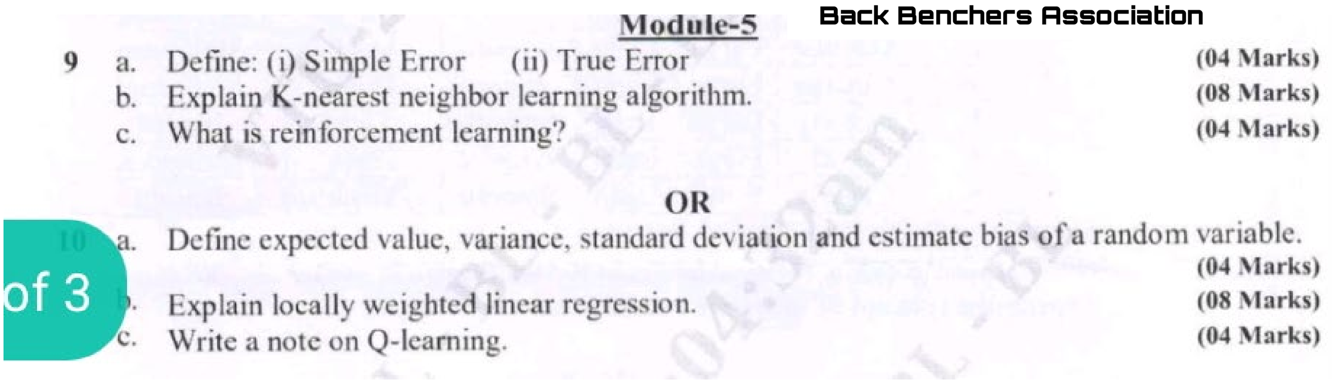


Module-5

9 a. Define: (i) Simple Error (ii) True Error (04 Marks)
b. Explain K-nearest neighbor learning algorithm. (08 Marks)
c. What is reinforcement learning? (04 Marks)

OR

10 a. Define expected value, variance, standard deviation and estimate bias of a random variable. (04 Marks)
b. Explain locally weighted linear regression. (08 Marks)
c. Write a note on Q-learning. (04 Marks)

9A)

Sample Error –
The sample error of a hypothesis with respect to some sample S of instances drawn from X is the fraction of S that it misclassifies.
Definition: The sample error (***errors(h)***) of hypothesis h with respect to target function f and data sample S is

$$error_S(h) \equiv \frac{1}{n} \sum_{x \in S} \delta(f(x), h(x))$$

True Error –
The true error of a hypothesis is the probability that it will misclassify a single randomly drawn instance from the distribution **D**.
Definition: The true error (error**D**(h)) of hypothesis h with respect to target function f and distribution **D**, is the probability that h will misclassify an instance drawn at random according to **D**.

$$error_{\mathcal{D}}(h) \equiv \Pr_{x \in \mathcal{D}}[f(x) \neq h(x)]$$

9B)

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*

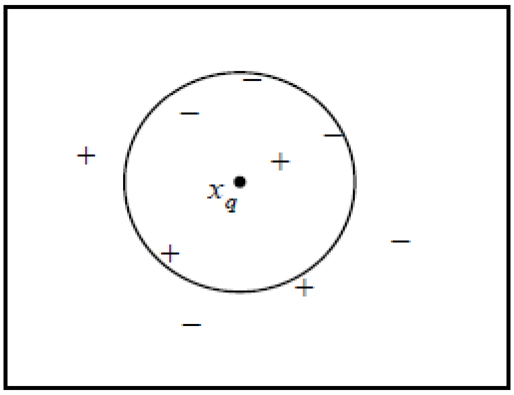
Classification algorithm:
- Given a query instance x_q to be classified,
  - Let $x_1 \dots x_k$ denote the k instances from *training_examples* that are nearest to x_q
  - Return

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

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

- The value f(xq) returned by this algorithm as its estimate of f(xq) is just the most common value of f among the k training examples nearest to xq.
- If k = 1, then the 1- Nearest Neighbor algorithm assigns to f(xq) the value f(xi). Where xi is the training instance nearest to xq.
- For larger values of k, the algorithm assigns the most common value among the k nearest training examples.

- Below figure illustrates the operation of the k-Nearest Neighbor algorithm for the case where the instances are points in a two-dimensional space and where the target function is Boolean valued.



- The positive and negative training examples are shown by "+" and "-" respectively. A query point xq is shown as well.
- The 1-Nearest Neighbor algorithm classifies xq as a positive example in this figure, whereas the 5-Nearest Neighbor algorithm classifies it as a negative example.

The K- Nearest Neighbor algorithm for approximation a **real-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
              $\hat{f}(x_q) \leftarrow \frac{\sum_{i=1}^{k} f(x_i)}{k}$
```

9C) Reinforcement learning addresses the question of how an autonomous agent that senses and acts in its environment can learn to choose optimal actions to achieve its goals.

INTRODUCTION

- Consider building a **learning robot**. The robot, or ***agent***, has a set of sensors to observe the state of its environment, and a set of actions it can perform to alter this state.
- Its task is to learn a control strategy, or ***policy***, for choosing actions that achieve its goals.
- The goals of the agent can be defined by a ***reward function*** that assigns a numerical value to each distinct action the agent may take from each distinct state.
- This reward function may be built into the robot, or known only to an external teacher who provides the reward value for each action performed by the robot.
- The **task** of the robot is to perform sequences of actions, observe their consequences, and learn a control policy.
- The control policy is one that, from any initial state, chooses actions that maximize the reward accumulated over time by the agent.

10A) **Mean, Variance and Standard Deviation**

The Mean (expected value) is the average of the values taken on by repeatedly sampling the random variable

Definition: Consider a random variable Y that takes on the possible values y1, . . . yn. The expected value (Mean) of Y, E[Y], is

$$E[Y] \equiv \sum_{i=1}^{n} y_i \Pr(Y = y_i)$$

The Variance captures how far the random variable is expected to vary from its mean value.
Definition: The variance of a random variable Y, Var[Y], is

$$Var[Y] \equiv E[(Y - E[Y])^2]$$

The variance describes the expected squared error in using a single observation of Y to estimate its mean E[Y].
The square root of the variance is called the standard deviation of Y, denoted σy

Definition: The standard deviation of a random variable Y, σy, is

$$\sigma_Y \equiv \sqrt{E[(Y - E[Y])^2]}$$

 Estimator:
errors(h) an ***estimator*** for the true error errorD(h): An estimator is any random variable used to estimate some parameter of the underlying population from which the sample is drawn
 Estimation bias: is the difference between the expected value of the estimator and the true value of the parameter.

Definition: The estimation bias of an estimator Y for an arbitrary parameter p is

$$E[Y] - p$$

10B) **Locally Weighted Linear Regression**
 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 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.
1. Minimize the squared error over just the k nearest neighbors:

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

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 xq :

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

3. Combine 1 and 2:

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

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

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

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.

10C) **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***

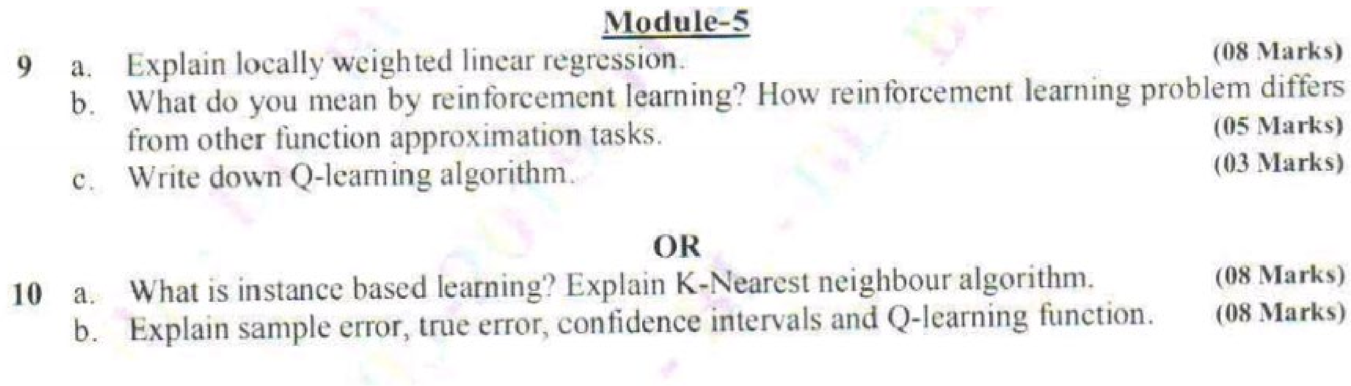
$$V^*(s) = \max_{a'} Q(s, a')$$

Rewriting Equation

$$Q(s, a) = r(s, a) + \gamma \max_{a'} Q(\delta(s, a), a')$$

□ **Q learning algorithm:**

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



Module-5

9 a. Explain locally weighted linear regression. (08 Marks)
b. What do you mean by reinforcement learning? How reinforcement learning problem differs from other function approximation tasks. (05 Marks)
c. Write down Q-learning algorithm. (03 Marks)

OR

10 a. What is instance based learning? Explain K-Nearest neighbour algorithm. (08 Marks)
b. Explain sample error, true error, confidence intervals and Q-learning function. (08 Marks)

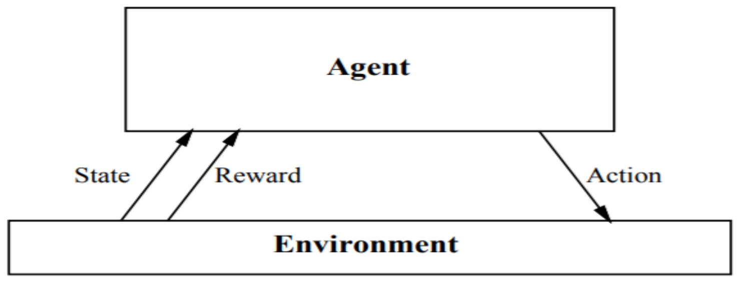
9A)REPEATED 9C REPEATED

9B) **Reinforcement Learning Problem**

 An agent interacting with its environment. The agent exists in an environment described by some set of possible states S.

 Agent perform any of a set of possible actions A. Each time it performs an action a, in some state st the agent receives a real-valued reward r, that indicates the immediate value of this state-action transition. This produces a sequence of states si, actions ai, and immediate rewards ri as shown in the figure.

 The agent's task is to learn a control policy, $\boldsymbol{\pi}$**: S → A**, that maximizes the expected sum of these rewards, with future rewards discounted exponentially by their delay.



$$\begin{tikzcd} s_0 \arrow[r, "a_0", "r_0"'] & s_1 \arrow[r, "a_1", "r_1"'] & s_2 \arrow[r, "a_2", "r_2"'] & \dots \end{tikzcd}$$

Goal: Learn to choose actions that maximize

$$r_0 + \gamma r_1 + \gamma^2 r_2 + \ldots \text{ , where } 0 \leq \gamma < 1$$

Reinforcement learning problem characteristics

1. Delayed reward: The task of the agent is to learn a target function π that maps from the current state s to the optimal action a = π (s). In reinforcement learning, training information is not available in (s, π (s)). Instead, the trainer provides only a sequence of immediate reward values as the agent executes its sequence of actions. The agent, therefore, faces the problem of ***temporal credit assignment***: determining which of the actions in its sequence are to be credited with producing the eventual rewards.

2. Exploration: In reinforcement learning, the agent influences the distribution of training examples by the action sequence it chooses. This raises the question of which experimentation strategy produces most effective learning. The learner faces a trade-off in choosing whether to favor exploration of unknown states and actions, or exploitation of states and actions that it has already learned will yield high reward.

3. Partially observable states: The agent's sensors can perceive the entire state of the environment at each time step, in many practical situations sensors provide only partial information. In such cases, the agent needs to consider its previous observations together with its current sensor data when choosing actions, and the best policy may be one that chooses actions specifically to improve the observability of the environment.

4. Life-long learning: Robot requires to learn several related tasks within the same environment, using the same sensors. For example, a mobile robot may need to learn how to dock on its battery charger, how to navigate through narrow corridors, and how to pick up output from laser printers. This setting raises the possibility of using previously obtained experience or knowledge to reduce sample complexity when learning new tasks.

10A) *k*- NEAREST NEIGHBOR LEARNING

 The most basic instance-based method is the K- Nearest Neighbor Learning. This algorithm assumes all instances correspond to points in the n-dimensional space Rn.

 The nearest neighbors of an instance are defined in terms of the standard Euclidean distance.

 Let an arbitrary instance x be described by the feature vector

((a1(x), a2(x), ………, an(x))
Where, ar(x) denotes the value of the rth attribute of instance x.
 Then the distance between two instances xi and xj is defined to be d(xi , xj)

Where,

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

10B)

The *Q* Function

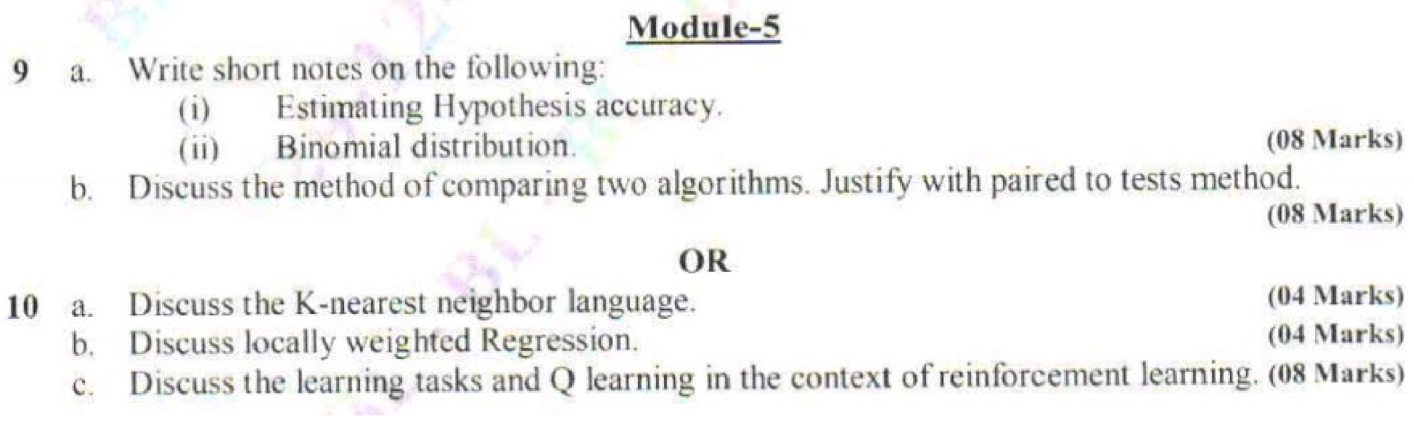
The value of Evaluation function ***Q(s, a)*** is the reward received immediately upon executing action a from state *s*, plus the value (discounted by γ) of following the optimal policy thereafter

$$Q(s,a) \equiv r(s,a) + \gamma V^*(\delta(s,a)) \qquad \textbf{equ (4)}$$

Rewrite Equation (3) in terms of ***Q(s, a)*** as

$$\pi^*(s) = \underset{a}{\operatorname{argmax}}\, Q(s,a) \qquad \textbf{equ (5)}$$

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)***.



Module-5

9 a. Write short notes on the following:
(i) Estimating Hypothesis accuracy.
(ii) Binomial distribution. **(08 Marks)**
b. Discuss the method of comparing two algorithms. Justify with paired to tests method. **(08 Marks)**

OR

10 a. Discuss the K-nearest neighbor language. **(04 Marks)**
b. Discuss locally weighted Regression. **(04 Marks)**
c. Discuss the learning tasks and Q learning in the context of reinforcement learning. **(08 Marks)**

9A)
i) **ESTIMATING HYPOTHESIS ACCURACY**
Sample Error –
The sample error of a hypothesis with respect to some sample S of instances drawn from X is the fraction of S that it misclassifies.
Definition: The sample error (***errors(h)***) of hypothesis h with respect to target function f and data sample S is

$$error_S(h) \equiv \frac{1}{n} \sum_{x \in S} \delta(f(x), h(x))$$

Where n is the number of examples in S, and the quantity δ(f(x), h(x)) is 1 if f (x) ≠ h(x), and 0 otherwise.

True Error –

The true error of a hypothesis is the probability that it will misclassify a single randomly drawn instance from the distribution **D**.

Definition: The true error (error**D**(h)) of hypothesis h with respect to target function f and distribution **D**, is the probability that h will misclassify an instance drawn at random according to **D**.

$$error_{\mathcal{D}}(h) \equiv \Pr_{x \in \mathcal{D}} [f(x) \neq h(x)]$$

ii) **The Binomial Distribution**

Consider the following problem for better understanding of Binomial Distribution

 Given a worn and bent coin and estimate the probability that the coin will turn up heads when tossed.

 Unknown probability of heads ***p***. Toss the coin ***n*** times and record the number of times ***r*** that it turns up heads.

Estimate of ***p* = *r* / *n***

 If the experiment were *rerun*, generating a new set of ***n*** coin tosses, we might expect the number of heads ***r*** to vary somewhat from the value measured in the first experiment, yielding a somewhat different estimate for ***p***.

 The Binomial distribution describes for each possible value of ***r*** (i.e., from 0 to n), the probability of observing exactly ***r*** heads given a sample of ***n*** independent tosses of a coin whose true probability of heads is ***p***.

10b repeated

10c)