Scheme of Evaluation
Internal Assessment Test 2 – Dec.2022

Sub:	Artificial Intelligence & Machine Learning							**Code:**	18CS71
Date**:**	01/12/2022	Duration:	90mins	Max Marks:	50	**Sem:**	VII	**Branch:**	ISE

Note: Answer Any Five Questions

Question #	**Description**	**Marks Distribution**	**Max Marks**
1	• Back propagation algorithm • Deriving the derivatives rule	4 M 6 M	10 M
2	• Finding the overall probabilities • Applying Naïve Bayes to calculate the conditional probabilities • Predicting the result	2 M 6M 2M	10 M
3	• Explaining KNN algorithm for discrete values • Pseudo Code	6M 4M	10 M
4	• Finding the euclidean distance • Updating the centroid • Applications of k-Means clustering	4M 4M 2M	10 M
5a) 5b)	• Explaining locally weighted regression with ex. • Explanation of Q-learning	5M 5M	10 M
6a) 6b)	• Bayesian belief networks explanation with example • EM algorithm explanation	5M 5M	10 M
7a) 7b)	• Explaining CADET system with example • Explanation of Radial Basis function	6 M 4 M	10 M

Internal Assessment Test 2 Solutions– Dec.2022

Sub:	Artificial Intelligence & Machine Learning							Code:	18CS71
Date:	01/12/2022	Duration:	90mins	Max Marks:	50	Sem:	VII	Branch:	ISE

Note: Answer Any Five Questions

1. 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.

Solution:

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

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

- Deriving the stochastic gradient descent rule: Stochastic gradient descent involves
- For each training example d every weight w_{ji} is updated by adding to it Δw_{ji}

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

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

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

Case 1: Training Rule for Output Unit Weights.

- 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

$$\frac{\partial E_d}{\partial net_j} = \frac{\partial E_d}{\partial o_j} \frac{\partial o_j}{\partial net_j} \quad \text{.....equ(3)}$$

To begin, consider just the first term in Equation (3)

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

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 $k = j$.

$$\begin{aligned} \frac{\partial E_d}{\partial o_j} &= \frac{\partial}{\partial o_j} \frac{1}{2}(t_j - o_j)^2 \\ &= \frac{1}{2} 2(t_j - o_j) \frac{\partial (t_j - o_j)}{\partial o_j} \\ &= -(t_j - o_j) \end{aligned} \quad \text{.....equ (4)}$$

Next consider the second term in Equation (3). 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,

$$\begin{aligned} \frac{\partial o_j}{\partial net_j} &= \frac{\partial \sigma(net_j)}{\partial net_j} \\ &= o_j(1 - o_j) \end{aligned} \quad \text{.....equ(5)}$$

Substituting expressions (4) and (5) into (3), we obtain

$$\frac{\partial E_d}{\partial net_j} = -(t_j - o_j)\, o_j(1 - o_j) \quad \text{.....equ(6)}$$

and combining this with Equations (1) and (2), we have the stochastic gradient descent rule for output units

$$\Delta w_{ji} = -\eta \frac{\partial E_d}{\partial w_{ji}} = \eta\, (t_j - o_j)\, o_j(1 - o_j) x_{ji} \quad \text{.....equ (7)}$$

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 and denoted this set of units by ***Downstream(*** j).
- net_j can influence the network outputs only through the units in ***Downstream(j).***

Therefore, we can write

$$\frac{\partial E_d}{\partial net_j} = \sum_{k \in Downstream(j)} \frac{\partial E_d}{\partial net_k} \frac{\partial net_k}{\partial net_j}$$

$$= \sum_{k \in Downstream(j)} -\delta_k \frac{\partial net_k}{\partial net_j}$$

$$= \sum_{k \in Downstream(j)} -\delta_k \frac{\partial net_k}{\partial o_j} \frac{\partial o_j}{\partial net_j}$$

$$= \sum_{k \in Downstream(j)} -\delta_k \, w_{kj} \frac{\partial o_j}{\partial net_j}$$

$$= \sum_{k \in Downstream(j)} -\delta_k \, w_{kj} \, o_j(1 - o_j) \qquad \text{.....equ (8)}$$

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

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

---

2. Classify the test data {Red, SUV, Domestic} using NAÏVE Bayes classifier for the dataset shown below.

Color	Type	Origin	Stolen
Red	Sports	Domestic	Yes
Red	Sports	Domestic	No
Red	Sports	Domestic	Yes
Yellow	Sports	Domestic	No
Yellow	Sports	Imported	Yes
Yellow	SUV	Imported	No
Yellow	SUV	Imported	Yes
Yellow	SUV	Domestic	No
Red	SUV	Imported	No
Red	Sports	Imported	Yes

Solution:

Our task is to predict the target value ***(yes or no)*** of the target concept ***Stolen*** for this new instance

The probabilities of the different target values can easily be estimated based on their frequencies over the 10 training examples.

- P(Yes) = 5/10 = 0.5
- P(No) = 5/10 = 0.5

For new data {Red, SUV, Domestic} we need to classify the result

$$v_{NB} = \underset{v_j \in V}{\operatorname{argmax}} P(v_j) \prod_i P(a_i | v_j)_{\text{No}\}}$$

Now we need to find the conditional probabilities for the test data w.r.t '**Yes**' as mentioned below.

- P(Red|Vj=Yes) = 3/5 = 0.6
- P(SUV|Vj=Yes) = 1/5 = 0.2
- P(Domestic|Vj=Yes) = 2/5 = 0.4

Now we need to find the conditional probabilities for the test data w.r.t '**No**'as mentioned below.

- P(Red|Vj=No) = 2/5 = 0.4
- P(SUV|Vj=No) = 3/5 = 0.6
- P(Domestic|Vj=No) = 3/5 = 0.6

Finally for the test data we have the formula as below.

$$v_{NB} = \underset{v_j \in V}{\operatorname{argmax}} P(v_j) \prod_i P(a_i | v_j)$$

VNB {Yes} = P (Yes)*P (Red|Yes)*P (SUV|Yes)*P (Domestic|Yes) = 0.5*0.6*0.2*0.4 = 0.024

VNB {No} = P (No)* P (Red|No)*P (SUV|No)*P (Domestic|No) = 0.5*0.4*0.6*0.6 = 0.072

So for new data {Red, SUV, Domestic} the result is No

---

3. Explain the K – nearest neighbour algorithm for approximating a discrete – valued function f-> Rn-V with pseudo code.

Solution:

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.

TABLE 8.1
The k-NEAREST NEIGHBOR algorithm for approximating a discrete-valued function $f : \Re^n \rightarrow V$.

Pseudo Code:

1. Load the data
2. Initialize the value of k
3. For getting the predicted class, iterate from 1 to total number of training data points
    1. Calculate the distance between test data and each row of training data. Here we will use Euclidean distance as our distance metric since it's the most popular method. The other metrics that can be used are cosine, etc.
    2. Sort the calculated distances in ascending order based on distance values
    3. Get top k rows from the sorted array
    4. Get the most frequent class of these rows
    5. Return the predicted class

---

4. Consider the following iris dataset. Using the k-Means Clustering approach, classify the below examples into k clusters by taking k value as 2.Also mention the applications of k-Means clustering approach.(Can consider 2 initial values for the first step as No.3 and No.6).

No	sepal.length	sepal.width
1	5.1	3.5
2	4.9	3
3	7	3.2
4	6.4	3.2
5	6.3	3.3
6	5.8	2.7

Solution:

Since k=2, initial centroid values are as below.

Initial centroid	X	Y
c1	7	3.2
c2	5.8	2.7

2)Calculate the euclidean distance of the given equation

Distance(X,Y)(a,b) = Sqrt(X-a)2+(X-b)2

Initial centroid	X	Y	Distance from cluster1	Distance from cluster2
1	5.1	3.5	sqrt(7-5.1)2+(3.2-3.5)2 = 1.92	sqrt(5.8-5.1)2+(2.7-3.5)2 = 1.02
2	4.9	3	sqrt(7-4.9)2+(3.2-3)2 = 2.10	sqrt(5.8-4.9)2+(2.7-3)2 = 0.94
3	7	3.2	sqrt(7-7)2+(3.2-3.2)2 = 0	sqrt(5.8-7)2+(2.7-3.2)2 = 1.30
4	6.4	3.2	sqrt(7-6.4)2+(3.2-3.2)2 = 0.6	sqrt(5.8-6.4)2+(2.7-3.2)2 = 0.94
5	6.3	3.3	sqrt(7-6.3)2+(3.2-3.3)2 = 0.70	sqrt(5.8-6.3)2+(2.7-3.3)2 = 0.81
6	5.8	2.7	sqrt(7-5.8)2+(3.2-2.7)2 = 1.30	sqrt(5.8-5.8)2+(2.7-2.7)2 = 0

1st iteration

	C1	C2	assigned to
1	1.92	1.02	c2
2	2.1	0.94	c2
3	0	1.3	c1
4	0.6	0.94	c1
5	0.7	0.81	c1
6	1.3	0	c2

Values 3, 4, 5 belongs to c1 and 1, 2, 6 belongs to c2. Now we need to calculate the new centroids.

- c1= (7+6.4+6.3)/3 = 6.56, (3.2+3.2+3.3)/3 =3.23 = (6.6,3.2)
- c2=(5.1+4.9+5.8)/3 = 5.26, (3.5+3+2.7)/3 =3.06 = (5.3,3.1)

2nd iteration

Find the distance w.r.t the updated centroid 6,6, 3.2 and 5.3,3.1

Initial centroid	X	Y	Distance from cluster1	Distance from cluster2
1	7	3.2	sqrt(6.6-5.1)2+(3.2-3.5)2 = 1.52	sqrt(5.3-5.1)2+(3.1-3.5)2 = 0.44
2	5.8	2.7	sqrt(6.6-4.9)2+(3.2-3)2 = 1.71	sqrt(5.3-4.9)2+(3.1-3)2 = 0.41
3	7	3.2	sqrt(6.6-7)2+(3.2-3.2)2 = 0.4	sqrt(5.3-7)2+(3.1-3.2)2 = 1.70
4	6.4	3.2	sqrt(6.6-6.4)2+(3.2-3.2)2 = 0.2	sqrt(5.3-6.4)2+(3.1-3.2)2 = 1.10
5	6.3	3.3	sqrt(6.6-6.3)2+(3.2-3.3)2 = 0.31	sqrt(5.3-6.3)2+(3.1-3.3)2 = 1.07
6	5.8	2.7	sqrt(6.6-5.8)2+(3.2-2.7)2 = 0.94	sqrt(5.3-5.8)2+(3.1-2.7)2 = 0.78

	C1	C2	assigned to
1	1.52	0.44	c2
2	1.71	0.41	c2
3	0.4	1.7	c1
4	0.2	1.1	c1
5	0.31	1.07	c1
6	0.94	0.78	c2

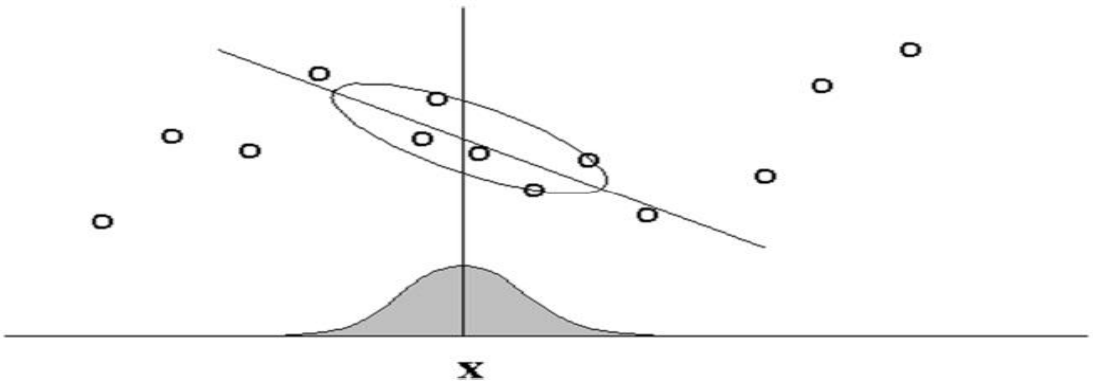
Values 3, 4, 5 belongs to c1 and 1, 2, 6 belongs to c2. Since there is no change in the previous cluster values, we will stop here and the final clusters are as mentioned below.

	C1	C2	assigned to
1	1.52	0.44	c2
2	1.71	0.41	c2
3	0.4	1.7	c1
4	0.2	1.1	c1
5	0.31	1.07	c1
6	0.94	0.78	c2

5.a) Explain locally weighted linear regression with an example.

Solution:

- a note on terminology:
  - *Regression* means approximating a real-valued target function
  - *Residual* is the error $\hat{f}(x) - f(x)$ in approximating the target function
  - *Kernel function* is the function of distance that is used to determine the weight of each training example. In other words, the kernel function is the function K such that $w_i = K(d(x_i, x_q))$
- nearest neighbor approaches can be thought of as approximating the target function at the single query point x_q
- locally weighted regression is a generalization to this approach, because it constructs an explicit approximation of f over a local region surrounding x_q



- target function is approximated using a **linear function**

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

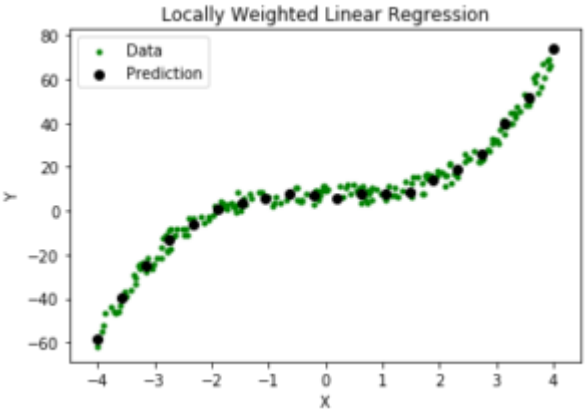
- methods like **gradient descent** can be used to calculate the coefficients $w_0, w_1, \ldots, w_n$ to minimize the error in fitting such linear functions
- ANNs require a global approximation to the target function
- here, just a local approximation is needed

$\Rightarrow$ the error function has to be redefined

- Consider a query point x = 5.0 and let x^{(1)} and x^{(2) be two points in the training set such that x^{(1)} = 4.9 and x^{(2)} = 3.0.
  Using the formula w^{(i)} = exp(frac{-(x^{(i)} - x)^2}{2tau^2}) with tau = 0.5:
  w^{(1)} = exp(frac{-(4.9 - 5.0)^2}{2(0.5)^2}) = 0.9802
  w^{(2)} = exp(frac{-(3.0 - 5.0)^2}{2(0.5)^2}) = 0.000335

- So, J(theta) = 0.9802*(theta^Tx^{(1)} - y^{(1)}) + 0.000335*(theta^Tx^{(2)} - y^{(2)})
  Thus, the weights fall exponentially as the distance between x and x^{(i)} increases and so does the contribution of error in prediction for x^{(i)} to the cost.

Consequently, while computing theta, we focus more on reducing (theta^Tx^{(i)} - y^{(i)})^2 for the points lying closer to the query point (having larger value of w^{(i)}).



Steps involved in locally weighted linear regression are:

Compute theta to minimize the cost. J(theta) = $sum_{i=1}^{m} w^{(i)}(theta^Tx^{(i)} - y^{(i)})^2
Predict Output: for given query point x,
return: theta^Tx

5.b)Write a note on Q-learning.
Solution:

For each s, a initialize the table entry $\hat{Q}(s, a)$ to zero
Oberserve the current state s

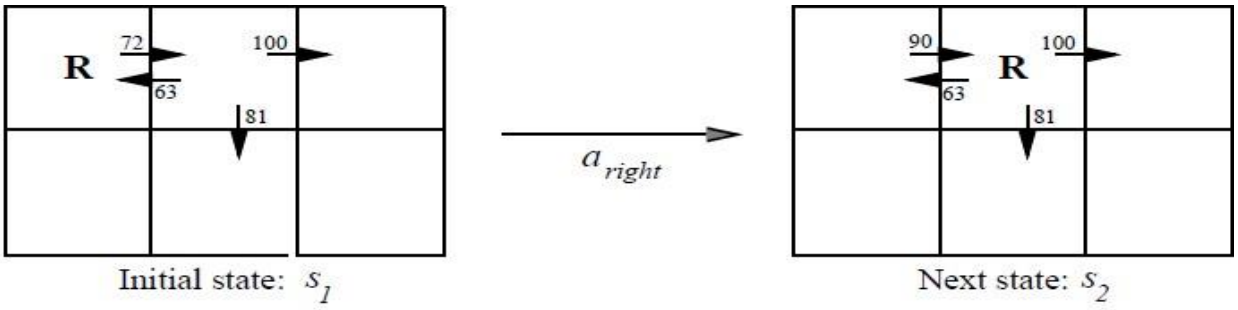
Do forever:

- Select an action a and execute it
- Receive immediate reward r
- Observe new state s'
- Update each table entry for $\hat{Q}(s, a)$ as follows

$$\hat{Q}(s, a) \leftarrow r + \gamma max_{a'} \hat{Q}(s', a')$$

- $s \leftarrow s'$

$\Rightarrow$ using this algorithm the agent's estimate $\hat{Q}$ converges to the actual Q, provided the system can be modeled as a deterministic Markov decision process, r is bounded, and actions are chosen so that every state-action pair is visited infinitely often



$$\hat{Q}(s_1, a_{right}) \leftarrow r + \gamma \cdot \max_{a'} \hat{Q}(s_2, a')$$
$$\leftarrow 0 + 0.9 \cdot \max\{66, 81, 100\}$$
$$\leftarrow 90$$

- each time the agent moves, Q Learning propagates $\hat{Q}$ estimates *backwards* from the new state to the old

---

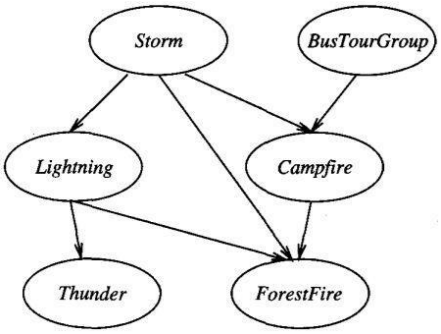
6.a. Explain Bayesian Belief Networks and conditional independence with example.

Solution:

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.



	S,B	S,¬B	¬S,B	¬S,¬B
C	0.4	0.1	0.8	0.2
¬C	0.6	0.9	0.2	0.8

Campfire

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.*

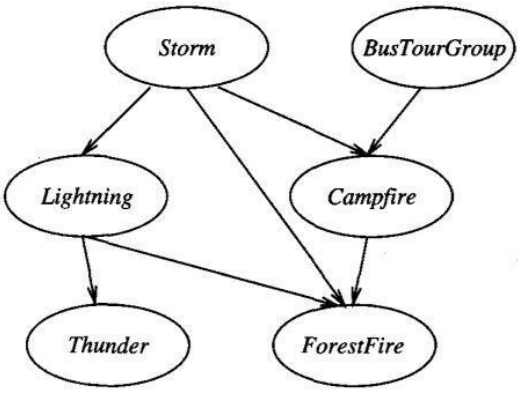
- BN represented by a directed acyclic graph, together with sets of local conditional probabilities.
- Each variable in the joint space is represented by a node in the Bayesian network.
- The network arcs represent the assertion that the variable is conditionally independent of its non-descendants in 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

$$P(y_1, \ldots, y_n) = \prod_{i=1}^{n} P(y_i | Parents(Y_i))$$

Where, Parents(Y_i) denotes the set of immediate predecessors of Y_i 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***.



	S,B	S,¬B	¬S,B	¬S,¬B
C	0.4	0.1	0.8	0.2
¬C	0.6	0.9	0.2	0.8

Campfire

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

6.b. Explain the EM Algorithm in detail.

Solution:

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

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

7.a. Explain the CADET System with Case based reasoning with example.
Solution:

a. Case-based reasoning (CBR) is a learning paradigm based on lazy learning methods and they classify new query instances by analysing similar instances while ignoring instances that are very different from the query.
b. In CBR represent instances are not represented as real-valued points, but instead, they use a ***rich symbolic*** representation.
c. CBR has been applied to problems such as conceptual design of mechanical devices based on a stored library of previous designs, reasoning about new legal cases based on previous rulings, and solving planning and scheduling problems by reusing and combining portions of previous solutions to similar problems

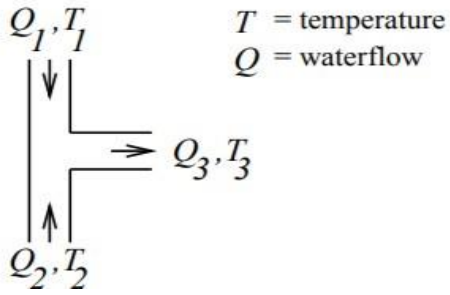
A prototypical example of a case-based reasoning

d. The CADET system employs case-based reasoning to assist in the conceptual design of simple mechanical devices such as water faucets.
e. It uses a library containing approximately 75 previous designs and design fragments to suggest conceptual designs to meet the specifications of new design problems.
f. Each instance stored in memory (e.g., a water pipe) is represented by describing both its structure and its qualitative function.

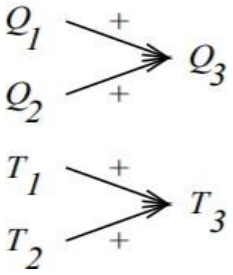
The problem setting is illustrated in below figure

A stored case: T−junction pipe

Structure:



Function:



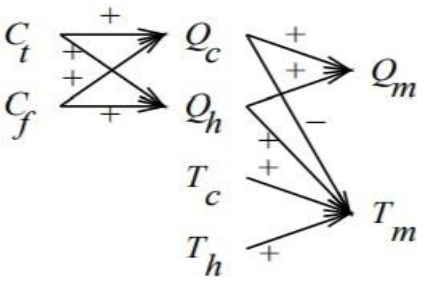
a. The function is represented in terms of the qualitative relationships among the water- flow levels and temperatures at its inputs and outputs.
b. In the functional description, an arrow with a "+" label indicates that the variable at the arrowhead increases with the variable at its tail. A "-" label indicates that the variable at the head decreases with the variable at the tail.
c. Here Q_c refers to the flow of cold water into the faucet, Q_h to the input flow of hot water, and Q_m to the single mixed flow out of the faucet.
d. T_c, T_h, and T_m refer to the temperatures of the cold water, hot water, and mixed water respectively.
e. The variable C_t denotes the control signal for temperature that is input to the faucet, and C_f denotes the control signal for waterflow.
f. The controls C_t and C_f are to influence the water flows Q_c and Q_h, thereby indirectly influencing the faucet output flow Q_m and temperature T_m.

A problem specification: Water faucet

Structure:

?

Function:



g. CADET searches its library for stored cases whose functional descriptions match the design problem. If an exact match is found, indicating that some stored case implements exactly the desired function, then this case can be returned as a suggested solution to the design problem. If no exact match occurs, CADET may find cases that match various subgraphs of the desired functional specification.

7.b. Write a note on Radial basis function.

Solution:

- 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
- 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(xu, 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.

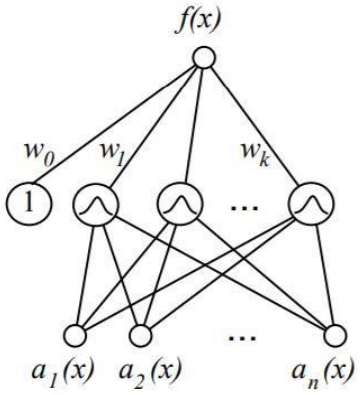
Example: Radial basis function (RBF) network

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.
- 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.
