Reinforcement Learning:

Reinforcement learning is a type of machine learning where an agent learns how to behave in an environment by performing actions and receiving feedback in the form of rewards or penalties. The goal is for the agent to discover the best way to achieve a specific objective over time. The agent explores different actions, learns from the consequences, and adjusts its strategy to maximize cumulative rewards. It's like teaching a computer to make decisions through trial and error, with the ultimate aim of making better choices based on past experiences.

Supervised learning is not always possible due to challenges in obtaining labeled data, where each input has a corresponding correct output. Here are some simple examples:

1. **Rare diseases diagnosis:**
   - Example: Identifying extremely rare diseases may not have enough labeled cases, making it difficult to train a supervised model effectively.

2. **Autonomous driving:**
   - Example: Collecting labeled data for every possible driving scenario, including rare and dangerous situations, can be impractical and may not cover all possible conditions on the road.

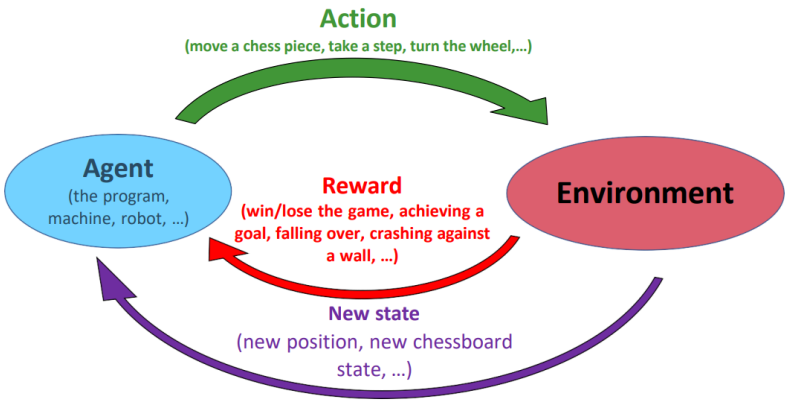
3. **Fraud detection:**
   - Example: Fraudulent activities are relatively uncommon, and obtaining labeled data for various types of fraud instances may be limited, making it challenging to train a supervised model to accurately detect fraud.

4. **Speech recognition for diverse accents:**
   - Example: Training a supervised model for recognizing speech in various accents would require a vast amount of labeled data for each accent, which may not be readily available.

5. **Predicting natural disasters:**
   - Example: Anticipating rare and extreme events like earthquakes or tsunamis might lack sufficient labeled data due to the infrequency of such occurrences.

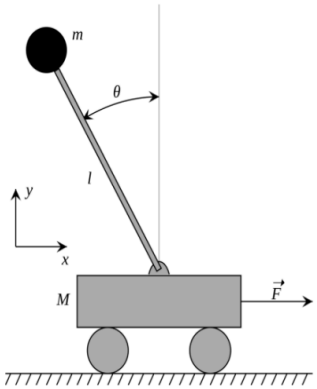
6. **Sentiment analysis in new domains:**
   - Example: If you want to perform sentiment analysis on user reviews for a completely new product or service, you may not have labeled data for that specific domain, making supervised learning challenging.

In these cases, it may be more practical to explore other machine learning approaches, such as unsupervised learning or reinforcement learning, which don't rely on having labeled examples for every possible scenario.



Flow Chart

Explain the Cart-Pole Problem



Objective: The goal is to balance the pole for as long as possible by making appropriate movements with the cart. The environment gives a reward for every time step the pole remains upright.

State: The agent (the algorithm or model) receives information about the current state of the system, including the cart's position, velocity, the pole's angle, and its angular velocity.

Action: The agent can take actions to move the cart left or right.

Reward: The agent learns through trial and error, receiving rewards(1) when it successfully keeps the pole upright and penalties when it fails.

Challenges of reinforcement learning:

- Evaluative feedback: Need trial and error to find the right action
- Delayed feedback: Actions may not lead to immediate reward
- Non-stationarity: Data distribution of visited states changes when the policy changes
- Fleeting nature of time and online data

What is Markov decision process?

A Markov Decision Process (MDP) is a mathematical framework used to model decision-making in situations where an agent interacts with an environment. The key idea is the Markov property, which says that the future state of the system only depends on the current state and the action taken, not on the history of events leading up to it.

Differences between Fully Observed MDP and Partially observed MDP:

Observability:

Fully Observed MDP: Agent has direct access to the true and complete state of the environment.

Partially Observed MDP: Agent receives incomplete or noisy observations and maintains a belief over possible states.

Decision-Making Challenge:

Fully Observed MDP: Decision-making is relatively straightforward as the agent has perfect knowledge of the current state.

Partially Observed MDP: Decision-making is more challenging as the agent must deal with uncertainty and make decisions based on partial information.

Representation:

Fully Observed MDP: The state provides a complete and sufficient representation of the environment.

Partially Observed MDP: The agent needs to maintain a belief or probability distribution over possible states.

Examples:

Fully Observed MDP: Robot navigation with accurate sensors.

Partially Observed MDP: Surveillance drone with limited visibility.

An MDP is defined as a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathbb{T}, \gamma)$

- $\mathcal{S}$: Set of possible states
- $\mathcal{A}$: Set of possible actions
- $\mathcal{R}(s, a, s')$: Distribution of reward
- $\mathbb{T}(s, a, s')$: Transition probability distribution, also written as p(s'|s,a)
- γ : Discount factor

Policy:

A policy is a strategy or set of rules that guides the decision-making of an agent in different states. The goal of the agent is to find the best or optimal policy, which leads to the highest cumulative reward over time.

Optimal Policy:
An optimal policy is the best possible strategy an agent can follow to achieve the maximum expected cumulative reward in an MDP. It specifies the action the agent should take in each possible state to maximize its long-term rewards.
Finding the optimal policy is the central objective in many reinforcement learning problems.

A policy π^* is called **optimal** if it has maximal value for all states $s \in S$:

$$V_{\pi^*}(s) = \max_{\pi} V_{\pi}(s)\,.$$

Discount Factor: The discount factor, often denoted by the symbol (gamma), is a parameter used in reinforcement learning and Markov Decision Processes (MDPs). It influences the agent's decision-making by assigning different weights to immediate rewards compared to future rewards. The discount factor is a value between 0 and 1.

Q: What is Additive discounted reward? How can you justify it?

Additive Rewards in MDPs:
In an MDP, the agent interacts with an environment by taking actions in different states. At each time step, the agent receives a reward based on the action taken and the resulting state transition. The total reward collected by the agent during its interaction with the environment is the sum of these individual rewards.

Discounted Rewards in MDPs:
Discounted rewards involve applying a discount factor (usually denoted by γ, where 0 ≤ γ ≤ 1) to future rewards in order to account for the time preference of the agent. The discounted cumulative reward, or return, at a given time step t is calculated as the sum of γ^(t-1) * R_t, where R_t is the reward at time step t.

Additive discounted reward in an MDP is the sum of the discounted rewards over all time steps:

$$G_t = R_1 + \gamma R_2 + \gamma^2 R_3 + \ldots + \gamma^{t-1} R_t$$

Discounting encourages the agent to prioritize immediate rewards over delayed rewards, which can be essential in situations where actions have long-term consequences and where there is uncertainty about the future.
Combining additive rewards with a discount factor in the context of an MDP helps in formulating a meaningful objective function for the agent, considering both the total reward and the temporal aspect of the decision-making process.

Q-Learning:

Exploration-exploitation is a fundamental challenge in reinforcement learning, representing the trade-off between trying new actions to discover their effects (exploration) and choosing actions that are known to yield high rewards based on past experiences (exploitation). Striking the right balance between exploration and exploitation is crucial for an agent to learn an optimal policy in a reinforcement learning problem.

Here's a more detailed explanation:

1. **Exploration:**
   - **Purpose:** To discover the effects of different actions in various states.
   - **Strategy:** The agent intentionally selects actions that it hasn't tried much or doesn't have much information about. This helps in learning more about the environment and improving the agent's understanding of the state-action space.
   - **Challenge:** Too much exploration may lead to suboptimal immediate rewards since the agent is deliberately trying less-promising actions.

2. **Exploitation:**
   - **Purpose:** To choose actions that are expected to yield high rewards based on the agent's current knowledge.
   - **Strategy:** The agent selects actions that it believes have been effective in the past or are likely to be the best based on its current understanding of the environment. Exploitation aims to maximize short-term rewards.
   - **Challenge:** Over-reliance on exploitation may cause the agent to miss out on discovering better actions or adapting to changes in the environment.

In practice, a common strategy for balancing exploration and exploitation is the ε-greedy strategy:

- With probability ε (exploration rate), the agent selects a random action.
- With probability \(1 - ε\) (exploitation rate), the agent selects the action with the highest estimated value (Q-value) based on its current knowledge.

This strategy allows the agent to explore new possibilities while still favoring actions that are deemed promising based on its existing knowledge.

The choice of the exploration rate (ε) is crucial, and it often decreases over time as the agent gains more experience. Initially, higher exploration rates help the agent explore the environment thoroughly, and as it learns, a shift towards more exploitation helps in maximizing cumulative rewards.

Algorithm:

Q-learning + Epsilon-Greedy

Q-learning algorithm: Find for all $s \in S$ and $a \in A$ a function $Q(s, a)$, which gives a good approximation for $Q^*(a, s)$.

1. Start with random values for $Q(s, a)$. (e.g. all zero)
2. Choose a starting state $s_0 \in S$.
3. Look up the current best action in that state, i.e. $a_0 = \text{argmax}_{a \in A} Q(s_0, a)$ or choose a random action $a_0 \in A$ with probability $\epsilon \in [0, 1]$ (Epsilon-Greedy Algorithm).
4. Apply this action and get a new state s_1 and reward $r_0 = R(s_0, a_0, s_1)$.
5. Update the value $Q(s_0, a_0)$ as follows (**Bellman equation**)

$$Q(s_0, a_0) = (1 - \alpha)Q(s_0, a_0) + \alpha \left(r_0 + \gamma \max_{a \in S} Q(s_1, a) \right) .$$

   Here $\alpha \in [0, 1]$ is the **learning rate**.
6. If s_1 is not a terminal state repeat with step 3.

The parameter ε controls the exploration rate, and it is a hyperparameter that can be tuned based on the learning task and environment.

Q: Discuss the role and intuition of each component of the update (state and action) equation.

In the Q-learning algorithm, the update equation is a crucial component that guides how the Q-values, denoted as Q(s0, a0), are adjusted based on the observed rewards and the maximum expected future rewards. The update equation has several components, each serving a specific role in shaping the learning process. Let's break down the update equation:

$$Q(s_0, a_0) = (1 - \alpha)Q(s_0, a_0) + \alpha \left(r_0 + \gamma \max_{a \in S} Q(s_1, a) \right) .$$

1. Q(s0, a0):
   - **Role:** Represents the current estimate of the Q-value for taking action a0 in state s0.
   - **Intuition:** This is the value being updated. It reflects the agent's current belief about the cumulative future rewards associated with taking action a0 in state s0.

2. alpha:
   - **Role:** Learning rate, a hyperparameter between 0 and 1 that controls the step size of the update.

- **Intuition:** The learning rate determines how much the new information should influence the current estimate. A higher learning rate means the new information has a larger impact on the Q-value.

3. r0
- **Role:** Immediate reward observed after taking action a0 in state s0.
- **Intuition:** Represents the immediate reinforcement obtained by the agent. The agent uses this reward to update its Q-value, incorporating the knowledge of the immediate consequences of the action.

4. gamma:
- **Role:** Discount factor, a hyperparameter between 0 and 1 that discounts the impact of future rewards.
- **Intuition:** Encourages the agent to consider not only immediate rewards but also future rewards. A high gamma values long-term rewards more, and a low gamma values short-term rewards more.

5. Max Q(s1, a):
- **Role:** Represents the maximum Q-value for the next state s1 over all possible actions a.
- **Intuition:** This term captures the expected future rewards. The agent estimates the maximum potential cumulative reward it can obtain from the next state s1 by selecting the action that maximizes the Q-value.

Clustering:

Advantages and disadvantages of min(Single Link) and max(Complete Link) of inter-cluster distance:

Single Linkage:

Advantages:

1. **Sensitivity to local structures:** Single linkage tends to form clusters based on local structures and is more likely to capture elongated and irregularly shaped clusters.

2. **Early merging of nearby points:** It is computationally efficient and often results in early merging of nearby data points, which can be advantageous in certain scenarios.

Disadvantages:

1. **Chain effect:** Single linkage is sensitive to outliers and noise, leading to a "chain effect" where outliers can join clusters in a chain-like manner, connecting distant points.

2. **Susceptible to chaining:** It can create long chains of data points even if they are not part of the same cluster, making it susceptible to the chaining phenomenon.

Complete Linkage:

Advantages:

1. **Robust to outliers:** Complete linkage is less sensitive to outliers compared to single linkage, as it considers the maximum distance between points from different clusters.

2. **Compact and spherical clusters:** It tends to form more compact and spherical clusters, which can be advantageous in scenarios where clusters have a more globular shape.

Disadvantages:

1. **Merge bias:** Complete linkage tends to have a merge bias towards globular clusters and may not perform well with clusters that have irregular shapes or elongated structures.

2. **Tendency to produce unbalanced clusters:** It may lead to unbalanced clusters, where one cluster is substantially larger than the others, particularly if there are outliers or noise.

(c) Consider the dataset given below. Cluster the records using agglomerative hierarchical clustering.

ID	A1	A2	A3
1	0	0	0
2	1	1	0
3	1	1	1
4	1	1	0

Define the distance metric using Manhattan distance between clusters. Show the intermediate result using single link and the final result in dendrogram.

Integer-43

3(c) using manhattan;

$d(1,2) = d(2,1) = |0-1| + |0-1| + |0-0| = 2$

$d(1,3) = d(3,1) = |0-1| + |0-1| + |0-1| = 3$

$d(1,4) = d(4,1) = |0-1| + |0-1| + |0-0| = 2$

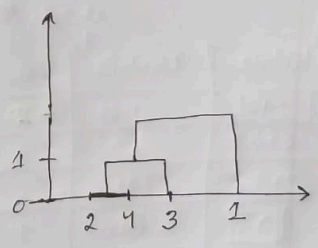
$d(2,3) = d(3,2) = |1-1| + |1-1| + |0-1| = 1$

$d(2,4) = d(4,2) = |1-1| + |1-1| + |0-0| = 0$

$d(3,4) = d(4,3) = |1-1| + |1-1| + |1-0| = 1$

Dist	1	2	3	4
1	0	2	3	2
2	2	0	1	0
3	3	1	0	1
4	2	0	1	0

Dendogram:



using Single linkage;

Min dist	1	(2,4)	3
1	0	2	3
(2,4)	2	0	1
3	3	1	0

mindist	1	(2,4),3
1	0	2
((2,4),3)	2	0

∴ The minimum distance is 2

PCA and LDA:

Scenarios in which PCA works better:

Exploratory Data Analysis: PCA is well-suited for exploratory data analysis when the primary goal is to understand the overall structure and relationships within the data. It helps in identifying patterns and reducing the dimensionality of the data.

Unsupervised Learning: PCA is an unsupervised technique, making it suitable for situations where class labels or groupings are not available or not relevant. It is often used for feature extraction in tasks such as image compression or denoising.

Dimensionality Reduction for Efficiency: When computational efficiency is a concern, PCA is a good choice, as it is computationally efficient and can handle large datasets.

Scenarios in which LDA works better:

Classification Tasks: LDA is specifically designed for classification problems. When the goal is to maximize the separation between classes, LDA often outperforms PCA because it takes class information into account during dimensionality reduction.

Supervised Learning with Labeled Data: LDA requires labeled data, and it is most effective when there is a clear distinction between classes. It is commonly used in face recognition, speech recognition, and other classification tasks.

Preserving Discriminant Information: If the goal is to preserve information that helps discriminate between classes, LDA is the preferred choice. It optimizes for both variance within classes and separation between classes.

Limitations of LDA:

1. Limited Number of Feature Projections:

Explanation: LDA can generate at most C-1 feature projections, where C is the number of classes. This means that if the classification error suggests the need for more features, additional methods must be used to provide those extra features.
Example: If you have a dataset with three classes (C=3), LDA can generate at most two feature projections.

2. Parametric Assumption - Unimodal Gaussian Likelihoods:

Explanation: LDA assumes that the data within each class follows a unimodal Gaussian (normal) distribution. If the actual distributions significantly deviate from this assumption and are non-Gaussian, LDA may not effectively preserve the complex structure of the data needed for classification.
Example: If the data within a class has a distribution that is not bell-shaped like a normal distribution, LDA might not perform optimally.

3. Failure when Discriminatory Information is in Variance, Not Mean:

Explanation: LDA assumes that the discriminatory information is primarily in the mean of the data. If the crucial information for classification is in the variance (spread) of the data rather than the mean, LDA may fail to capture this essential information.
Example: In situations where the differences between classes are better represented by the spread or variability of the data rather than the average values, LDA may not be the most suitable method.

PCA MATH:

e) The dataset shown in Table 1, reports the price and weight of three products. Show the step-by-step calculation to derive the feature vector by applying principal component analysis (PCA).

Table 1

Product	Price	Weight
1	4	7
2	5	6
3	3	1

Integer-43

3(e)

Data:

	X	Y
Product	Price	Weight
1	4	7
2	5	6
3	3	1

Step-1: Get some data

$X = 12$ $\quad Y = 14$

$\bar{X} = 3$ $\quad \bar{Y} = 4.67$

$n = 3$

Step-2: Substract the mean

	$X_i - \bar{X}$	$Y_i - \bar{Y}$
1	1	2.33
2	2	1.33
3	0	-3.67

Step-3: Calculate cov

$$cov = \begin{pmatrix} 2.5 & 6.25225 \\ 6.25225 & 10.33335 \end{pmatrix} = A$$

Step-4: Calculate eigenvalues and eigenvectors

$$A - \lambda I = \begin{pmatrix} 2.5 & 6.25225 \\ 6.25225 & 10.33335 \end{pmatrix} - \lambda \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

$$= \begin{pmatrix} 2.5 - \lambda & 6.25225 \\ 6.25225 & 10.33335 - \lambda \end{pmatrix} = 0$$

$\Rightarrow (2.5 - \lambda)(10.33335 - \lambda) - 39.09063006 = 0$

$\Rightarrow 25.833375 - 2.5\lambda - 10.33335\lambda + \lambda^2 - 39.09063006 = 0$

$\Rightarrow \lambda^2 - 12.83335\lambda - 13.25725506 = 0$

$\Rightarrow \lambda_1 = 13.79440996$

$\lambda_2 = -0.9610599583$

$\therefore$ eigenvalues $= \begin{pmatrix} 13.79440996 \\ -0.9610599583 \end{pmatrix}$

Step-5: choosing Components and forming feature vectors

For $\lambda_1 = 13.79440996$

$$\begin{pmatrix} 2.5 - 13.79440996 & 6.25225 \\ 6.25225 & 10.33335 - 13.79440996 \end{pmatrix} \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} = 0$$

$$= \begin{pmatrix} -11.29440996 & 6.25225 \\ 6.25225 & -3.46105996 \end{pmatrix} \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} = 0$$

Now,

$-11.29440996 x_1 + 6.25225 y_1 = 0 \quad \text{---} (i)$

$6.25225 x_1 - 3.46105996 y_1 = 0 \quad \text{---} (ii)$

(i) $\Rightarrow x_1 = 0.5535 y_1$

(ii) $\Rightarrow x_1 = 0.5535 y_1$

$$e_1 \sim \begin{bmatrix} \frac{0.5535}{A} \\ \frac{1}{A} \end{bmatrix} \quad \therefore e_1 \sim \begin{bmatrix} 0.4842 \\ 0.8749 \end{bmatrix}$$

$A = \sqrt{(0.5535)^2 + (1)^2} = 1.1429$

For $\lambda_2 = -0.9610599583$

$$\begin{pmatrix} 25 + 0.9610599583 & 6.25225 \\ 6.25225 & 10.33335 + 0.9610599583 \end{pmatrix} \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} = 0$$

$$\begin{pmatrix} 3.461059 & 6.25225 \\ 6.25225 & 11.29440 \end{pmatrix} \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} = 0$$

Now,

$$3.461059 x_1 + 6.25225 y_1 = 0 \quad \text{---} (III)$$
$$6.25225 x_1 + 11.29440 y_1 = 0 \quad \text{---} (IV)$$

$(III) \Rightarrow x_1 = 1.8064 y_1$

$(IV) \Rightarrow y_1 = 1.8064 y_1$

$$e_2 \sim \begin{bmatrix} \frac{1.8064}{A} \\ \frac{1}{A} \end{bmatrix}$$

$$A = \sqrt{(1.8064)^2 + (1)^2} = 2.06$$

$$\therefore e_2 \sim \begin{bmatrix} 0.8768 \\ 0.4854 \end{bmatrix}$$

$\therefore$ The feature vector is $\begin{bmatrix} 0.4842 \\ 0.8749 \end{bmatrix}$ for $\lambda_1 = 13.7944$

LDA MATH:

Question 4. [Marks: 14]

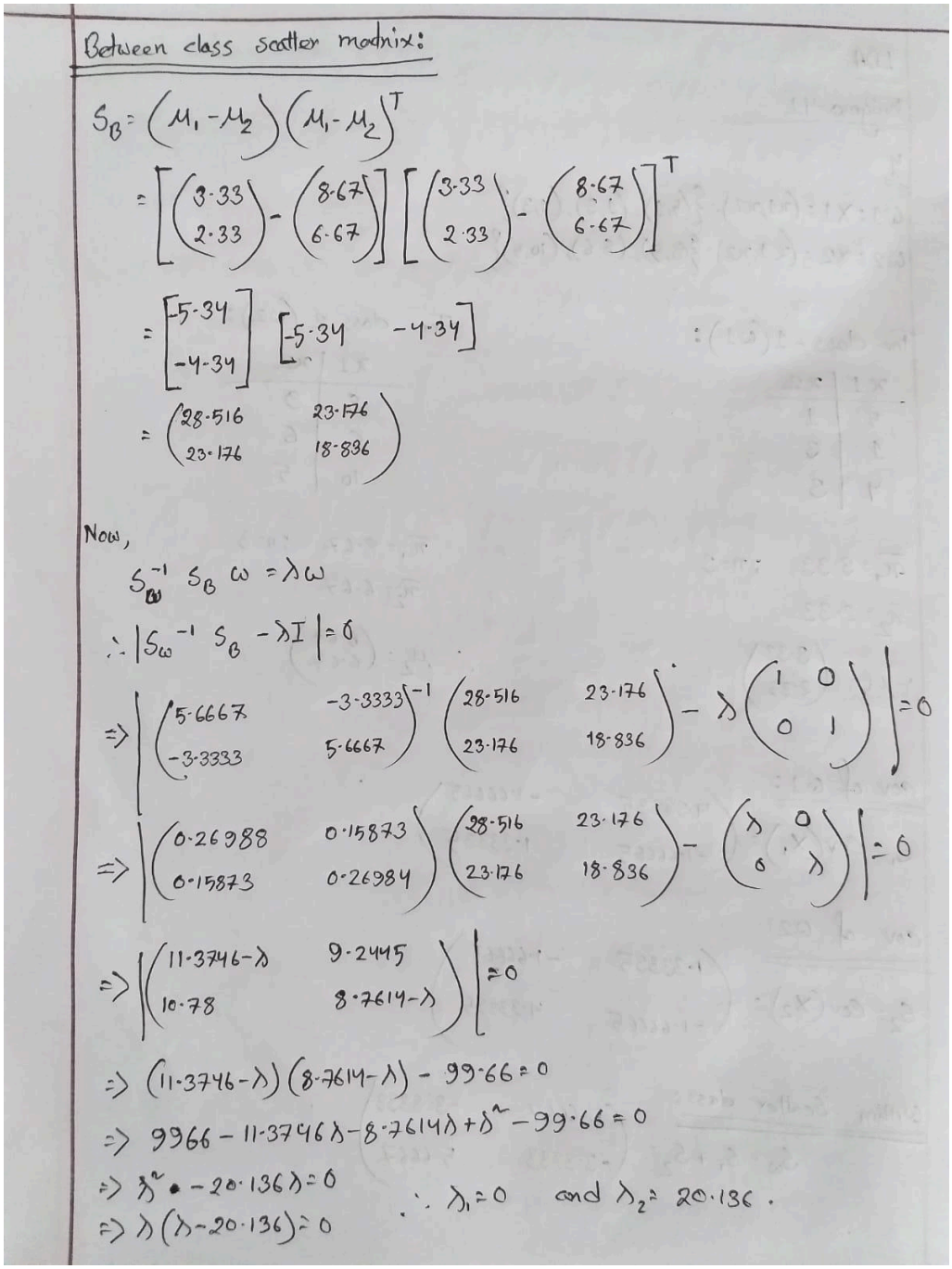
Compute the Linear Discriminant projection for the following two dimensional datasets. [10+4]

ω_1: $X_1 = (x_1, x_2) = \{(5,1), (1,3), (4,3)\}$

ω_2: $X_2 = (x_1, x_2) = \{(8,9), (8,6), (10,5)\}$

PCA & LDA

Also, is it possible to combine both PCA and LDA together to get a better result? Justify your answer with appropriate logic.



Between class scatter matrix:

$$S_B = (\mu_1 - \mu_2)(\mu_1 - \mu_2)^T$$

$$= \left[\begin{pmatrix}3.33\\2.33\end{pmatrix} - \begin{pmatrix}8.67\\6.67\end{pmatrix}\right]\left[\begin{pmatrix}3.33\\2.33\end{pmatrix} - \begin{pmatrix}8.67\\6.67\end{pmatrix}\right]^T$$

$$= \begin{bmatrix}-5.34\\-4.34\end{bmatrix}\begin{bmatrix}-5.34 & -4.34\end{bmatrix}$$

$$= \begin{pmatrix}28.516 & 23.176\\23.176 & 18.836\end{pmatrix}$$

Now,

$$S_W^{-1} S_B \omega = \lambda\omega$$

$$\therefore |S_W^{-1} S_B - \lambda I| = 0$$

$$\Rightarrow \left|\begin{pmatrix}5.6667 & -3.3333\\-3.3333 & 5.6667\end{pmatrix}^{-1}\begin{pmatrix}28.516 & 23.176\\23.176 & 18.836\end{pmatrix} - \lambda\begin{pmatrix}1 & 0\\0 & 1\end{pmatrix}\right| = 0$$

$$\Rightarrow \left|\begin{pmatrix}0.26988 & 0.15873\\0.15873 & 0.26984\end{pmatrix}\begin{pmatrix}28.516 & 23.176\\23.176 & 18.836\end{pmatrix} - \begin{pmatrix}\lambda & 0\\0 & \lambda\end{pmatrix}\right| = 0$$

$$\Rightarrow \left|\begin{pmatrix}11.3746-\lambda & 9.2445\\10.78 & 8.7614-\lambda\end{pmatrix}\right| = 0$$

$$\Rightarrow (11.3746-\lambda)(8.7614-\lambda) - 99.66 = 0$$

$$\Rightarrow 9966 - 11.3746\lambda - 8.7614\lambda + \lambda^2 - 99.66 = 0$$

$$\Rightarrow \lambda^2 - 20.136\lambda = 0$$

$$\Rightarrow \lambda(\lambda - 20.136) = 0$$

$\therefore \lambda_1 = 0$ and $\lambda_2 = 20.136$.

LDA

Enigma-41

4

$\omega 1 : X1 = (x1, x2) = \{(5,1), (1,3), (4,3)\}$

$\omega 2 : X2 = (x1, x2) = \{(8,9), (8,6), (10,5)\}$

For class-1 ($\omega 1$):

x1	x2
5	1
1	3
4	3

$\bar{x_1} = 3.33 \quad ; n = 3$

$\bar{x_2} = 2.33$

$\therefore \mu_1 = \begin{pmatrix} 3.33 \\ 2.33 \end{pmatrix}$

For class-2 ($\omega 2$):

x1	x2
8	9
8	6
10	5

$\bar{x_1} = 8.67 \quad ; n = 3$

$\bar{x_2} = 6.67$

$\therefore \mu_2 = \begin{pmatrix} 8.67 \\ 6.67 \end{pmatrix}$

cov of $\omega 1$:

$$S_1 = Cov(X_1) = \begin{pmatrix} 4.33335 & -1.66665 \\ -1.66665 & 1.33335 \end{pmatrix}$$

cov of $\omega 2$:

$$S_2 = Cov(X_2) = \begin{pmatrix} 1.33335 & -1.66665 \\ -1.66665 & 4.33335 \end{pmatrix}$$

Within Scatter class:

$$S_w = S_1 + S_2 = \begin{pmatrix} 5.6667 & -3.3333 \\ -3.3333 & 5.6667 \end{pmatrix}$$

For $\lambda_2 = 20.136$;

$$\begin{pmatrix} 11.3746 - 20.136 & 9.2445 \\ 10.78 & 8.7614 - 20.136 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = 0$$

$$-8.7614 x_1 + 9.2445 x_2 = 0 \quad \text{---} (i)$$

$$10.78 x_1 - 11.3746 x_2 = 0 \quad \text{---} (ii)$$

(i) $\Rightarrow x_1 = 1.055 x_2$

(ii) $\Rightarrow x_2 = 1055 x_2$

So, $e \sim \begin{bmatrix} \frac{1.055}{A} \\ \frac{1}{A} \end{bmatrix}$

Here, $A = \sqrt{(1.055)^2 + 1^2}$

$= 1.4537$

$\therefore e \sim \begin{bmatrix} 0.726 \\ 0.687 \end{bmatrix}$ for max^n λ.

(Ans)

For $\lambda_1 = 0$;

$$11.3746 x_1 + 9.2445 x_2 = 0 \quad \text{---} (iii)$$

$$10.78 x_1 + 8.7614 x_2 = 0 \quad \text{---} (iv)$$

(iii) $\Rightarrow x_1 = -0.813 x_2$

(iv) $\Rightarrow x_1 = -0.813 x_2$

So, $e \sim \begin{bmatrix} \frac{-0.813}{A} \\ \frac{1}{A} \end{bmatrix}$

$A = \sqrt{(-0.813)^2 + 1}$

$= 1.289$

$e \sim \begin{bmatrix} -0.631 \\ 0.776 \end{bmatrix}$

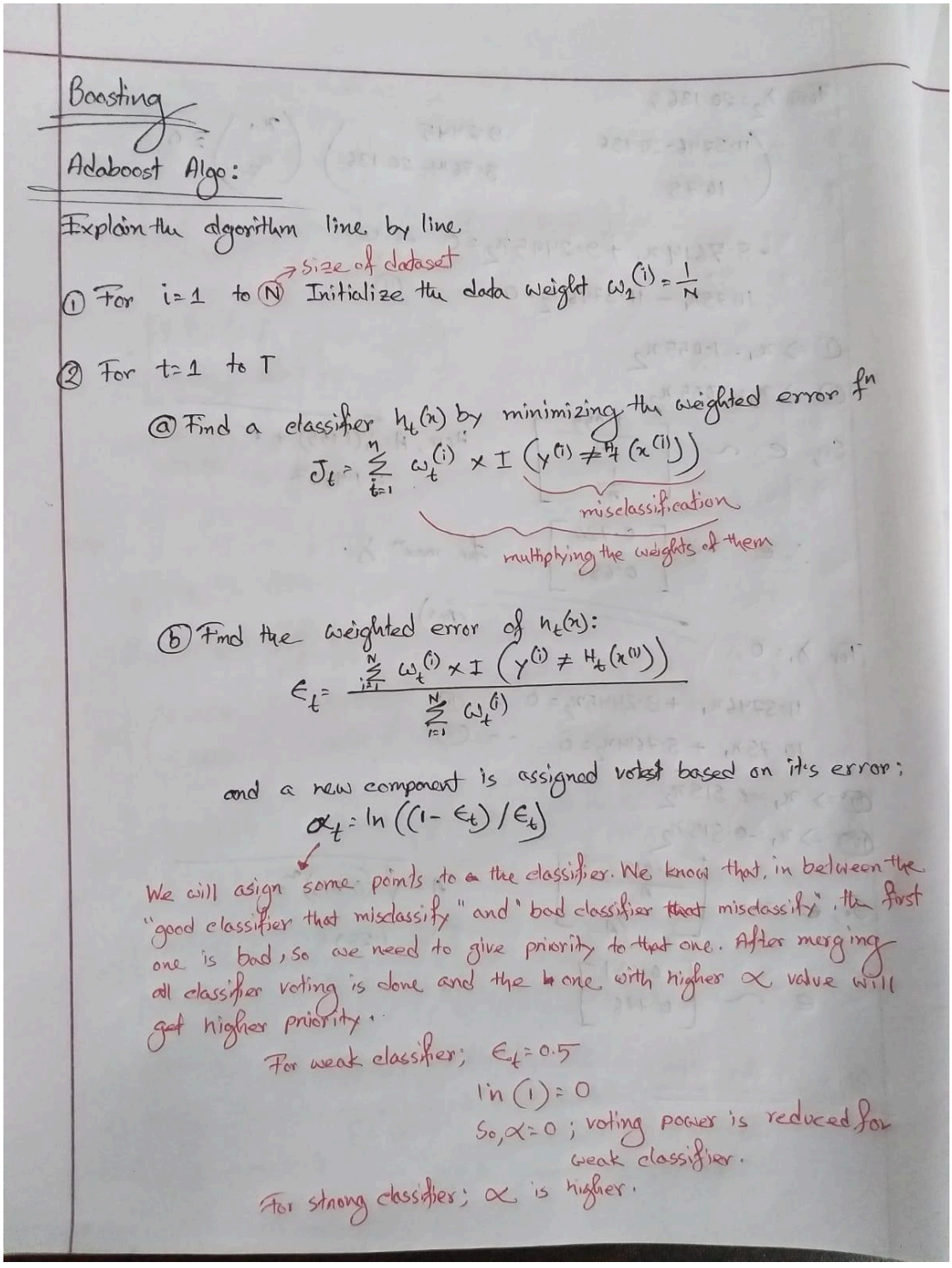
Yes, it is possible to combine PCA and LDA to get a better result in certain scenarios.

- PCA can be applied initially to reduce dimensionality and capture the overall variance in the data.
- The reduced dataset from PCA can then be used as input for LDA, which focuses on maximizing class separability in the lower-dimensional space.
- This combination leverages both techniques for noise reduction (PCA) and discriminative projection (LDA).

Justification:
PCA can help eliminate noise and capture the most significant variability in the data, providing a cleaner input for LDA. LDA, in turn, can then focus on the discriminating information within the reduced feature space, potentially improving classification performance.
In summary, the combination of PCA and LDA can be beneficial in situations where both noise reduction and class separability are important considerations.

Boosting:

Bias-Variance Tradeoff: [Bias-Variance Trade Off - Machine Learning - GeeksforGeeks]



Boosting

Adaboost Algo:

Explain the algorithm line by line

① For $i=1$ to N (size of dataset) Initialize the data weight $w_1^{(i)} = \frac{1}{N}$

② For $t=1$ to T

ⓐ Find a classifier $h_t(x)$ by minimizing the weighted error fn

$$J_t = \sum_{i=1}^{n} w_t^{(i)} \times I\left(y^{(i)} \neq h_t(x^{(i)})\right)$$

misclassification

multiplying the weights of them

ⓑ Find the weighted error of $h_t(x)$:

$$\epsilon_t = \frac{\sum_{i=1}^{N} w_t^{(i)} \times I\left(y^{(i)} \neq H_t(x^{(i)})\right)}{\sum_{i=1}^{N} w_t^{(i)}}$$

and a new component is assigned votes based on it's error:

$$\alpha_t = \ln\left((1-\epsilon_t)/\epsilon_t\right)$$

We will asign some points to the classifier. We know that, in between the "good classifier that misclassify" and "bad classifier that misclassify", the first one is bad, so we need to give priority to that one. After merging all classifier voting is done and the one with higher α value will get higher priority.

For weak classifier; $\epsilon_t = 0.5$

$\ln(1) = 0$

So, $\alpha = 0$; voting power is reduced for weak classifier.

For strong classifier; α is higher.

(c) The normalized weights are updated:

$$w_{t+1}^{(i)} = w_t^{(i)} e^{\alpha_t I\left(y^{(i)} \neq h_t(x^{(i)})\right)}$$

$w_t^{(i)}$: old weight; $I(y^{(i)} \neq h_t(x^{(i)}))$: misclassified data

e was taken to reduce error exponentially.

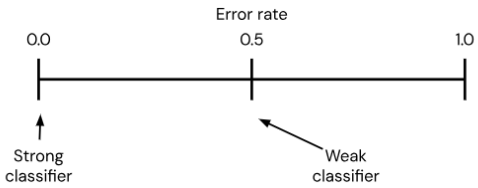
Here, For weak classifier, the weight will stay same.
For strong " " " will be updated.

3) Combined classifier $\hat{y} = \text{sign}\left(H_T(x)\right)$ where $H_T(x) = \sum_{t=1}^{M} \alpha_t h_t(x)$

Integer-43:

e) Write the differences between bagging and boosting methods of ensemble learning. Define weak and strong classifiers. How does the AdaBoost algorithm assign weights to different classifiers so that combined classification error is minimized?

- Bagging is a method of merging the same type of predictions. Boosting is a method of merging different types of predictions.
- Bagging decreases variance, not bias, and solves over-fitting issues in a model. Boosting decreases bias, not variance.
- In Bagging, each model receives an equal weight. In Boosting, models are weighed based on their performance.
- Models are built independently in Bagging. New models are affected by a previously built model's performance in Boosting.
- In Bagging, training data subsets are drawn randomly with a replacement for the training dataset. In Boosting, every new subset comprises the elements that were misclassified by previous models.
- Bagging is usually applied where the classifier is unstable and has a high variance. Boosting is usually applied where the classifier is stable and simple and has high bias.



Weak Classifier:
A weak classifier is a model that performs slightly better than random chance but is not highly accurate on its own. It may make errors and is often considered a simple or basic model. Weak classifiers are typically used in ensemble methods like boosting, where multiple weak classifiers are combined to create a strong classifier.

Strong Classifier:
A strong classifier is a model that is highly accurate and performs well on its own. It has the ability to capture complex patterns and make accurate predictions. Strong classifiers are often the result of combining multiple weak classifiers or through the use of sophisticated algorithms.

The algorithm starts by assigning equal weights to all training instances. In each iteration, a weak learner is trained on the weighted training data, and its classification error is computed. The weight of each instance is then updated based on whether it was classified correctly or not. The weight of misclassified instances is increased, while the weight of correctly classified instances is decreased. This process is repeated for a fixed number of iterations or until the desired accuracy is achieved. The final classifier is a weighted combination of all the weak learners, where the weights are proportional to their accuracy.

Template Matching:

(b) Explain Bellman's optimality principle. Find the edit distance between the word "pattern" and its misspelled version "petern". What is the sequence of edit operations to achieve them?