1. Questions based on SVM (2 Marks each):

a. Explain the formulation of SVM as a constrained quadratic optimization problem. Why is quadratic programming used in SVM, and what does it optimize?

1 mark: Explanation of SVM formulation as a quadratic optimization problem.

- Objective function: Minimizing ½ $\| w \|^2$.
- Constraints: $y_i(w \cdot x_i+b) >= 1, \forall i$.

1 mark: Explanation of why quadratic programming (QP) is used and what it optimizes.

- QP ensures the convexity of the problem, guaranteeing a global minimum.
- Optimization balances margin maximization and misclassification penalty.

b. Define the primal and dual formulations in SVM. What is the role of the soft margin in SVM, and how does it handle non-linearly separable data?

1 mark: Definition and formulation of primal and dual problems.

- Primal: Directly minimizes ½ $\| w \|^2$ subject to constraints.
- Dual: Reformulates the problem in terms of Lagrange multipliers to handle constraints indirectly.

1 mark: Role of the soft margin and how it handles non-linearly separable data.

- Introduces slack variables (ξi) to relax constraints.
- Balances margin maximization and penalty for misclassification via CCC-parameter.

c. Describe the role of a kernel in SVM. Provide an example of a kernel function and illustrate how it helps in mapping data to higher-dimensional spaces.

1 mark: Explanation of how kernels map data to higher-dimensional spaces.

- Avoids explicit computation of transformations using the kernel trick.

1 mark: Any 1 example and illustration of a kernel function (e.g., polynomial or Gaussian/RBF kernel).

1. Polynomial kernel

It is popular in image processing.
Equation is:

$$k(\mathbf{x_i}, \mathbf{x_j}) = (\mathbf{x_i} \cdot \mathbf{x_j} + 1)^d$$

Polynomial kernel equation

where d is the degree of the polynomial.

2. Gaussian kernel

It is a general-purpose kernel; used when there is no prior knowledge about the data. Equation is:

$$k(x, y) = \exp\left(-\frac{\|x - y\|^2}{2\sigma^2}\right)$$

Gaussian kernel equation

3. Gaussian radial basis function (RBF)

It is a general-purpose kernel; used when there is no prior knowledge about the data.
Equation is:

$$k(\mathbf{x_i}, \mathbf{x_j}) = \exp(-\gamma \|\mathbf{x_i} - \mathbf{x_j}\|^2)$$

Gaussian radial basis function (RBF)

4. Laplace RBF kernel

It is general-purpose kernel; used when there is no prior knowledge about the data.
Equation is:

$$k(x, y) = \exp\left(-\frac{\|x - y\|}{\sigma}\right)$$

Laplace RBF kernel equation

5. Hyperbolic tangent kernel

We can use it in neural networks.
Equation is:

$$k(\mathbf{x_i}, \mathbf{x_j}) = \tanh(\kappa \mathbf{x_i} \cdot \mathbf{x_j} + c)$$

Hyperbolic tangent kernel equation

, for some (not every) k>0 and c<0.

6. Sigmoid kernel

We can use it as the proxy for neural networks. Equation is

$$k(x, y) = \tanh(\alpha x^T y + c)$$

Sigmoid kernel equation

d. List and explain at least two key properties that a function must satisfy to be used as a kernel function in SVM.

1 mark each for any 2 properties

1. **Symmetry:**
   - The kernel function $K(\mathbf{x}, \mathbf{z})$ must satisfy $K(\mathbf{x}, \mathbf{z}) = K(\mathbf{z}, \mathbf{x})$ for all $\mathbf{x}$ and $\mathbf{z}$ in the input space.
2. **Positive Semi-Definiteness:**
   - The kernel matrix K, defined such that $K_{ij} = K(\mathbf{x}_i, \mathbf{x}_j)$ for a set of data points $\{\mathbf{x}_i\}$, must be positive semi-definite.
   - This ensures that $\sum_{i,j} \alpha_i \alpha_j K(\mathbf{x}_i, \mathbf{x}_j) \geq 0$ for any set of coefficients α_i.
3. **Mercer's Theorem Compliance:**
   - The kernel function should satisfy Mercer's condition, which states that it corresponds to a valid inner product in some (potentially high-dimensional) feature space.

e. Given a labeled dataset, explain how the SVM is trained algorithmically.

Mark accordingly

Steps to Train an SVM Algorithmically:

1. **Preprocess Data:** Standardize features for consistent scaling.
2. **Define Objective:** Minimize margin width with slack variables for misclassifications (soft margin).
3. **Kernel Transformation:** Use kernel functions for non-linear separable data.
4. **Formulate QP Problem:** Optimize Lagrange multipliers subject to constraints.
5. **Solve Optimization:** Use algorithms like SMO to find optimal multipliers.
6. **Identify Support Vectors:** Points with non-zero multipliers ($\alpha_i > 0$).
7. **Compute Parameters:** Derive $\mathbf{w}$ and bias b from support vectors.
8. **Make Predictions:** Use $f(\mathbf{x}) = \sum \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}) + b$.
9. **Tune Hyperparameters:** Adjust C and kernel parameters with cross-validation.

---

2. Questions based on ANN (2 Marks each)

a. Describe how a perceptron works as a linear classifier. How does it decide the class of an input?

A perceptron works as a linear classifier by performing the following steps:

- **Input and Weights**: The perceptron receives input features and assigns weights to them.
- **Weighted Sum**: It computes the weighted sum of the inputs:
  $\text{sum} = w_1x_1 + w_2x_2 + \cdots + w_nx_n + b$ where $w_1, w_2, \ldots, w_n$ are the weights, $x_1, x_2, \ldots, x_n$ are the input features, and b is the bias term.
- **Activation Function**: The weighted sum is passed through an activation function (usually a step function) that outputs a decision:
  - If the sum is greater than a threshold (typically zero), the output is one class (e.g., 1).
  - If the sum is less than or equal to the threshold, the output is the other class (e.g., 0).
- **Classification Decision**: The perceptron classifies the input into one of two classes (binary classification) based on the sign of the weighted sum.

The perceptron adjusts its weights during training to minimize classification errors using a learning rule, typically gradient descent or other optimization methods.

b. Explain how a neuron can be seen as an extension of a logistic regression unit. What is the main difference in their computation and application?

1 mark for explanation 0.5 each for any 2 differences

Neuron as an Extension of Logistic Regression:

- A neuron can be seen as a generalization of logistic regression, as both compute a weighted sum of inputs followed by a nonlinear activation function (often sigmoid or ReLU).
- In logistic regression, the output is a probability for binary classification, while a neuron can be used in more complex neural networks for multiple classes or more advanced tasks.

Main Difference:

- **Computation**:
  - Logistic regression applies a sigmoid function directly to the weighted sum.
  - A neuron can use different activation functions (sigmoid, ReLU, etc.), providing more flexibility in complex tasks.
- **Application**:

- Logistic regression is typically used for simpler binary classification tasks.
- Neurons are used as building blocks in neural networks, which can handle multi-layered, complex tasks.

c. Why can a single perceptron not learn a nonlinear pattern, such as the XOR function? Explain with reference to the decision boundary.

1 mark each for any 2 points

Linear Decision Boundary: A perceptron can only create a linear decision boundary to separate classes. This means it can only classify data that is linearly separable.

XOR Pattern is Nonlinear: The XOR function has a nonlinear decision boundary. The XOR outputs 1 for inputs (0,1) and (1,0), and 0 for inputs (0,0) and (1,1), which cannot be separated by a single straight line.

No Linear Separation: There is no single straight line that can divide the XOR's 1s from its 0s in the input space, because the data points from the different classes are interspersed in a way that requires a nonlinear separation.

Inability to Model Complex Patterns: Since the perceptron can only learn a linear boundary, it cannot capture the complexity of the XOR function, which requires a more intricate decision surface (e.g., a curve or multiple boundaries).

d. Outline the backpropagation algorithm for training a multi-layer neural network. Describe the role of gradient descent in updating weights.

Backpropagation Steps:

1. **Forward Pass**: Compute the output of each neuron layer by layer.
2. **Compute Loss**: Calculate the error between the predicted and actual outputs.
3. **Backward Pass**: Calculate the gradient of the loss with respect to each weight using the chain rule.
4. **Weight Update**: Adjust weights by subtracting a fraction of the gradient (learning rate) from each weight.

Role of Gradient Descent:

- Gradient descent minimizes the loss function by updating weights iteratively in the direction that reduces the error.

e. What is the role of the hidden layer in a neural network? What does the hidden layer "hide"?

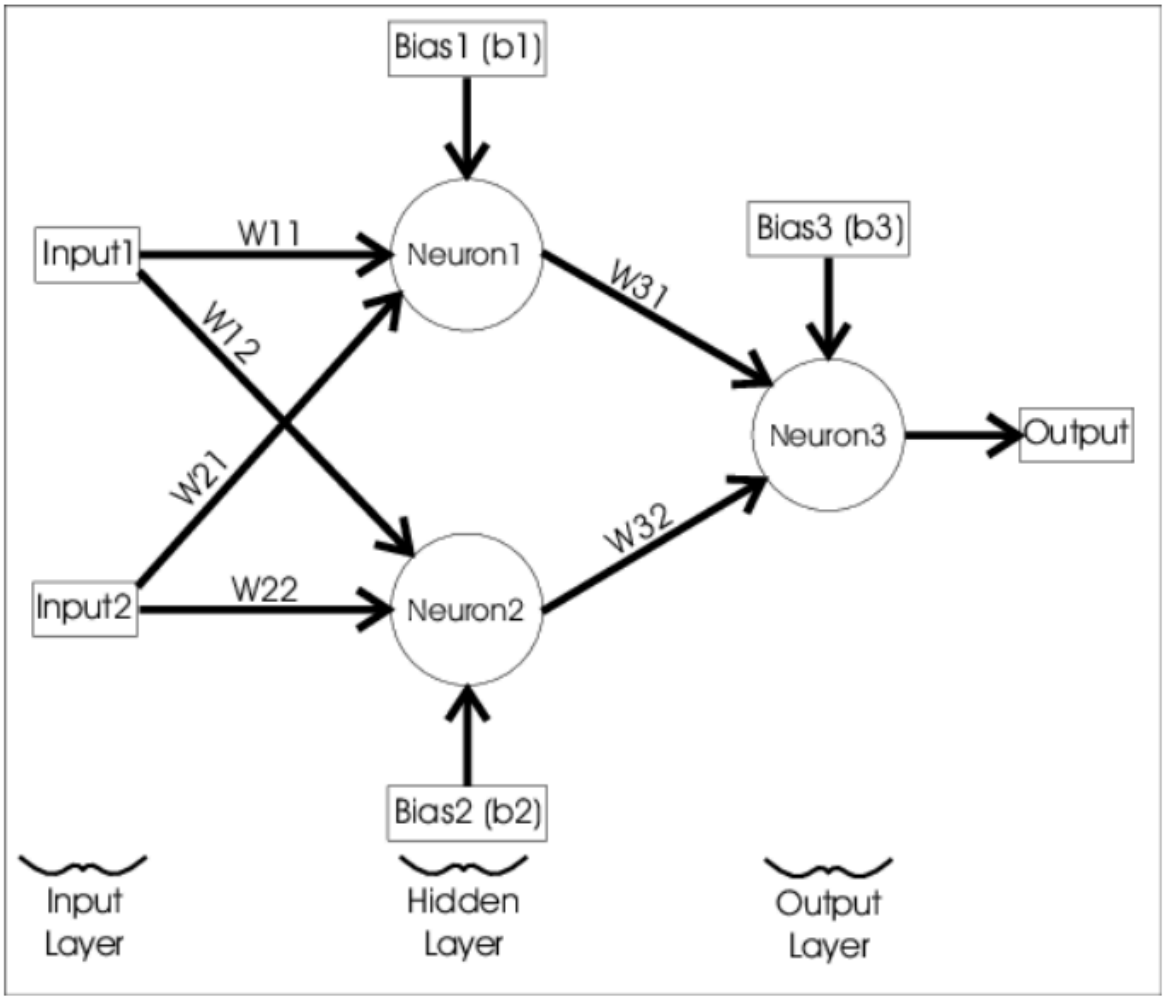
Purpose:

- The hidden layer transforms inputs into useful representations for the network to learn complex patterns.

What it "Hides":

- The hidden layer hides the internal complexity and intermediate steps that help map input to output in a more abstract form.

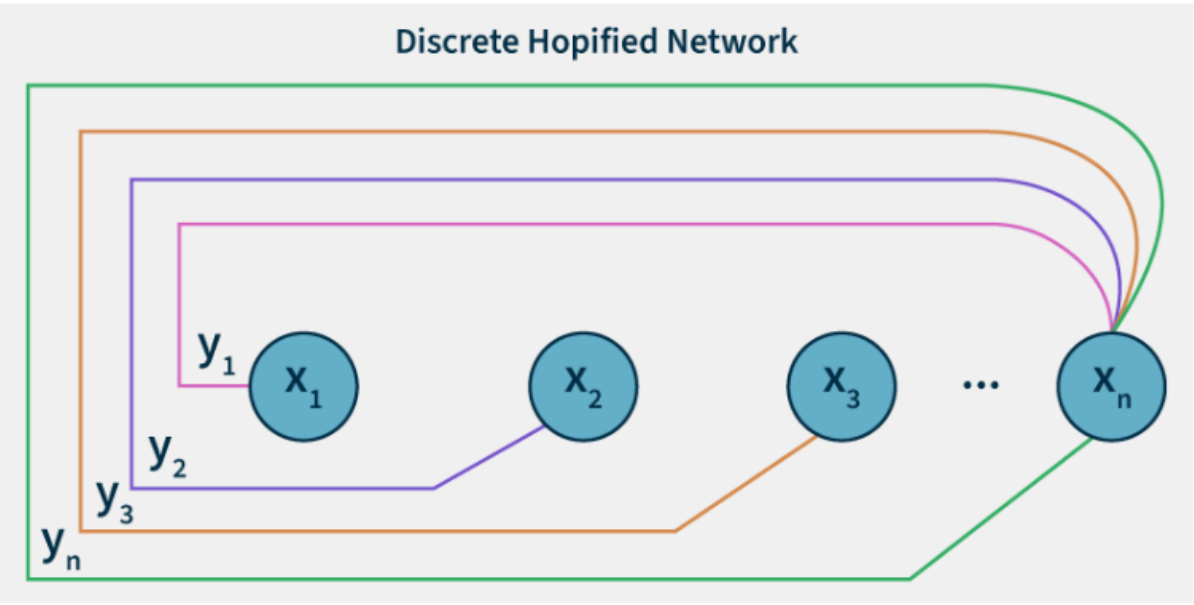
f. Implement the backpropagation algorithm for a neural network to learn the XOR function with two inputs. Show the steps briefly.



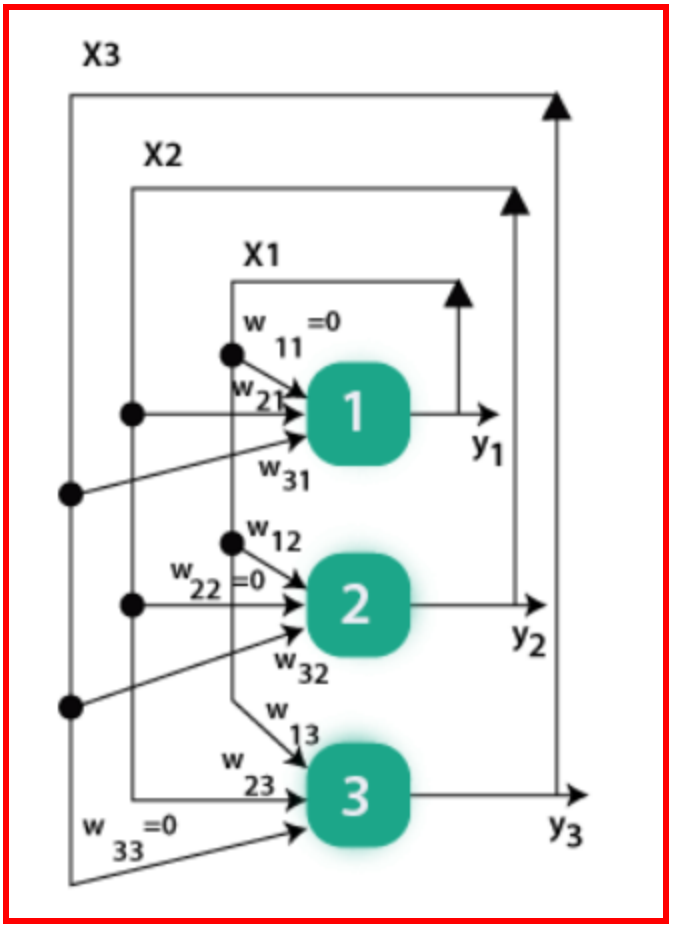
The Neural Network Model to solve the XOR Logic (from: https://stopsmokingaids.me/)

g. Explain the architecture of a basic Hopfield network and its use for auto-association. What is meant by "fundamental memory"?

- Fully connected, symmetric, recurrent network with a set of binary neurons.
- Each neuron has an output of 1 or -1 (binary states).



Discrete Hopfield Network Architecture



Use for Auto-association:

- The network is used for pattern completion: when part of a pattern is given, it recalls the full pattern.

A "fundamental memory" is a stable state that the network can store and recall, representing a stored pattern.

h. Compare the merits and demerits of using neural networks versus linear and logistic regression for solving ML problems.

Merits of Neural Networks:

- Can model complex, non-linear relationships.
- Better for high-dimensional data and large datasets.

Demerits of Neural Networks:

- Computationally expensive and requires more data.
- Difficult to interpret.

Merits of Linear/Logistic Regression:

- Simple, interpretable, and fast.
- Works well for linearly separable data.

Demerits of Linear/Logistic Regression:

- Limited to linear relationships.
- Cannot capture complex patterns in data.

i. To use the identical network (two hidden layers with input and output) for regression, binary, and multi-class classification, what minimum changes are required in the network to deal with all the above three kinds of problems? Also, define the error/loss function in all three cases with justification.

Minimum Changes for Regression, Binary, and Multi-class Classification

- **Minimum Changes**:
  - **Regression**: Use linear activation in the output layer.
  - **Binary Classification**: Use sigmoid activation in the output layer and binary cross-entropy loss.
  - **Multi-class Classification**: Use softmax activation in the output layer and categorical cross-entropy loss.
- **Error/Loss Functions**:
  - **Regression**: Mean Squared Error (MSE) – suitable for continuous output.
  - **Binary Classification**: Binary Cross-Entropy – measures error between predicted and true probabilities.
  - **Multi-class Classification**: Categorical Cross-Entropy – generalizes binary cross-entropy for multi-class outputs.

Question 3. Comparison of Classifiers (5 Marks)

Two classifiers were evaluated on a test dataset with the following results:

Metric	Classifier-I	Classifier-II
True Positives (TP)	50	55
True Negatives (TN)	40	35

False Positives (FP)	10	15
False Negatives (FN)	20	15

- Compute the Accuracy, Precision, and Recall of both classifiers. Select the best

Criteria	Marks	Comments/Guidelines
1. Correct Calculation of Accuracy (1 Mark)	1	**Accuracy** = $\frac{TP+TN}{TP+TN+FP+FN}$
		- Classifier-I: $\frac{50+40}{50+40+10+20} = \frac{90}{120} = 0.75$
		- Classifier-II: $\frac{55+35}{55+35+15+15} = \frac{90}{120} = 0.75$
2. Correct Calculation of Precision (1 Mark)	1	**Precision** = $\frac{TP}{TP+FP}$
		- Classifier-I: $\frac{50}{50+10} = \frac{50}{60} = 0.833$
		- Classifier-II: $\frac{55}{55+15} = \frac{55}{70} = 0.786$
3. Correct Calculation of Recall (1 Mark)	1	**Recall** = $\frac{TP}{TP+FN}$
		- Classifier-I: $\frac{50}{50+20} = \frac{50}{70} = 0.714$
		- Classifier-II: $\frac{55}{55+15} = \frac{55}{70} = 0.786$
4. Comparison and Selection of Best Classifier (1 Mark)	1	- **Classifier-II** is selected as the best classifier because it has a higher recall (0.786) compared to Classifier-I (0.714).
		- Although both classifiers have the same accuracy (0.75), Classifier-II performs better in identifying true positives (higher recall).
5. Justification and Clear Explanation (1 Mark)	1	- The reasoning for selecting Classifier-II should include the recognition that recall is more important in many applications, as it indicates the classifier's ability to identify all actual positives.
		- Mention that precision, while also important, is secondary to recall in this case, as we are prioritizing reducing false negatives.

Question 4. Principal Component Analysis (6 Marks)

- What is the curse of dimensionality? What is the main objective of Principal Component Analysis (PCA)?
- Consider a small dataset of 3 observations and 2 features. Find the first principal component.

Observation	Feature 1 (x_1)	Feature 2 (x_2)
1	2	3
2	3	4
3	1	2

Curse of Dimensionality (Theoretical Explanation)

- **Definition**: The curse of dimensionality refers to the challenges that arise when working with high-dimensional data. As the number of features (dimensions) increases:
  - The volume of the feature space increases exponentially.
  - The data becomes sparse, making it difficult to find meaningful patterns or clusters.
  - Distance metrics (e.g., Euclidean distance) become less effective, as points tend to be far from each other in high-dimensional spaces.
- **Implications**:
  - Models may overfit due to the sparsity of data.
  - Increased computational cost due to the need to process more dimensions.
  - Difficulty in visualizing and understanding high-dimensional relationships.

Objective of Principal Component Analysis (PCA)

- **Reduce Dimensionality**: PCA aims to reduce the number of features (dimensions) while preserving as much variance (information) as possible.
- **Find Principal Components**: PCA identifies new orthogonal axes (principal components) that maximize the variance in the data, which can be used to represent the data in a lower-dimensional space.
- **Improve Interpretability and Efficiency**: By reducing dimensions, PCA can help make the data easier to visualize and speed up machine learning algorithms.

2.1. Calculate the Mean of Each Feature:

For each feature, calculate the mean:

- Mean of Feature 1 ($\bar{x}_1$):

$$\bar{x}_1 = \frac{2+3+1}{3} = 2$$

- Mean of Feature 2 ($\bar{x}_2$):

$$\bar{x}_2 = \frac{3+4+2}{3} = 3$$

2.2. Subtract the Mean from Each Data Point (Centering the Data):

- Subtract the mean of Feature 1 from each of the data points in Feature 1.
- Subtract the mean of Feature 2 from each of the data points in Feature 2.

Observation	Feature 1 (x_1)	Feature 2 (x_2)	$x_1 - \bar{x}_1$	$x_2 - \bar{x}_2$
1	2	3	2 - 2 = 0	3 - 3 = 0
2	3	4	3 - 2 = 1	4 - 3 = 1
3	1	2	1 - 2 = -1	2 - 3 = -1

2.3. Compute the Covariance Matrix:

The formula for the covariance matrix is:

$$\text{Cov}(x_1, x_2) = \frac{1}{n-1}\sum_{i-1}^{n}(x_{1i} - \bar{x}_1)(x_{2i} - \bar{x}_2)$$

For two features, the covariance matrix $\mathbf{C}$ is a 2x2 matrix:

$$\mathbf{C} = \begin{bmatrix} \text{Cov}(x_1, x_1) & \text{Cov}(x_1, x_2) \\ \text{Cov}(x_2, x_1) & \text{Cov}(x_2, x_2) \end{bmatrix}$$

Let's calculate each element of the covariance matrix:

- $\text{Cov}(x_1, x_1)$: The variance of Feature 1.

$$\text{Cov}(x_1, x_1) = \frac{1}{3-1}\left[(0)^2 + (1)^2 + (-1)^2\right] = \frac{1}{2}[0 + 1 + 1] = 1$$

- $\text{Cov}(x_1, x_2)$: The covariance between Feature 1 and Feature 2.

$$\text{Cov}(x_1, x_2) = \frac{1}{3-1}\left[(0)(0) + (1)(1) + (-1)(-1)\right] = \frac{1}{2}[0 + 1 + 1] = 1$$

- $\text{Cov}(x_2, x_1)$: Since covariance is symmetric, $\text{Cov}(x_2, x_1) = \text{Cov}(x_1, x_2) = 1$.
- $\text{Cov}(x_2, x_2)$: The variance of Feature 2.

$$\text{Cov}(x_2, x_2) = \frac{1}{3-1}\left[(0)^2 + (1)^2 + (-1)^2\right] = \frac{1}{2}[0 + 1 + 1] = 1$$

Thus, the covariance matrix $\mathbf{C}$ is:

$$\mathbf{C} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$$

Step 3: Compute the Eigenvalues and Eigenvectors

To find the principal components, we need to compute the **eigenvalues** and **eigenvectors** of the covariance matrix.

The **characteristic equation** for eigenvalues λ is:

$$\det(\mathbf{C} - \lambda \mathbf{I}) = 0$$

Where $\mathbf{I}$ is the identity matrix.

For the covariance matrix $\mathbf{C} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$, we have:

$$\begin{vmatrix} 1-\lambda & 1 \\ 1 & 1-\lambda \end{vmatrix} = 0$$

Expanding this determinant:

$$(1-\lambda)(1-\lambda) - 1 = 0$$

$$(1-\lambda)^2 - 1 = 0$$

$$1 - 2\lambda + \lambda^2 - 1 = 0$$

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

$$\lambda(\lambda - 2) = 0$$

Thus, the eigenvalues are $\lambda_1 = 2$ and $\lambda_2 = 0$.

Step 3.1: Find the Eigenvectors

- For $\lambda_1 = 2$, solve $(\mathbf{C} - 2\mathbf{I})\mathbf{v} = 0$:

$$\begin{bmatrix} 1-2 & 1 \\ 1 & 1-2 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = 0$$

$$\begin{bmatrix} -1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = 0$$

Solving this system gives the eigenvector corresponding to $\lambda_1 = 2$:

$$v_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

Thus, the first principal component (the eigenvector corresponding to the largest eigenvalue) is:

$$\mathbf{PC}_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

Question 5. Generative vs. Discriminative Models

a. Explain how a generative model differs from a discriminative model in terms of its approach to classification.

b. Why might generative models like GDA not be but Naive Bayes-based models like the Multinomial event model or Multivariate Bernoulli model be useful for designing a spam classifier?

- For a spam classifier that uses the Multinomial event model with Laplace smoothing or Naive Bayes with Laplace smoothing, design a toy spam filter with the following sample email data:

Words: “Hello,” “Dear,” “Got,” “Lottery,” “Money.”
Data:

Label (Y)	Hello	Dear	Got	Lottery	Money	Total Words	Total Mails
0	20	15	2	1	2	40	20
1	5	2	0	8	5	20	10

PART A 3 Marks

Generative Models:

- **Definition**: A generative model models the joint probability distribution $P(X, Y)$, i.e., it tries to model how the data is generated for each class.
- **Approach**: It learns how the data is generated from each class and then uses Bayes' theorem to compute the posterior probability $P(Y|X)$.
- **Example**: Gaussian Discriminant Analysis (GDA) and Naive Bayes.
- **How it Works**: In GDA, for each class, we model the distribution of the features X (e.g., using Gaussian distributions). Then, to classify a new data point, we compute the likelihood of the data point for each class, combine it with the prior of each class, and predict the class with the highest posterior probability.

Discriminative Models:

- **Definition**: A discriminative model models the conditional probability $P(Y|X)$, i.e., it directly models the decision boundary between the classes.
- **Approach**: It focuses on learning the boundaries that separate the different classes, rather than modeling how data is generated.
- **Example**: Logistic Regression, Support Vector Machines (SVM).
- **How it Works**: It learns a function $f(X)$ that directly maps input features X to the output class Y, without modeling the distribution of t eatures for each class.

PART B 3 Marks

Generative Models (e.g., GDA):

- GDA assumes a Gaussian distribution for the features (e.g., word frequencies in spam classification), which might not hold in text data, as word occurrences are often discrete and non-Gaussian in nature.
- For example, the distribution of word counts (e.g., how often "Lottery" appears in spam emails) may not fit well to Gaussian assumptions, leading to suboptimal performance.

Naive Bayes-Based Models:

- **Multinomial Event Model**: This is a generative model specifically suited for discrete count data (such as word occurrences). It models the probability of each word given the class label, assuming words are generated independently. This is a good fit for text classification, where the presence of a word in an email is independent of other words, given the class (spam or not).
- **Multivariate Bernoulli Model**: This model works well for binary features (whether a word appears or not). It is also a good fit for text classification where the goal is to determine if certain keywords appear in an email.

Why Naive Bayes-Based Models Are Useful for Spam Classifiers:

- **Text Data Nature**: Words in emails are discrete, and the Multinomial event model or Multivariate Bernoulli model is well-suited for such data. Each word occurrence or absence can be treated as a feature, and these models naturally work with the frequency of occurrences of words.
- **Simplicity and Efficiency**: Naive Bayes models are relatively simple and computationally efficient, making them ideal for spam classification, especially with limited training data.
- **Laplace Smoothing**: This technique prevents zero probabilities for words not seen in the training data, which is a common issue when dealing with large vocabularies.

PART C 4 Marks

Event Model with Laplace smoothing.

1. Define the Problem:

We want to classify emails as spam or non-spam based on the presence/absence of certain words.

- **Words to Consider**: "Hello," "Dear," "Got," "Lottery," "Money"
- **Classes**: Spam (1), Not Spam (0)

2. Prepare the Data (Training Data)

Label (Y)	Hello	Dear	Got	Lottery	Money	Total Words	Total Mails
0 (Not Spam)	20	15	2	1	2	40	20
1 (Spam)	5	2	0	8	5	20	10

- **Total Words in Dataset**: 40 (non-spam) + 20 (spam) = 60
- **Total Mails in Dataset**: 20 (non-spam) + 10 (spam) = 30

Solution with decimals:

3. Calculate the Probabilities for Each Word

To classify a new email, we calculate the probability of each word given the class label. This is done using the **Multinomial event model** with **Laplace smoothing** (adding 1 to avoid zero probabilities).

For each class $Y = 0$ (non-spam) and $Y = 1$ (spam), we compute:

$$P(\text{word}|Y) = \frac{\text{count of word in class Y} + 1}{\text{total words in class Y} + V}$$

Where V is the size of the vocabulary (the number of unique words). Here, $V = 5$ (since we have 5 words).

For Non-Spam (Y = 0):

- $P(\text{Hello}|0) = \frac{20+1}{40+5} = \frac{21}{45} = 0.4667$
- $P(\text{Dear}|0) = \frac{15+1}{40+5} = \frac{16}{45} = 0.3556$
- $P(\text{Got}|0) = \frac{2+1}{40+5} = \frac{3}{45} = 0.0667$
- $P(\text{Lottery}|0) = \frac{1+1}{40+5} = \frac{2}{45} = 0.0444$
- $P(\text{Money}|0) = \frac{2+1}{40+5} = \frac{3}{45} = 0.0667$

For Spam (Y = 1):

- $P(\text{Hello}|1) = \frac{5+1}{20+5} = \frac{6}{25} = 0.24$
- $P(\text{Dear}|1) = \frac{2+1}{20+5} = \frac{3}{25} = 0.12$
- $P(\text{Got}|1) = \frac{0+1}{20+5} = \frac{1}{25} = 0.04$
- $P(\text{Lottery}|1) = \frac{8+1}{20+5} = \frac{9}{25} = 0.36$
- $P(\text{Money}|1) = \frac{5+1}{20+5} = \frac{6}{25} = 0.24$

4. Calculate the Prior Probabilities

The prior probabilities are simply the proportion of each class in the dataset.

- $P(Y = 0) = \frac{20}{30} = 0.6667$ (Not Spam)
- $P(Y = 1) = \frac{10}{30} = 0.3333$ (Spam)

5. Classify a New Email

For example, suppose we want to classify a new email containing the words "Lottery" and "Money."

- For Y = 0 (Not Spam):

$$P(Y = 0|\text{Lottery, Money}) \propto P(Y = 0) \times P(\text{Lottery}|0) \times P(\text{Money}|0)$$

$$= 0.6667 \times 0.0444 \times 0.0667 = 0.00196$$

- For Y = 1 (Spam):

$$P(Y = 1|\text{Lottery, Money}) \propto P(Y = 1) \times P(\text{Lottery}|1) \times P(\text{Money}|1)$$

$$= 0.3333 \times 0.36 \times 0.24 = 0.026$$

Since $P(Y = 1|\text{Lottery, Money}) > P(Y = 0|\text{Lottery, Money})$, we classify this email as **Spam**.

Solution with fractions intact:

Step 2: Calculate Probabilities for Each Word

For **Non-Spam (Y = 0)**:

- Total word occurrences in non-spam = 20 + 15 + 2 + 1 = 38
- Probabilities for each word:

$$P(Hello|Y=0) = \frac{20+1}{38+5} = \frac{21}{43} = \frac{21}{43}$$

$$P(Dear|Y=0) = \frac{15+1}{38+5} = \frac{16}{43} = \frac{16}{43}$$

$$P(Got|Y=0) = \frac{2+1}{38+5} = \frac{3}{43} = \frac{3}{43}$$

$$P(Lottery|Y=0) = \frac{1+1}{38+5} = \frac{2}{43} = \frac{2}{43}$$

$$P(Money|Y=0) = \frac{2+1}{38+5} = \frac{3}{43} = \frac{3}{43}$$

For **Spam (Y = 1)**:

- Total word occurrences in spam = 5 + 2 + 0 + 8 + 5 = 20
- Probabilities for each word:

$$P(Hello|Y=1) = \frac{5+1}{20+5} = \frac{6}{25} = \frac{6}{25}$$

$$P(Dear|Y=1) = \frac{2+1}{20+5} = \frac{3}{25} = \frac{3}{25}$$

$$P(Got|Y=1) = \frac{0+1}{20+5} = \frac{1}{25} = \frac{1}{25}$$

$$P(Lottery|Y=1) = \frac{8+1}{20+5} = \frac{9}{25} = \frac{9}{25}$$

$$P(Money|Y=1) = \frac{5+1}{20+5} = \frac{6}{25} = \frac{6}{25}$$

Step 3: Spam Classifier Decision (Bayes' Theorem)

For a new email with the words: "Hello", "Dear", "Got", "Lottery", "Money", we will calculate the **posterior probability** for both classes (spam and non-spam). The class with the higher posterior probability is the predicted label.

Bayes' Theorem for classification:

$$P(Y|X) = \frac{P(X|Y) \cdot P(Y)}{P(X)}$$

Since $P(X)$ is constant for both classes, we can ignore it for comparison.

Step 4: Calculate Likelihood for Each Class

For Non-Spam (Y = 0):

$$P(X|Y=0) = P(Hello|Y=0) \times P(Dear|Y=0) \times P(Got|Y=0) \times P(Lottery|Y=0) \times P(Money|Y=0)$$

$$P(X|Y=0) = \frac{21}{43} \times \frac{16}{43} \times \frac{3}{43} \times \frac{2}{43} \times \frac{3}{43} = \frac{21 \times 16 \times 3 \times 2 \times 3}{43^5}$$

For Spam (Y = 1):

$$P(X|Y=1) = P(Hello|Y=1) \times P(Dear|Y=1) \times P(Got|Y=1) \times P(Lottery|Y=1) \times P(Money|Y=1)$$

$$P(X|Y=1) = \frac{6}{25} \times \frac{3}{25} \times \frac{1}{25} \times \frac{9}{25} \times \frac{6}{25} = \frac{6 \times 3 \times 1 \times 9 \times 6}{25^5}$$

Step 5: Compare Posterior Probabilities

- The class with the higher $P(Y|X)$ will be selected as the predicted class.

---

6. (6 Marks) Attempt any 1

a. Explain the following:

- Gaussian Discriminant Analysis (GDA)
- Multivariate Bernoulli event model
- Multinomial event model

1. Gaussian Discriminant Analysis (GDA)

Overview:

Gaussian Discriminant Analysis (GDA) is a generative probabilistic classification model that assumes the features for each class are drawn from a Gaussian (normal) distribution. It models the conditional probability of a feature vector X given a class Y (i.e., $P(X|Y)$) and applies Bayes' Theorem to compute the posterior probability $P(Y|X)$.

Key Assumptions:

- Each class has its own Gaussian distribution for the features.
- Features for each class are independently normally distributed (within each class).

Steps in GDA:

1. **Modeling the Class Conditional Distribution:**
   For each class k, the features are assumed to be normally distributed:

$$P(X|Y=k) = \frac{1}{(2\pi)^{d/2}|\Sigma_k|^{1/2}} \exp\left(-\frac{1}{2}(X-\mu_k)^T\Sigma_k^{-1}(X-\mu_k)\right)$$

   where μ_k is the mean vector and Σ_k is the covariance matrix for class k, and d is the number of features.

2. **Applying Bayes' Theorem:**
   To compute the posterior probability for class k:

$$P(Y=k|X) = \frac{P(X|Y=k)P(Y=k)}{P(X)}$$

   where $P(Y=k)$ is the prior probability of class k and $P(X)$ is the marginal likelihood of X.

3. **Classification:**
   For a new sample X, we compute $P(Y=k|X)$ for each class and assign X to the class with the highest posterior probability.

2. Multivariate Bernoulli Event Model

Overview:

The Multivariate Bernoulli Event Model is a variation of Naive Bayes where the features (e.g., words) are binary (present or absent), and each feature follows a Bernoulli distribution. This model is particularly useful in text classification when each feature corresponds to the presence or absence of a word in a document.

Key Assumptions:

- Each feature x_i is binary (0 or 1), representing whether the i^{th} word is present in the document.
- The features are conditionally independent given the class.

Likelihood for a Document:

The likelihood of observing a particular document $X = (x_1, x_2, \ldots, x_n)$ given class $Y = k$ is modeled as:

$$P(X = x|Y = k) = \prod_{i=1}^{n} P(x_i|Y = k)$$

Where $P(x_i|Y = k)$ is the Bernoulli probability for word i in class k:

- $P(x_i = 1|Y = k) = p_k$
- $P(x_i = 0|Y = k) = 1 - p_k$

Classification:

For a new document, we compute the posterior probability for each class using Bayes' Theorem and select the class with the highest posterior.

3. Multinomial Event Model

Overview:

The Multinomial Event Model is another variation of Naive Bayes, but here, the features are discrete counts, typically representing the frequency of words in a document. The model assumes that the frequency of each word follows a multinomial distribution, and the features are conditionally independent given the class.

Key Assumptions:

- Each feature x_i represents the count of the i^{th} word in a document.
- The features are conditionally independent given the class.

Likelihood for a Document:

The likelihood of observing a document $X = (x_1, x_2, \ldots, x_n)$ given class $Y = k$ is:

$$P(X = x|Y = k) = \frac{(n_1 + n_2 + \cdots + n_n)!}{n_1!n_2! \ldots n_n!} \prod_{i=1}^{n} P(x_i|Y = k)^{x_i}$$

Where $P(x_i|Y = k)$ is the probability of word i appearing in class k, and x_i is the count of word i in the document.

Classification:

We use Bayes' Theorem to compute the posterior probability for each class and select the class with the highest posterior.

b. Explain the Gaussian Mixture Model (GMM). Write the algorithmic steps for the Expectation-Maximization algorithm for GMM.

Overview:

A **Gaussian Mixture Model (GMM)** is a probabilistic model that assumes the data is generated from a mixture of several Gaussian distributions. Each Gaussian component has its own mean and covariance, and the overall distribution is a weighted sum of these Gaussian components. GMMs are often used for clustering, where each data point belongs to one of the Gaussian components.

Key Assumptions:

- The data is generated from K Gaussian distributions.
- Each Gaussian has a weight π_k, which represents the probability of a data point being generated from the k^{th} Gaussian.

The probability density function of the GMM is:

$$P(X) = \sum_{k=1}^{K} \pi_k \mathcal{N}(X|\mu_k, \Sigma_k)$$

Where:

- μ_k is the mean of the k^{th} Gaussian.
- Σ_k is the covariance matrix of the k^{th} Gaussian.
- π_k is the weight of the k^{th} Gaussian component.

and the **M-step** (Maximization step). These steps are iterated until convergence.

1. **Initialization**:
   - Choose K initial guesses for the mean μ_k, covariance Σ_k, and weight π_k of each Gaussian component (e.g., using k-means for initialization).
2. **E-step**:
   - For each data point X_i, compute the responsibility γ_{ik}, which is the probability that data point X_i belongs to the k^{th} Gaussian:

$$\gamma_{ik} = \frac{\pi_k \mathcal{N}(X_i|\mu_k, \Sigma_k)}{\sum_{k-1}^{K} \pi_k \mathcal{N}(X_i|\mu_k, \Sigma_k)}$$

3. **M-step**:
   - Update the parameters of the Gaussian components:
     - **Means**:

$$\mu_k = \frac{\sum_{i-1}^{N} \gamma_{ik} X_i}{\sum_{i-1}^{N} \gamma_{ik}}$$

     - **Covariances**:

$$\Sigma_k = \frac{\sum_{i-1}^{N} \gamma_{ik} (X_i - \mu_k)(X_i - \mu_k)^T}{\sum_{i-1}^{N} \gamma_{ik}}$$

     - **Weights**:

$$\pi_k = \frac{1}{N} \sum_{i-1}^{N} \gamma_{ik}$$

4. **Convergence**:
   - Repeat the E-step and M-step until the log-likelihood converges (i.e., the parameter estimates do not change significantly between iterations).