x, y) if $x > 0$	$\rightarrow$	$(0, y)$	Empty the 4-gallon jug on the ground
	6	(x, y) if $y > 0$	$\rightarrow$	$(x, 0)$	Empty the 3-gallon jug on the ground
	7	(x, y) if $x + y \geq 4$ and $y > 0$	$\rightarrow$	$(4, y - (4 - x))$	Pour water from the 3-gallon jug into the 4-gallon jug until the 4-gallon jug is full
	8	(x, y) if $x + y \geq 3$ and $x > 0$	$\rightarrow$	$(x - (3 - y), 3)$	Pour water from the 4-gallon jug into the 3-gallon jug until the 3-gallon jug is full
	9	(x, y) if $x + y \leq 4$ and $y > 0$	$\rightarrow$	$(x + y, 0)$	Pour all the water from the 3-gallon jug into the 4-gallon jug
	10	(x, y) if $x + y \leq 3$ and $x > 0$	$\rightarrow$	$(0, x + y)$	Pour all the water from the 4-gallon jug into the 3-gallon jug
	11	$(0, 2)$	$\rightarrow$	$(2, 0)$	Pour the 2 gallons from the 3-gallon jug into the 4-gallon jug
	12	$(2, y)$	$\rightarrow$	$(0, y)$	Empty the 2 gallons in the 4-gallon jug on the ground

The solution to the water-jug problem is:

Gallons in the 4-Gallon Jug	Gallons in the 3-Gallon Jug	Rule Applied
0	0	
		2
0	3	
		9
3	0	
		2
3	3	
		7
4	2	
		5 or 12
0	2	
		9 or 11
2	0	

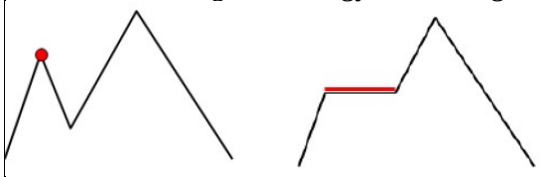
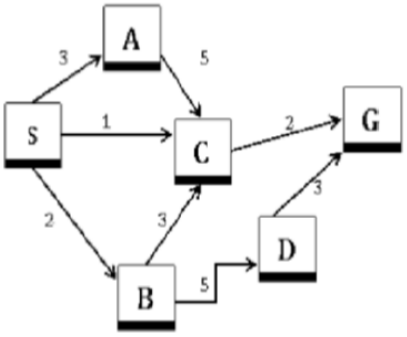
1 b. **Explain steepest Ascent Hill Climbing Technique with an algorithm. Comment on its drawback and how to overcome these drawbacks**

12

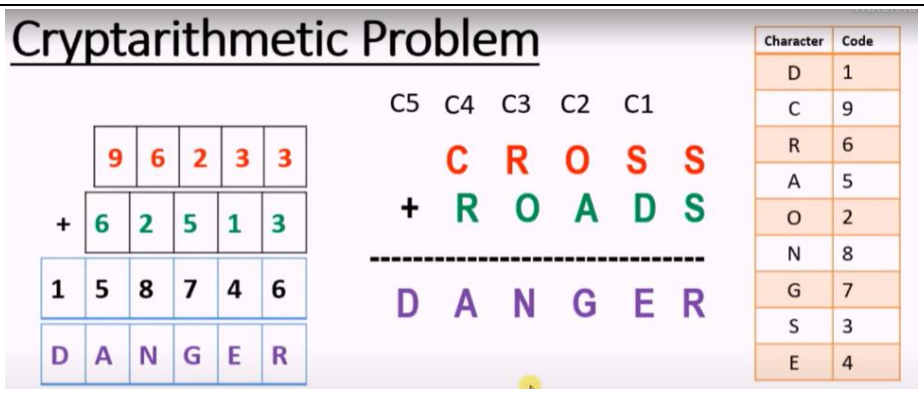
A useful variation on simple hill climbing considers all the moves from the current state and selects the best one as the next state. This method is called *steepest-ascent hill climbing* or *gradient search*

Algorithm:

1. Evaluate the initial state. If it is also a goal state then return it and quit. Otherwise continue with the initial state as the current state.
2. Loop until a solution is found or until a complete iteration produces no change to current state:

	a. Let SUCC be a state such that any possible successor of the current state will be better than SUCC. b. For each operator that applies to the current state do: i. Apply the operator and generate a new state. ii. Evaluate the new state. If it is a goal state, then return it and quit. If not compare it to SUCC. If it is better, then set SUCC to this state. If it is not better, leave SUCC alone. c. IF the SUCC is better than current state, then set current state to SUCC. Bothe basic and steepest-ascent hill climbing may fail to find a solution. Either algorithm may terminate not by finding a goal state but by getting a state from which no better states can be generated. This will happen if the program has reached a local maximum, a plateau or a ridge. A *local maximum* is a state that is better than all its neighbors but it not better than some other states farther away. At the local maximum, all moves appear to make things worse. Local maxima are particularly frustrating because they often occur almost within sight of a solution. In this case, they are called *foothills*. A *plateau* is a flat area of the search space in which a whole set of neighboring states has the same value. In this, it is not possible to determine the best direction in which to move by making local comparisons. A *ridge* is a special kind of maximum. It is an area of the search space that is higher than surrounding areas and that itself has a slope. There are some ways of dealing with these problems, although these methods are by no means guaranteed: □ Backtrack to some earlier node and try going in a different direction. This is particularly reasonable if at that node there was another direction that looked as promising or almost as promising as the one that was chosen earlier. This is a fairly good way to deal with local maxima. □ Make a big jump in some direction to try to get to a new section of the search space. This is a good way of dealing with plateaus. □ Apply two or more rules before doing the test. This corresponds to moving in several directions at once. This is a good strategy for dealing with ridges. 	
2a	Explain problem reduction with respect to AND-OR graph with suitable example. | State	h(n)
---	---
S	6
A	2
B	4
C	1
D	2
G	0 |  **AND-OR Graphs:** • Useful for representing the solution of problems that can be solved by **decomposing them into a set of smaller problems**, all of which must be then solved. Some problems are best represented as achieving sub goals, some of	7

	which achieved simultaneously and independently (AND) Up to now, only dealt with OR options **AO* Algorithm:** 1. Initialize the graph to start node 2. Traverse the graph following the current path accumulating nodes that have not yet been expanded or solved 3. Pick any of these nodes and expand it and if it has no successors call this value *FUTILITY* otherwise calculate only *f'* for each of the successors. 4. If *f'* is 0 then mark the node as *SOLVED* 5. Change the value of *f'* for the newly created node and let f' reflect on its predecessors by back propagation. 6. Wherever possible use the most promising routes and if all descendants of a node is marked as *SOLVED* then mark the parent node as *SOLVED*. If starting node is *SOLVED* or value greater than *FUTILITY*, stop, else repeat from 2. Current state is S • f(A)= 3+2 =5, f(B)= 2+4=6, f(C)=1+1=2 • Since, C is having smaller distance compared to other nodes, we have chosen current state as C. 2.Current State is C • f(C)= 2+0 = 2 Path = S -> C -> G	
2b	Write AO algorithm **AO* Algorithm:** 1. Initialize the graph to start node 2. Traverse the graph following the current path accumulating nodes that have not yet been expanded or solved 3. Pick any of these nodes and expand it and if it has no successors call this value *FUTILITY* otherwise calculate only *f'* for each of the successors. 4. If *f'* is 0 then mark the node as *SOLVED* 5. Change the value of *f'* for the newly created node and let f' reflect on its predecessors by back propagation. 6. Wherever possible use the most promising routes and if all descendants of a node is marked as *SOLVED* then mark the parent node as *SOLVED*. If starting node is *SOLVED* or value greater than *FUTILITY*, stop, else repeat from 2.	7
2c	Discuss about constraint satisfaction and solve the below crypt arithmetic problem. CROSS+ ROADS = DANGER	6

<table>
<tr>
<td></td>
<td>


</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>
Consider the following sentences:
1. John likes all kinds of food
2. Apples are food
3. Chicken is food
4. Anything anyone eats and isn't killed is food
5. Bill eats peanuts and is still alive
6. Sue eats every everything Bill eats.
(i) Translate these sentences into formulas in predicate logic.
(ii) Prove that John likes peanuts using backward chaining.
(iii)Convert the formulas of (i) into clause form.
(iv)Prove John likes peanuts using resolution.
Step-1: Conversion of Facts into FOL
In the first step we will convert all the given statements into its first order logic.
a. ∀x: food(x) → likes(John, x)
b. food(Apple) ∧ food(vegetables)
c. ∀x ∀y: eats(x, y) ∧ ¬ killed(x) → food(y)
d. eats (Anil, Peanuts) ∧ alive(Anil).
e. ∀x : eats(Anil, x) → eats(Harry, x)
f. ∀x: ¬ killed(x) → alive(x) } added predicates.
g. ∀x: alive(x) →¬ killed(x) }
h. likes(John, Peanuts)
Step-2: Conversion of FOL into CNF
In First order logic resolution, it is required to convert the FOL into CNF as CNF form makes easier for resolution proofs.
</td>
<td>5

5</td>
</tr>
</table>

- **Eliminate all implication (→) and rewrite**
  a. ∀x ¬ food(x) V likes(John, x)
  b. food(Apple) Λ food(vegetables)
  c. ∀x ∀y ¬ [eats(x, y) Λ ¬ killed(x)] V food(y)
  d. eats (Anil, Peanuts) Λ alive(Anil)
  e. ∀x ¬ eats(Anil, x) V eats(Harry, x)
  f. ∀x¬ [¬ killed(x)] V alive(x)
  g. ∀x ¬ alive(x) V ¬ killed(x)
  h. likes(John, Peanuts).

5

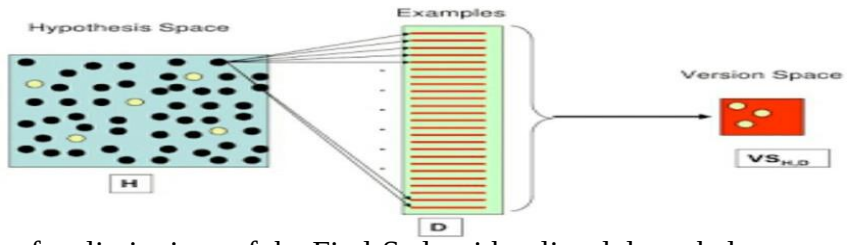
- **Eliminate all implication (→) and rewrite**
  a. ∀x ¬ food(x) V likes(John, x)
  b. food(Apple) Λ food(vegetables)
  c. ∀x ∀y ¬ [eats(x, y) Λ ¬ killed(x)] V food(y)
  d. eats (Anil, Peanuts) Λ alive(Anil)
  e. ∀x ¬ eats(Anil, x) V eats(Harry, x)
  f. ∀x¬ [¬ killed(x)] V alive(x)
  g. ∀x ¬ alive(x) V ¬ killed(x)
  h. likes(John, Peanuts).

- **Move negation (¬)inwards and rewrite**
  a. ∀x ¬ food(x) V likes(John, x)
  b. food(Apple) Λ food(vegetables)
  c. ∀x ∀y ¬ eats(x, y) V killed(x) V food(y)
  d. eats (Anil, Peanuts) Λ alive(Anil)
  e. ∀x ¬ eats(Anil, x) V eats(Harry, x)
  f. ∀x ¬killed(x)] V alive(x)
  g. ∀x ¬ alive(x) V ¬ killed(x)
  h. likes(John, Peanuts).

	◦ **Eliminate existential instantiation quantifier by elimination.** In this step, we will eliminate existential quantifier ∃, and this process is known as **Skolemization**. But in this example problem since there is no existential quantifier so all the statements will remain same in this step. ◦ **Drop Universal quantifiers.** In this step we will drop all universal quantifier since all the statements are not implicitly quantified so we don't need it. a. ¬ food(x) V likes(John, x) b. food(Apple) c. food(vegetables) d. ¬ eats(y, z) V killed(y) V food(z) e. eats (Anil, Peanuts) f. alive(Anil) g. ¬ eats(Anil, w) V eats(Harry, w) h. killed(g) V alive(g) i. ¬ alive(k) V ¬ killed(k) j. likes(John, Peanuts).	5
4a	Distinguish forward and backward reasoning with an example Definition of Forward Reasoning The solution of a problem generally includes the initial data and facts in order to arrive at the solution. These unknown facts and information is used to deduce the result. For example, while diagnosing a patient the doctor first check the symptoms and medical condition of the body such as temperature, blood pressure, pulse, eye colour, blood, etcetera. After that, the patient symptoms are analysed and compared against the predetermined symptoms. Then the doctor is able to provide the medicines according to the symptoms of the patient. So, when a solution employs this manner of reasoning, it is known as forward reasoning. Definition of Backward Reasoning The **backward reasoning** is inverse of forward reasoning in which goal is analysed in order to deduce the rules, initial facts and data. We can understand the concept by the similar example given in the above definition, where the doctor is trying to diagnose the patient with the help of the inceptive data such as symptoms. However, in this case, the patient is experiencing a problem in his body, on the basis of which the doctor is going to prove the symptoms.	04

	This kind of reasoning comes under backward reasoning.	
4b	Find maximally specific hypothesis for the training instances given below. Also write Find-S algorithm. The concept of this particular problem will be on what days does a person lines to go on walk Let's have a look at the algorithm of Find-S: 1. Initialize the value of the hypothesis for all attributes with the most specific one. That is, h0 = < ϕ, ϕ, ϕ, ϕ…….. > 2. Take the next example, if the taken example is negative leave them and move on to another example without changing our hypothesis for the step. 3. Now, if the taken example is a positive example, then For each attribute, check if the value of the attribute is equal to that of the value we took in our hypothesis. So now, the general hypothesis is: h0 = {'Morning', 'Sunny', 'Warm', 'Yes', 'Mild', 'Strong'} This is our general hypothesis, and now we will consider each example one by one, but only the positive examples. h1= {'Morning', 'Sunny', '?', 'Yes', '?', '?'} h2 = {'?', 'Sunny', '?', 'Yes', '?', '?'} We replaced all the different values in the general hypothesis to get a resultant hypothesis.	8
5a	Define version space. Discuss the limitations of finds algorithm over candidate elimination algorithm A representation of the set of all hypotheses which are consistent with D Definition: The version space, denoted VSH,D with respect to hypothesis space H and training examples D, is the subset of hypotheses from H consistent with the training examples in D VSH,D ≡{h ∈ H | Consistent(h, D)}	7

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



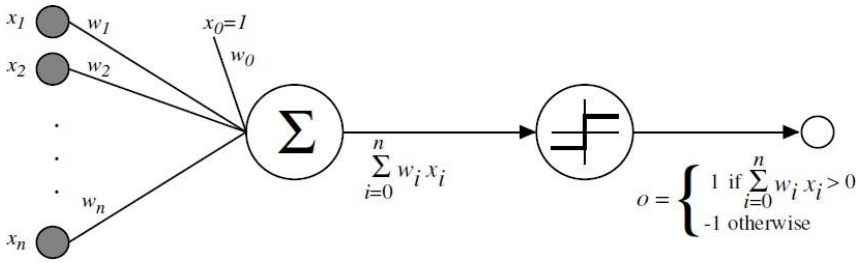
There are a few limitations of the Find-S algorithm listed down below:

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

5b

How a single perceptron can be used to represent the Boolean functions such as AND, OR.

- One type of ANN system is based on a unit called a perceptron. Perceptron is a single layer neural network.



- 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.
- Given inputs x through x, the output O(x1, . . . , xn) computed by the perceptron is

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

- Where, each wi is a real-valued constant, or weight, that determines the contribution of input xi to the perceptron output.
- -w0 is a threshold that the weighted combination of inputs w1x1 + . . . + wnxn must surpass in order for the perceptron to output a 1.

Sometimes, the perceptron function is written as,

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

Where,

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

6

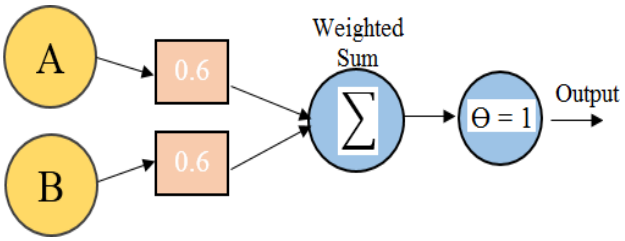
Learning a perceptron involves choosing values for the weights w0 , . . . , wn . Therefore, the space H of candidate hypotheses considered in perceptron learning is the set of all possible real-valued weight vectors

$$H = \{\vec{w} \mid \vec{w} \in \Re^{(n+1)}\}$$

Perceptrons can represent all of the primitive Boolean functions AND, OR, NAND (~ AND), and NOR (~OR)
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

Example: Representation of AND functions

A	B	A ^ B
0	0	0
0	1	0
1	0	0
1	1	1



If A=0 & B=0 → 0*0.6 + 0*0.6 = 0.
This is not greater than the threshold of 1, so the output = 0.
If A=0 & B=1 → 0*0.6 + 1*0.6 = 0.6.
This is not greater than the threshold, so the output = 0.
If A=1 & B=0 → 1*0.6 + 0*0.6 = 0.6.
This is not greater than the threshold, so the output = 0.
If A=1 & B=1 → 1*0.6 + 1*0.6 = 1.2.
This exceeds the threshold, so the output = 1.

5c

Write Gradient Descent algorithm to train a linear unit along with the derivation

Consider locally weighted regression in which the target function f is approximated near xq using a linear function of the form

$$\hat{f}(x) = w_0 + w_1 a_1(x) + \cdots + w_n a_n(x)$$

Where, ai(x) denotes the value of the ith attribute of the instance x

7

Derived methods are used to choose weights that minimize the squared error summed over the set D of training examples using gradient descent

$$E \equiv \frac{1}{2} \sum_{x \in D} (f(x) - \hat{f}(x))^2$$

Which led us to the gradient descent training rule

Where, η is a constant learning rate

$$\Delta w_j = \eta \sum_{x \in D} (f(x) - \hat{f}(x)) a_j(x)$$

- Need to modify this procedure to derive a local approximation rather than a global one. The simple way is to redefine the error criterion E to emphasize fitting the local training examples. Three possible criteria are given below.

1. Minimize the squared error over just the k nearest neighbors:

$$E_1(x_q) \equiv \frac{1}{2} \sum_{x \in k \text{ nearest nbrs of } x_q} (f(x) - \hat{f}(x))^2 \qquad \textbf{equ(1)}$$

2. Minimize the squared error over the entire set D of training examples, while weighting the error of each training example by some decreasing function K of its distance from x_q :

$$E_2(x_q) \equiv \frac{1}{2} \sum_{x \in D} (f(x) - \hat{f}(x))^2 \; K(d(x_q, x)) \qquad \textbf{equ(2)}$$

3. Combine 1 and 2:

$$E_3(x_q) \equiv \frac{1}{2} \sum_{x \in k \text{ nearest nbrs of } x_q} (f(x) - \hat{f}(x))^2 \; K(d(x_q, x)) \qquad \textbf{equ(3)}$$

1. Combine 1 and 2:

$$E_3(x_q) \equiv \frac{1}{2} \sum_{x \in k \text{ nearest nbrs of } x_q} (f(x) - \hat{f}(x))^2 \; K(d(x_q, x)) \qquad \textbf{equ(3)}$$

If we choose criterion three and re-derive the gradient descent rule, we obtain the following training rule

$$\Delta w_j = \eta \sum_{x \in k \text{ nearest nbrs of } x_q} K(d(x_q, x)) \; (f(x) - \hat{f}(x)) \; a_j(x)$$

The differences between this new rule and the rule given by Equation (3) are that the contribution of instance x to the weight update is now multiplied by the distance penalty **K(d(xq, x)),** and that the error is summed over only the k nearest training examples

6a	What do you mean by Gain and entropy? How it is used to build the decision tree Information Gain:	8

- Information gain is the expected reduction in entropy caused by partitioning the examples on an attribute.
- The higher the information gain the more effective the attribute in classifying training data.
- Expected reduction in entropy knowing A

Gain(S, A) = Entropy(S) − Σ Entropy(Sv)
$v \in$ Values(A)

Values(A) possible values for A
Sv subset of S for which A has value v

Entropy:

- Entropy measures the impurity of a collection of examples. It depends from the distribution of the random variable p.
  - S is a collection of training examples
  - p+ the proportion of positive examples in S
  - p– the proportion of negative examples in S

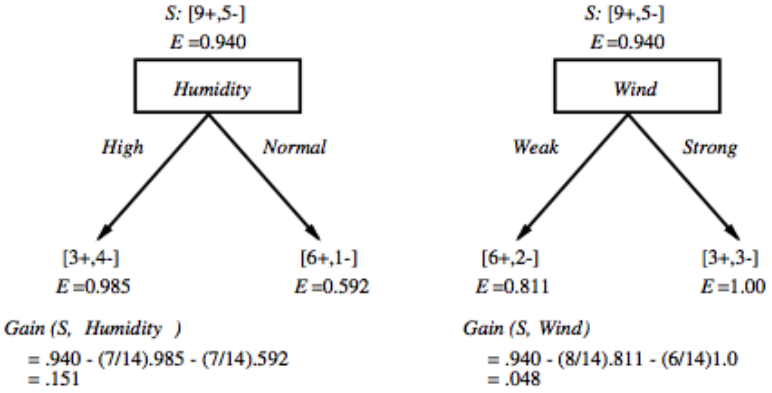
Entropy (S) ≡ – p+ log2 p+ – p–log2 p– [0 log20 = 0]
Entropy ([14+, 0–]) = – 14/14 log2 (14/14) – 0 log2 (0) = 0
Entropy ([9+, 5–]) = – 9/14 log2 (9/14) – 5/14 log2 (5/14) = 0,94
Entropy ([7+, 7–]) = – 7/14 log2 (7/14) – 7/14 log2 (7/14) =
= 1/2 + 1/2 = 1 [log21/2 = – 1]
Note: the log of a number < 1 is negative, $0 \leq p \leq 1$, $0 \leq$ entropy ≤ 1

Example:

- Let
  - Values(Wind) = {Weak, Strong}
  - S = [9+, 5−]
  - SWeak = [6+, 2−]
  - SStrong = [3+, 3−]
- Information gain due to knowing Wind:

Gain(S, Wind) = Entropy(S) − 8/14 Entropy(SWeak) − 6/14 Entropy(SStrong)
= 0,94 − 8/14 × 0,811 − 6/14 × 1,00
= 0,048

Which attribute is the best classifier?



Gain (S, Humidity)
= .940 - (7/14).985 - (7/14).592
= .151

Gain (S, Wind)
= .940 - (8/14).811 - (6/14)1.0
= .048

6b

Explain Back propagation algorithm? Why is it not likely to be trapped in local minima? 8

Algorithm:

Initialize all weights to small random numbers.
Until satisfied, Do

- For each training example, Do
  1. Input the training example to the network and compute the network outputs
  2. For each output unit k

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

  3. For each hidden unit h

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

  4. Update each network weight $w_{i,j}$

$$w_{i,j} \leftarrow w_{i,j} + \Delta w_{i,j}$$

where

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

It is not likely to be trapped in a local minima because of the following reasoning:

	Gradient descent to some local minimum • Perhaps not global minimum... • Add momentum • Stochastic gradient descent • Train multiple nets with different inital weights Nature of convergence • Initialize weights near zero • Therefore, initial networks near-linear • Increasingly non-linear functions possible as training progresses	
6c	**Discuss the perceptron training rule and delta rule that solves the learning problem of perceptron** ***Perceptron training rule:*** $w_i \leftarrow w_i + \Delta w_i$ Where $\Delta w_i = \eta(t - o)x_i$ Where η – is the learning rate t – is the target O/P o – is the output generated by the perceptron x_i – is the input ***Delta rule:*** $E(\vec{w}) \equiv \frac{1}{2}\sum_{d \in D}(t_d - o_d)^2$ Where D – is the training samples td – is the target O/P for the training examples o_d – is the O/P of the linear unit for training examples $E(w)$ – is half the squared difference between the target O/P, td and the linear unit output, o_d	4
7a	**Explain Naïve Bayes Classifier** The naive Bayes classifier applies to learning tasks where each instance x is described by a conjunction of attribute values and where the target function f (x) can take on any value from some finite set V. A set of training examples of the target function is provided, and a new instance is presented, described by the tuple of attribute values (al, a2.. .am). The learner is asked to predict the target value, or classification, for this new instance.	8

The Bayesian approach to classifying the new instance is to assign the most probable target value, VMAP, given the attribute values (al, a2.. .am) that describe the instance

The naive Bayes classifier is based on the assumption that the attribute values are conditionally independent given the target value. Means, the assumption is that given the target value of the instance, the probability of observing the conjunction (al, a2.. .am), is just the product of the probabilities for the individual attributes:

Naive Bayes classifier:

Where, VNB denotes the target value output by the naive Bayes classifier

An Illustrative Example

Let us apply the naive Bayes classifier to a concept learning problem i.e., classifying days according to whether someone will play tennis.

The below table provides a set of 14 training examples of the target concept PlayTennis, where each day is described by the attributes Outlook, Temperature, Humidity, and Wind

Use the naive Bayes classifier and the training data from this table to classify the following novel instance:

< Outlook = sunny, Temperature = cool, Humidity = high, Wind = strong > · Our task is to predict the target value (yes or no) of the target concept PlayTennis for this new instance

The probabilities of the different target values can easily be estimated based on their frequencies over the 14 training examples

· P(P1ayTennis = yes) = 9/14 = 0.64

· P(P1ayTennis = no) = 5/14 = 0.36

Similarly, estimate the conditional probabilities. For example, those for Wind = strong

· P(Wind = strong | PlayTennis = yes) = 3/9 = 0.33

· P(Wind = strong | PlayTennis = no) = 3/5 = 0.60

Thus, the naive Bayes classifier assigns the target value PlayTennis = no to this new instance, based on the probability estimates learned from the training data.

By normalizing the above quantities to sum to one, calculate the conditional probability that the target value is no, given the observed attribute values

ay	Outlook	Temperature	Humidity	Wind	PlayTennis
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	

7b **Explain Bayesian Belief Networks and conditional independence with an example.** 8

BAYESIAN BELIEF NETWORKS

The naive Bayes classifier makes significant use of the assumption that the values of the attributes a1 . . .an are conditionally independent given the target value v.

This assumption dramatically reduces the complexity of learning the target function A Bayesian belief network describes the probability distribution governing a set of variables by specifying a set of conditional independence assumptions along with a set of conditional probabilities

Bayesian belief networks allow stating conditional independence assumptions that apply to subsets of the variables

Representation

A Bayesian belief network represents the joint probability distribution for a set of variables. Bayesian networks (BN) are represented by directed acyclic graphs.

The Bayesian network in above figure represents the joint probability distribution over the boolean variables *Storm, Lightning, Thunder, ForestFire, Campfire,* and *BusTourGroup*

A Bayesian network (BN) represents the joint probability distribution by specifying a

	set of *conditional independence assumptions* BN represented by a directed acyclic graph, together with sets of local conditional probabilities Each variable in the joint space is represented by a node in the Bayesian network The network arcs represent the assertion that the variable is conditionally independent of its non-descendants in the network given its immediate predecessors in the network. A ***conditional probability table*** **(CPT)** is given for each variable, describing the probability distribution for that variable given the values of its immediate predecessors The joint probability for any desired assignment of values (y1, . . . , yn) to the tuple of network variables (Y1 . . . Ym) can be computed by the formula Where, Parents(Yi) denotes the set of immediate predecessors of Yi in the network. **Example:** Consider the node ***Campfire***. The network nodes and arcs represent the assertion that ***Campfire*** is conditionally independent of its non-descendants ***Lightning*** and ***Thunder***, given its immediate parents Storm and ***BusTourGroup***. This means that once we know the value of the variables ***Storm*** and ***BusTourGroup***, the variables ***Lightning*** and ***Thunder*** provide no additional information about ***Campfire*** The conditional probability table associated with the variable ***Campfire.*** The assertion is P(Campfire = True | Storm = True, BusTourGroup = True) = 0.4	
7c	**Let us assume dangerous fires are rare (1%) but smoke is fairly common (10%) due to the barbecues, and 90% of dangerous fires make smokes. Find the probability of dangerous fire when there is smoke.** Let us say P(Fire) means how often there is fire, and P(Smoke) means how often we see smoke, then: P (Fire | Smoke) means how often there is fire when we can see smoke P(Smoke | Fire) means how often we can see smoke when there fire Given, dangerous fires are rare (1%) i.e. P(Fire) but smoke is fairly common (10%) due to barbecues i.e. P(Smoke) and 90% of dangerous fires make smoke i.e. P(Smoke | Fire) Probability of dangerous fire when there is smoke: P(Fire | Smoke) = P(Fire) x P(Smoke | Fire) / P(Smoke) = 1 % x 90% / 10% = 9%	4

8a **Discuss minimum description length principle in brief.** 8

The Minimum Description Length principle is motivated by interpreting the definition of h_{MAP} in the light of basic concepts from information theory. Consider again the now familiar definition of h_{MAP}.

$$h_{MAP} = \underset{h \in H}{\operatorname{argmax}} P(D|h)P(h)$$

which can be equivalently expressed in terms of maximizing the $\log_2$

$$h_{MAP} = \underset{h \in H}{\operatorname{argmax}} \log_2 P(D|h) + \log_2 P(h)$$

or alternatively, minimizing the negative of this quantity

$$h_{MAP} = \underset{h \in H}{\operatorname{argmin}} -\log_2 P(D|h) - \log_2 P(h) \quad (6.16)$$

Somewhat surprisingly, Equation (6.16) can be interpreted as a statement that short hypotheses are preferred, assuming a particular representation scheme for encoding hypotheses and data. To explain this, let us introduce a basic result from information theory: Consider the problem of designing a code to transmit messages drawn at random, where the probability of encountering message i is p_i. We are interested here in the most compact code; that is, we are interested in the code that minimizes the expected number of bits we must transmit in order to encode a message drawn at random. Clearly, to minimize the expected code length we should assign shorter codes to messages that are more probable. Shannon and Weaver (1949) showed that the optimal code (i.e., the code that minimizes the expected message length) assigns $-\log_2 p_i$ bits† to encode message i. We will refer to the number of bits required to encode message i using code C as the *description length of message i with respect to C*, which we denote by $L_C(i)$.

Let us interpret Equation (6.16) in light of the above result from coding theory.

- $-\log_2 P(h)$ is the description length of h under the optimal encoding for the hypothesis space H. In other words, this is the size of the description of hypothesis h using this optimal representation. In our notation, $L_{C_H}(h) = -\log_2 P(h)$, where C_H is the optimal code for hypothesis space H.
- $-\log_2 P(D|h)$ is the description length of the training data D given hypothesis h, under its optimal encoding. In our notation, $L_{C_{D|h}}(D|h) = -\log_2 P(D|h)$, where $C_{D|h}$ is the optimal code for describing data D assuming that both the sender and receiver know the hypothesis h.
- Therefore we can rewrite Equation (6.16) to show that h_{MAP} is the hypothesis h that minimizes the sum given by the description length of the hypothesis plus the description length of the data given the hypothesis.

$$h_{MAP} = \underset{h}{\operatorname{argmin}} L_{C_H}(h) + L_{C_{D|h}}(D|h)$$

where C_H and $C_{D|h}$ are the optimal encodings for H and for D given h, respectively.

The Minimum Description Length (MDL) principle recommends choosing the hypothesis that minimizes the sum of these two description lengths. Of course to apply this principle in practice we must choose specific encodings or representations appropriate for the given learning task. Assuming we use the codes C_1 and C_2 to represent the hypothesis and the data given the hypothesis, we can state the MDL principle as

Minimum Description Length principle: Choose h_{MDL} where

$$h_{MDL} = \underset{h \in H}{\operatorname{argmin}} L_{C_1}(h) + L_{C_2}(D|h) \quad (6.17)$$

	The above analysis shows that if we choose C_1 to be the optimal encoding of hypotheses C_H, and if we choose C_2 to be the optimal encoding $C_{D\|h}$, then $h_{MDL} = h_{MAP}$.	
8b	**Explain brute force MAP learning algorithm.** In many learning scenarios, the learner considers some set ∈of candidate hypotheses H and is interested in finding the most probable hypothesis h H given the observed data D. Any such maximally probable hypothesis is called a maximum a posteriori (MAP) hypothesis. Bayes theorem to calculate the posterior probability of each candidate hypothesis is hMAP is a MAP hypothesis provided · P(D) can be dropped, because it is a constant independent of h Maximum Likelihood (ML) Hypothesis · In some cases, it is assumed that every hypothesis in H is equally probable a priori (P(hi) = P(hj) for all hi and hj in H). · In this case the below equation can be simplified and need only consider the term P(D|h) to find the most probable hypothesis. P(D|h) is often called the likelihood of the data D given h, and any hypothesis that maximizes P(D|h) is called a maximum likelihood (ML) hypothesis Example: Consider a medical diagnosis problem in which there are two alternative hypotheses: (1) that the patient has particular form of cancer, and (2) that the patient does not. The available data is from a particular laboratory test with two possible outcomes: + (positive) and - (negative). We have prior knowledge that over the entire population of people only .008 have this disease. Furthermore, the lab test is only an imperfect indicator of the disease. The test returns a correct positive result in only 98% of the cases in which the disease is actually present and a correct negative result in only 97% of the cases in which the disease is not present. In other cases, the test returns the opposite result. The above situation can be summarized by the following probabilities: Suppose a new patient is observed for whom the lab test returns a positive (+) result. Should we diagnose the patient as having cancer or not? The exact posterior probabilities can also be determined by normalizing the above quantities so that they sum to 1	8
8c	**Explain EM algorithm.**	4

Step 1: Calculate the expected value $E[z_{ij}]$ of each hidden variable z_{ij}, assuming the current hypothesis $h = \langle \mu_1, \mu_2 \rangle$ holds.

Step 2: Calculate a new maximum likelihood hypothesis $h' = \langle \mu'_1, \mu'_2 \rangle$, assuming the value taken on by each hidden variable z_{ij} is its expected value $E[z_{ij}]$ calculated in Step 1. Then replace the hypothesis $h = \langle \mu_1, \mu_2 \rangle$ by the new hypothesis $h' = \langle \mu'_1, \mu'_2 \rangle$ and iterate.

Let us examine how both of these steps can be implemented in practice. Step 1 must calculate the expected value of each z_{ij}. This $E[z_{ij}]$ is just the probability that instance x_i was generated by the jth Normal distribution

$$E[z_{ij}] = \frac{p(x = x_i | \mu = \mu_j)}{\sum_{n=1}^{2} p(x = x_i | \mu = \mu_n)}$$

$$= \frac{e^{-\frac{1}{2\sigma^2}(x_i - \mu_j)^2}}{\sum_{n=1}^{2} e^{-\frac{1}{2\sigma^2}(x_i - \mu_n)^2}}$$

Thus the first step is implemented by substituting the current values $\langle \mu_1, \mu_2 \rangle$ and the observed x_i into the above expression.

In the second step we use the $E[z_{ij}]$ calculated during Step 1 to derive a new maximum likelihood hypothesis $h' = \langle \mu'_1, \mu'_2 \rangle$.
maximum likelihood hypothesis in this case is given by

$$\mu_j \leftarrow \frac{\sum_{i=1}^{m} E[z_{ij}] \; x_i}{\sum_{i=1}^{m} E[z_{ij}]}$$

9a Explain k- Nearest Neighbor learning algorithm 8

The k- Nearest Neighbor algorithm for approximation a discrete-valued target function is

given below:

Training algorithm:

- For each training example $\langle x, f(x) \rangle$, add the example to the list *training_examples*

Classification algorithm:

- Given a query instance x_q to be classified,
  - Let $x_1 \ldots x_k$ denote the k instances from *training_examples* that are nearest to x_q
  - Return

$$\hat{f}(x_q) \leftarrow \underset{v \in V}{\operatorname{argmax}} \sum_{i=1}^{k} \delta(v, f(x_i))$$

where $\delta(a, b) = 1$ if $a = b$ and where $\delta(a, b) = 0$ otherwise.

The K- Nearest Neighbor algorithm for approximation a real-valued target function is given below $f : \Re^n \rightarrow \Re$

Training algorithm:

- For each training example $\langle x, f(x)\rangle$, add the example to the list *training_examples*

Classification algorithm:

- Given a query instance x_q to be classified,
  - Let $x_1 \ldots x_k$ denote the k instances from *training_examples* that are nearest to x_q
  - Return

$$\hat{f}(x_q) \leftarrow \frac{\sum_{i=1}^{k} f(x_i)}{k}$$

9b

- Consider locally weighted regression in which the target function ***f*** is approximated near xq using a linear function of the form

$$\hat{f}(x) = w_0 + w_1 a_1(x) + \cdots + w_n a_n(x)$$

Where, $a_i(x)$ denotes the value of the i^{th} attribute of the instance x

- Derived methods are used to choose weights that minimize the squared error summed over the set D of training examples using gradient descent

$$E \equiv \frac{1}{2} \sum_{x \in D} (f(x) - \hat{f}(x))^2$$

Which led us to the gradient descent training rule

$$\Delta w_j = \eta \sum_{x \in D} (f(x) - \hat{f}(x)) a_j(x)$$

Where, η is a constant learning rate

- Need to modify this procedure to derive a local approximation rather than a global one. The simple way is to redefine the error criterion E to emphasize fitting the local training examples. Three possible criteria are given below.

- Minimize the squared error over just the k nearest neighbors:

$$E_1(x_q) \equiv \frac{1}{2} \sum_{x \in k \text{ nearest nbrs of } x_q} (f(x) - \hat{f}(x))^2 \qquad \textbf{equ(1)}$$

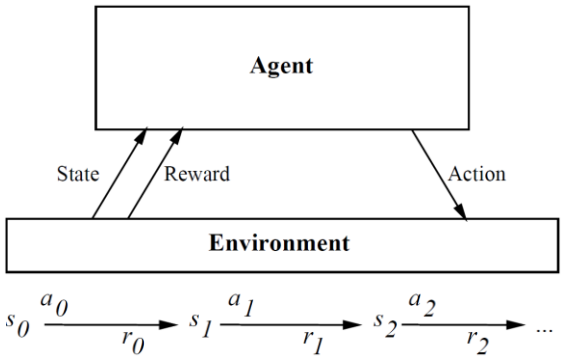
- Minimize the squared error over the entire set D of training examples, while weighting the error of each training example by some decreasing function K of its distance from xq :

$$E_2(x_q) \equiv \frac{1}{2} \sum_{x \in D} (f(x) - \hat{f}(x))^2 \; K(d(x_q, x)) \qquad \textbf{equ(2)}$$

- Combine 1 and 2:

$$E_3(x_q) \equiv \frac{1}{2} \sum_{x \in k \text{ nearest nbrs of } x_q} (f(x) - \hat{f}(x))^2 \; K(d(x_q, x)) \qquad \textbf{equ(3)}$$

8

	If we choose criterion three and re-derive the gradient descent rule, we obtain the following training rule $\Delta w_j = \eta \sum_{x \in k \text{ nearest nbrs of } x_q} K(d(x_q, x))\ (f(x) - \hat{f}(x))\ a_j(x)$ The differences between this new rule and the rule given by Equation (3) are that the contribution of instance x to the weight update is now multiplied by the distance penalty **K(d(xq, x)),** and that the error is summed over only the k nearest training examples.	
9c	What is Reinforcement learning? Reinforcement learning is an area of Machine Learning. It is about taking suitable action to maximize reward in a particular situation. It is employed by various software and machines to find the best possible behaviour or path it should take in a specific situation. Reinforcement learning differs from the supervised learning in a way that in supervised learning the training data has the answer key with it so the model is trained with the correct answer itself whereas in reinforcement learning, there is no answer but the reinforcement agent decides what to do to perform the given task. In the absence of training dataset, it is bound to learn from its experience.  Goal: Learn to choose actions that maximize $r_0 + \gamma r_1 + \gamma^2 r_2 + \ldots$, where $0 \leq \gamma < 1$	4
10a	Distinguish Eager learning vs Lazy learning algorithms. Eager learning method constructs general explicit description of the target function based on the provided training examples. Eg. Decision Tree based Learning Lazy learning method simply store the data and generalizes beyond these post point until an explicit request is made. Eg. KNN algorithm	4
10 b	Write short notes on Q-learning **The *Q* Function**	8

	The value of Evaluation function ***Q(s, a)*** is the reward received immediately upon executing action a from state ***s***, plus the value (discounted by γ) of following the optimal policy thereafter $$Q(s,a) \equiv r(s,a) + \gamma V^*(\delta(s,a))$$ **equ (4)** Rewrite Equation (3) in terms of ***Q(s, a)*** as $$\pi^*(s) = \underset{a}{\operatorname{argmax}}\, Q(s,a)$$ **equ (5)** Equation (5) makes clear, it need only consider each available action ***a*** in its current state ***s*** and choose the action that maximizes ***Q(s, a)***. **An Algorithm for Learning *Q*** • Learning the Q function corresponds to learning the **optimal policy**. • The key problem is finding a reliable way to estimate training values for ***Q***, given only a sequence of immediate rewards ***r*** spread out over time. This can be accomplished through ***iterative approximation*** $$V^*(s) = \max_{a'} Q(s,a')$$ Rewriting Equation $$Q(s,a) = r(s,a) + \gamma \max_{a'} Q(\delta(s,a), a')$$ --- *Q* learning algorithm For each s, a initialize the table entry $\hat{Q}(s,a)$ to zero. Observe the current state s Do forever: • Select an action a and execute it • Receive immediate reward r • Observe the new state s' • Update the table entry for $\hat{Q}(s,a)$ as follows: $$\hat{Q}(s,a) \leftarrow r + \gamma \max_{a'} \hat{Q}(s',a')$$ • $s \leftarrow s'$ ---	
10c	Discuss about Radial basis function in detail • One approach to function approximation that is closely related to distance-weighted regression and also to artificial neural networks is learning with radial basis functions	8

- In this approach, the learned hypothesis is a function of the form

$$\hat{f}(x) = w_0 + \sum_{u=1}^{k} w_u K_u(d(x_u, x)) \qquad \textbf{equ (1)}$$

- Where, each x_u is an instance from X and where the kernel function $K_u(d(x_u, x))$ is defined so that it decreases as the distance $d(x_u, x)$ increases.
- Here k is a user provided constant that specifies the number of kernel functions to be included.
- $\hat{f}$ is a global approximation to f (x), the contribution from each of the $K_u(d(x_u, x))$ terms is localized to a region nearby the point x_u.

Choose each function $K_u(d(x_u, x))$ to be a Gaussian function centred at the point x_u with some variance σ_u^2

$$K_u(d(x_u, x)) = e^{\frac{1}{2\sigma_u^2} d^2(x_u, x)}$$

- The functional form of equ(1) can approximate any function with arbitrarily small error, provided a sufficiently large number k of such Gaussian kernels and provided the width
  σ^2 of each kernel can be separately specified
- The function given by equ(1) can be viewed as describing a two layer network where the first layer of units computes the values of the various $K_u(d(x_u, x))$ and where the second layer computes a linear combination of these first-layer unit values

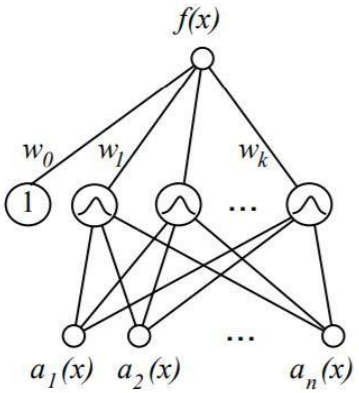
Example: Radial basis function (RBF) network

Given a set of training examples of the target function, RBF networks are typically trained in a two-stage process.

1. First, the number k of hidden units is determined and each hidden unit u is defined by choosing the values of x_u and σ_u^2 that define its kernel function $K_u(d(x_u, x))$
2. Second, the weights w, are trained to maximize the fit of the network to the training data, using the global error criterion given by

$$E \equiv \frac{1}{2} \sum_{x \in D} (f(x) - \hat{f}(x))^2$$

   Because the kernel functions are held fixed during this second stage, the linear weight values w, can be trained very efficiently



Several alternative methods have been proposed for choosing an appropriate number of hidden units or, equivalently, kernel functions.

- One approach is to allocate a Gaussian kernel function for each training example (x_i,f (x_i)), centring this Gaussian at the point x_i.
  Each of these kernels may be assigned the same width σ^2. Given this approach, the RBF network learns a global approximation to the target function in which each training example (x_i, f (x_i)) can influence the value of $\boldsymbol{f}$ only in the neighbourhood of x_i.
- A second approach is to choose a set of kernel functions that is smaller than the number of training examples. This approach can be much more efficient than the first approach, especially when the number of training examples is large.