SCHEME FOR EVALUATION
&
ANSWER KEY

Internal Assessment Test 2 – December 2023

Sub:	Artificial Intelligence and Machine Learning – Set 2					Sub Code:	18CS71	Branch:	ISE
Date:	04-12-2023	Duration:	90 Minutes	Max Marks:	50	Sem / Sec:	**7 A & B**		**OBE**

	Answer any FIVE FULL Questions	Total	Marks Split-up
1	a) Write an algorithm for back propagation which uses stochastic gradient descent method. b) Derive the back propagation rule considering the output layer and training rule for output unit weights	10	a) Algorithm (4M) Explanation (1M) b) Derivation (4M) Explanation (1M)
2	a) Explain Brute-Force Map Learning algorithm (6m) b) Write Bayes theorem and explain Maximum a Posteriori (MAP) Hypothesis and Maximum Likelihood (ML) Hypothesis(4m)	10	a) Algorithm (5M) Explanation (1M) b) Bayes theorem (2M) Brief Explanation of Hypothesis (1+1M)
3	a) Explain Bayesian Belief Networks and conditional independence with example(5m) b) Explain the EM Algorithm in detail(5m)	10	a) Explanation (3+2M) b) Algorithm (4M) Explanation (1M)
4	a) Explain k-nearest neighbour learning algorithm with example(6m) b) Explain case-based reasoning with example(4m)	10	a) Algorithm with explanation (3+1M) Example (2M) b) Explanation (2M) Example (2M)
5	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)	10	* Assign each data point to their closest centroid, which will form the predefined K clusters. (2) * Calculate the variance and place a new centroid of each cluster. (4) * Reassign each datapoint to the new closest centroid of each cluster. (4)
6	a) Explain locally weighted linear regression. (5m) b) Write a note on Q-learning(5m)	10	a) Explanation (5M) b) Explanation (5M)

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

1.

a) Write an algorithm for back propagation which uses stochastic gradient descent method.

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

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 ry unit

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

4. Update each network weight w_{ji}

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

Where

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

b) Derive the back propagation rule considering the output layer and training rule for output unit weights.

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

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

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 $k = j$.

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

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,

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

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

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

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

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

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 ofunit 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

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

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

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

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

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

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.

a) Explain Brute-Force Map Learning algorithm (6m)

1. For each hypothesis h in H, calculate the posterior probability

$$P(h|D) = \frac{P(D|h)P(h)}{P(D)}$$

2. Output the hypothesis h_{MAP} with the highest posterior probability

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

In order specify a learning problem for the BRUTE-FORCE MAP LEARNING algorithm we must specify what values are to be used for P(h) and for P(D|h) ?

Let's choose P(h) and for P(D|h) to be consistent with the following assumptions:

· The training data D is noise free (i.e., $d_i = c(x_i)$)

· The target concept c is contained in the hypothesis space H

· Do not have a priori reason to believe that any hypothesis is more probable than any other.

What values should we specify for P(h)?

· Given no prior knowledge that one hypothesis is more likely than another, it is reasonable to assign the same prior probability to every hypothesis h in H.

· Assume the target concept is contained in H and require that these prior probabilities sum to 1.

$$P(h) = \frac{1}{|H|} \text{ for all } h \in H$$

What choice shall we make for P(D|h)?

· P(D|h) is the probability of observing the target values D = (d_1 . . .d_m) for the fixed set of instances (x_1 . . . x_m), given a world in which hypothesis h holds

· Since we assume noise-free training data, the probability of observing classification d_i given h is just 1 if $d_i = h(x_i)$ and 0 if $d_i \neq h(x_i)$. Therefore,

$$P(D|h) = \begin{cases} 1 & \text{if } d_i = h(x_i) \text{ for all } d_i \in D \\ 0 & \text{otherwise} \end{cases}$$

Given these choices for P(h) and for P(D|h) we now have a fully-defined problem for the above BRUTE-FORCE MAP LEARNING algorithm.

b) Write Bayes theorem and explain Maximum a Posteriori (MAP) Hypothesis and Maximum Likelihood (ML) Hypothesis(4m)

BAYES THEOREM

Bayes theorem provides a way to calculate the probability of a hypothesis based on its prior probability, the probabilities of observing various data given the hypothesis, and the observed data itself.

Notations

· P(h) prior probability of h, reflects any background knowledge about the chance that h is correct

· P(D) prior probability of D, probability that D will be observed

· P(D|h) probability of observing D given a world in which h holds

· P(h|D) posterior probability of h, reflects confidence that h holds after D has been observed

Bayes theorem is the cornerstone of Bayesian learning methods because it provides a way to calculate the posterior probability P(h|D), from the prior probability P(h), together with P(D) and P(D|h).

Bayes Theorem:

$$P(h|D) = \frac{P(D|h)P(h)}{P(D)}$$

· P(h|D) increases with P(h) and with P(D|h) according to Bayes theorem.

· P(h|D) decreases as P(D) increases, because the more probable it is that D will be observed independent of h, the less evidence D provides in support of h.

Maximum a Posteriori (MAP) Hypothesis

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 ($P(h_i) = P(h_j)$ for all h_i and h_j 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

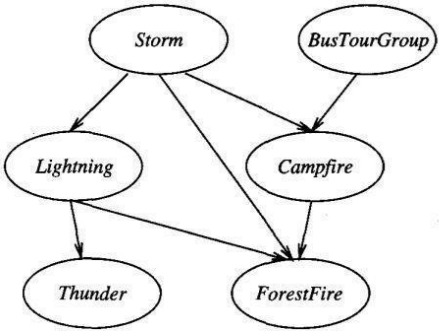
3.

a) Explain Bayesian Belief Networks and conditional independence with example(5m)

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 *conditionalindependence 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 ofits 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

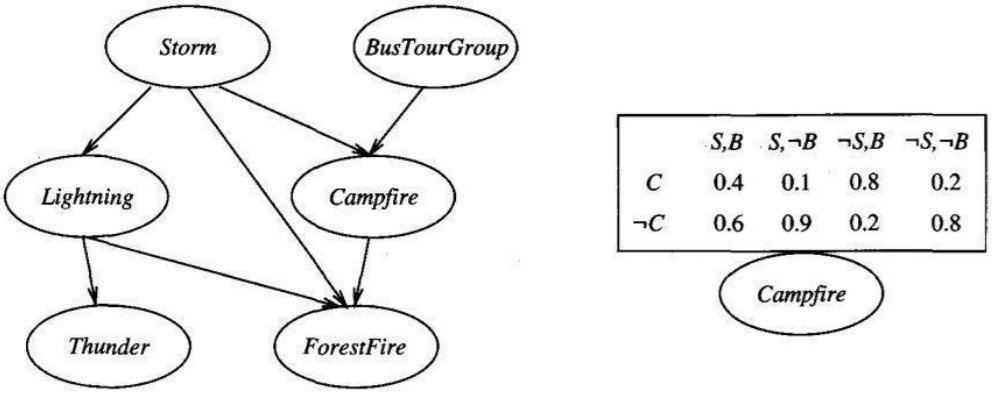
$$P(y_1, \ldots, y_n) = \prod_{i=1}^{n} P(y_i | Parents(Y_i))$$

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 parentsStorm 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

b) Explain the EM Algorithm in detail(5m)

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

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

4.

a) Explain k-nearest neighbour learning algorithm with example(6m).

- most basic instance-based method
- **assumption:**
  - instances correspond to a point in a n-dimensional space $\Re^n$
  - thus, nearest neighbors are defined in terms of the standard **Euclidean Distance**

$$d(x_i, x_j) \equiv \sqrt{\sum_{r=1}^{n} (a_r(x_i) - a_r(x_j))^2}$$

where an instance x is described by $< a_1(x), a_2(x), ..., a_n(x) >$

- target function may be either discrete-valued or real-valued

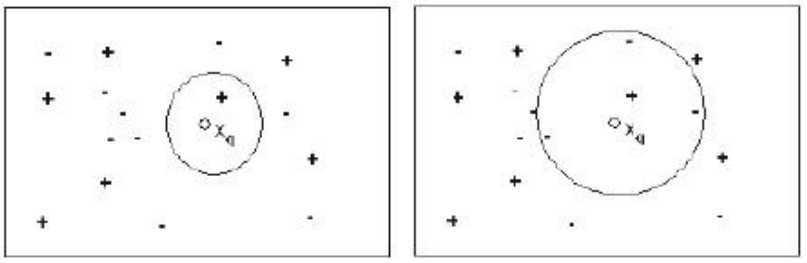
- **discrete-valued target function:**
  - $f : \Re^n \rightarrow V$ where V is the finite set $\{v_1, v_2, ..., v_s\}$
  - the target function value is the most common value among the k nearest training examples

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

  where $\delta(a, b) = (a == b)$

- **continuous-valued target function:**
  - algorithm has to calculate the mean value instead of the most common value
  - $f : \Re^n \rightarrow \Re$

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



- e.g. instances are points in a two-dimensional space where the target function is boolean-valued
  - 1-nearest neighbor: x_q is classified positive
  - 4-nearest neighbor: x_q is classified negative

- highly effective inductive inference method for many practical problems provided a sufficiently large set of training examples
- robust to noisy data
- weighted average smoothes out the impact of isolated noisy training examples
- **inductive bias of k-nearest neighbors**
  - assumption that the classification of x_q will be similar to the classification of other instances that are nearby in the Euclidean Distance
- **curse of dimensionality**
  - distance is based on all attributes
  - in contrast to decision trees and inductive logic programming
  - solutions to this problem
    - attributes can be weighted differently
    - eliminate least relevant attributes from instance space

b) Explain case-based reasoning with example(4m)

Can apply instance-based learning even when $X \neq \Re^n$

$\rightarrow$ need different "distance" metric

Case-Based Reasoning is instance-based learning applied to instances with symbolic logic descriptions

```
((user-complaint error53-on-shutdown)
 (cpu-model PowerPC)
 (operating-system Windows)
 (network-connection PCIA)
 (memory 48meg)
 (installed-applications Excel Netscape VirusScan)
 (disk 1gig)
 (likely-cause ???))
```

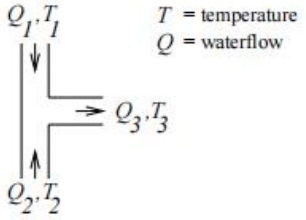
CADET: 75 stored examples of mechanical devices

- each training example: ⟨ qualitative function, mechanical structure⟩
- new query: desired function,
- target value: mechanical structure for this function

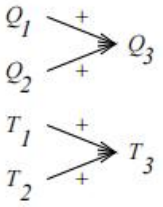
Distance metric: match qualitative function descriptions

A stored case: T–junction pipe

Structure:



Function:

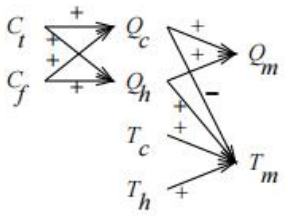


A problem specification: Water faucet

Structure:

?

Function:



- Instances represented by rich structural descriptions
- Multiple cases retrieved (and combined) to form solution to new problem
- Tight coupling between case retrieval and problem solving

Bottom line:

- Simple matching of cases useful for tasks such as answering help-desk queries
- Area of ongoing research

5.

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

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

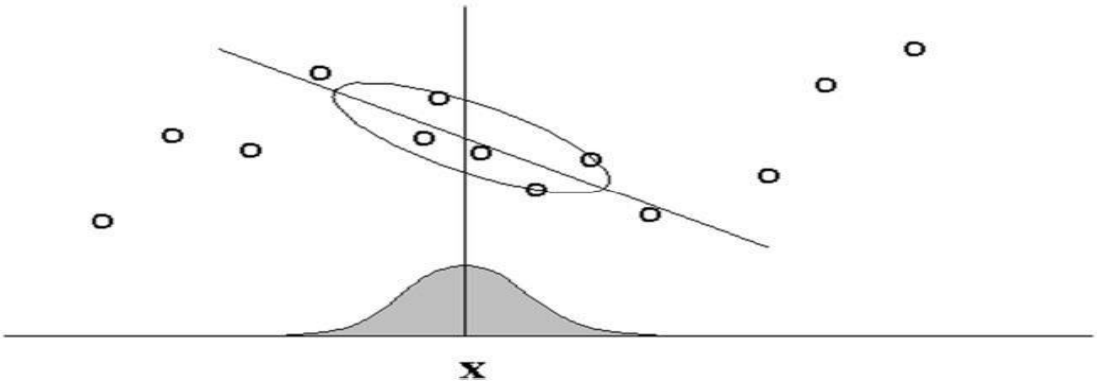
Values 3, 4, 5 belongs to c1 and 1, 2, 6 belongs to c2. Since there is no change in the previous clustervalues, we will stop here and the final clusters are as mentioned below.

	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

6.

a) Explain locally weighted linear regression. (5m)

- 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

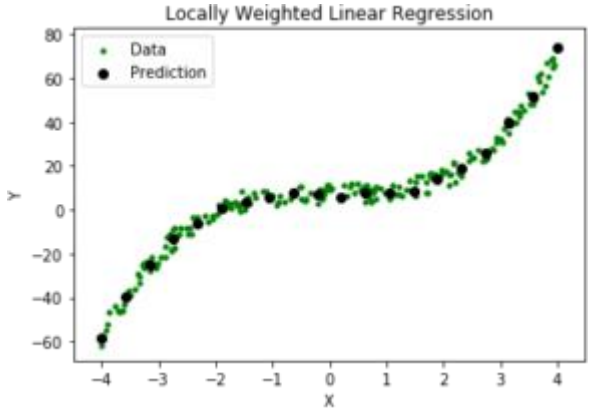
$\Rightarrow$ the error function has to be redefined

- Consider a query point x = 5.0 and let x^{(1)} and x^{(2) be two points in the training set suchthat 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 doesthe 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 thepoints 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)})^2Predict Output: for given query point x,

return: theta^Tx

b) Write a note on Q-learning(5m)

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

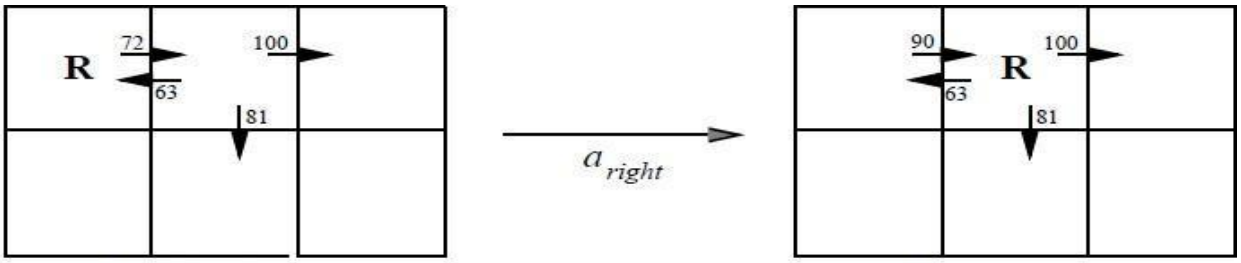
Do forever:

- 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



Initial state: s_1 Next state: s_2

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

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