ML MODULE 5

1. **Discuss the learning tasks and Q learning in the context of reinforcement learning**

> **THE LEARNING TASK**
>
> - Consider Markov decision process (MDP) where the agent can perceive a set S of distinct states of its environment and has a set A of actions that it can perform.
> - At each discrete time step t, the agent senses the current state s_t, chooses a current action a_t, and performs it.
> - The environment responds by giving the agent a reward $r_t = r(s_t, a_t)$ and by producing the succeeding state $s_{t+1} = \delta(s_t, a_t)$. Here the functions $\delta(s_t, a_t)$ and $r(s_t, a_t)$ depend only on the current state and action, and not on earlier states or actions.
>
> The task of the agent is to learn a policy, $\pi: S \rightarrow A$, for selecting its next action a_t based on the current observed state s_t; that is, $\pi(s_t) = a_t$.

Q LEARNING

How can an agent learn an optimal policy π^* for an arbitrary environment?

The training information available to the learner is the sequence of immediate rewards $r(s_i,a_i)$ for $i = 0, 1, 2, \ldots$. Given this kind of training information it is easier to learn a numerical evaluation function defined over states and actions, then implement the optimal policy in terms of this evaluation function.

What evaluation function should the agent attempt to learn?

One obvious choice is V^*. The agent should prefer state s_1 over state s_2 whenever $V^*(s_1) > V^*(s_2)$, because the cumulative future reward will be greater from s_1

The optimal action in state s is the action a that maximizes the sum of the immediate reward $r(s, a)$ plus the value V^* of the immediate successor state, discounted by γ.

$$\pi^*(s) = \underset{a}{\operatorname{argmax}}[r(s,a) + \gamma V^*(\delta(s,a))] \qquad \textbf{equ (3)}$$

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)***.

Define the following terms a) Sample error. b) True error. c) Expected value.

Definition: The sample error (***error$_s$(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 $\delta(f(x), h(x))$ is 1 if $f(x) \neq h(x)$, 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 $\mathcal{D}$.

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

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

2. Explain Binomial Distribution.

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.

3. Explain the K – nearest neighbour algorithm

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

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

4. **Explain CADET System using Case based reasoning.**
5. **Discuss the method of comparing two algorithms. Justify with paired to tests method.**