# GDA Programming

## Load Necessary Libraries and Process Data

In [10]:
```
import numpy as np
import pandas as pd

def load_dataset(path, add_intercept=True):
    """Load dataset from CSV file."""
    data = pd.read_csv(path)
    X = data.iloc[:, 1:-1].values  # Exclude 'Id' and 'Species'
    y = data['Species'].values
    if add_intercept:
        X = np.c_[np.ones(X.shape[0]), X]  # Add intercept term
    return X, y
```

# GDA Class

The Gaussian Discriminant Analysis (GDA) algorithm is employed to determine whether a flower belongs to the "Iris Virginica" species. In this binary classification task, the label 1 represents "Iris Virginica," while the label 0 denotes any other species. The GDA algorithm is specifically designed for binary classification problems, requiring the class labels to be in a binary format (0 or 1).

# Overview of GDA Model Components

### 1. Gaussian Distributions for Each Class

The GDA model assumes that the data from each class is normally distributed. Therefore, the probability of the features ( x ) given the class label ( y ) is modeled as a multivariate Gaussian (normal) distribution:

$$p(x \mid y=0) = \frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu_0)^T\Sigma^{-1}(x-\mu_0)\right)$$

$$p(x \mid y=1) = \frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu_1)^T\Sigma^{-1}(x-\mu_1)\right)$$

Here, $\mu_0$ and $\mu_1$ are the mean vectors for classes 0 and 1, respectively, and $\Sigma$ is the shared covariance matrix.

### 2. Class Prior Probability ($\phi$)

The prior probability $\phi$ represents the fraction of the training examples that belong to class 1. It is modeled as a Bernoulli random variable:

$$p(y) = \phi^y (1 - \phi)^{1-y}$$

### 3. Log-Likelihood of the Data

The log-likelihood of the data is the logarithm of the joint probability of the inputs $x^{(i)}$ and outputs $y^{(i)}$:

$$\ell(\phi, \mu_0, \mu_1, \Sigma) = \sum_{i=1}^{n} \log p(x^{(i)}, y^{(i)}; \phi, \mu_0, \mu_1, \Sigma)$$

This can be expanded using the chain rule of probability:

$$\ell(\phi, \mu_0, \mu_1, \Sigma) = \sum_{i=1}^{n} \left[\log p(x^{(i)} \mid y^{(i)}; \mu_0, \mu_1, \Sigma) + \log p(y^{(i)}; \phi)\right]$$

## How These Components Come Together in Your Code

Now, let's map these equations to the code you're writing for the `fit` method:

### 1. Calculate Class Prior ($\phi$)

- **Goal**: Compute $\phi$, the fraction of training examples that belong to class 1. This is a simple mean calculation over the binary target vector `y_binary`.
- **Mathematical formula**:

$$\phi = \frac{1}{n} \sum_{i=1}^{n} \mathbb{1}\{y^{(i)} = 1\}$$

- **Code**: You will compute this using the binary target array, typically with `numpy` operations.

### 2. Calculate Means ($\mu_0, \mu_1$)

- **Goal**: Compute the mean of the input features $x$ for each class. This involves separating the data into two subsets: one for class 0 and one for class 1.
- **Mathematical formula**:

$$\mu_0 = \frac{1}{n_0} \sum_{i=1}^{n} \mathbb{1}\{y^{(i)} = 0\} x^{(i)}, \quad \mu_1 = \frac{1}{n_1} \sum_{i=1}^{n} \mathbb{1}\{y^{(i)} = 1\} x^{(i)}$$

- **Code**: You will filter the `x` data by `y_binary` to compute $\mu_0$ and $\mu_1$.

### 3. Calculate Covariance Matrix ($\Sigma$)

- **Goal**: Compute the covariance matrix $\Sigma$ using all the training examples. This matrix describes the variance and covariance of the features, assuming they share the same covariance across both classes.
- **Mathematical formula**:

$$\Sigma = \frac{1}{n} \sum_{i=1}^{n} (x^{(i)} - \mu_{y^{(i)}})(x^{(i)} - \mu_{y^{(i)}})^T$$

- **Code**: This involves calculating the deviations of each data point from its corresponding class mean and then computing the sum of these deviations.

### 4. Compute Parameters ($\theta$ and $\theta_0$)

- **Goal**: Derive the parameters that define the decision boundary:

$$\theta = \Sigma^{-1}(\mu_1 - \mu_0)$$

$$\theta_0 = -\frac{1}{2}(\mu_1^T \Sigma^{-1} \mu_1 - \mu_0^T \Sigma^{-1} \mu_0) + \log\left(\frac{\phi}{1-\phi}\right)$$

- **Explanation**: The parameters $\theta$ and $\theta_0$ define the linear decision boundary that separates the classes in the feature space.

# How to Proceed in Your Code

1. **First Step**: Convert the target variable `y` to binary values and calculate the class prior $\phi$.
2. **Next Steps**: Implement the calculations for the mean vectors $\mu_0$ and $\mu_1$.
3. **Further Steps**: Calculate the shared covariance matrix $\Sigma$.

In [25]:
```
class GDA:
    """Gaussian Discriminant Analysis."""
    def __init__(self, step_size=0.01, max_iter=10000, eps=1e-5,
                 theta_0=None, verbose=True):
        self.theta = theta_0
        self.step_size = step_size
        self.max_iter = max_iter
        self.eps = eps
        self.verbose = verbose

    def fit(self, x, y):
        """Fit a GDA model to training set."""

        # Step 1: Convert target variable y to binary values
        y_binary = np.where(y=="Iris-virginica",1,0)
        number_of_samples = len(y)

        # Step 2: Calculate class prior phi
        self.phi = 1/(number_of_samples)*np.sum(y_binary)#--Mathematically cor
        #phi = np.mean(y_binary)

        # Step 3: Filter the data by class labels
        mask_0 = y_binary==0 # Filter for class 0, not Iris-Virginica using th
        mask_1 = y_binary==1# Filter for class 1, Iris-Virginica using this ma

        x_class_0 = x[mask_0] #Filtered classes
        x_class_1 = x[mask_1] #Filtered for class


        # Step 4: Compute the mean vectors for each class
        self.mu0 = np.mean(x_class_0, axis=0) #Averaging along the rows of mas
        self.mu1 = np.mean(x_class_1, axis=0) #Averaging along the rows of mas

        # Step 5: Initialize the covariance matrix
        matrix_of_zeros = np.zeros((x.shape[1],x.shape[1])) # Matrix to accumu


        # Step 6: Iterate through each sample to compute the covariance matrix
        for i, row in enumerate(x):
        # Determine the sample's deviation from the respective class mean
            if mask_0[i]:
            # Sample belongs to class 0, use mu0
                matrix_of_zeros+=np.outer(row-self.mu0, row-self.mu0)

            elif mask_1[i]:
            # Sample belongs to class 1, use mu1
                matrix_of_zeros+=np.outer(row-self.mu1, row-self.mu1)

        self.covariance = matrix_of_zeros/number_of_samples
        self.covariance += np.eye(self.covariance.shape[0]) * 1e-5 # Regulariz

    def predict(self, x):
        """Make a prediction given new inputs x."""
        self.inverted_covariance = np.linalg.inv(self.covariance) #Invert the
        self.theta = self.inverted_covariance@(self.mu1-self.mu0) #Find the th
        self.theta0 = (-1/2)*((self.mu1.T)@self.inverted_covariance@self.mu1-(
```

```
        self.log_likelihood = x@self.theta+self.theta0#calculate the log-likel

        self.sigmoid = 1/(1+np.exp(-self.log_likelihood))#apply the sigmoid fu


         # Classify based on the probability threshold of 0.5
        predicted_class = (self.sigmoid >= 0.5).astype(int)

        return predicted_class
```

## Step 1: Convert Target $y$ to Binary Values

#### Objective

We need to transform the target variable $y$ into binary form. Since we're working with binary classification (e.g., predicting if the flower is "Iris-virginica" or not), let's assume that "Iris-virginica" is the positive class (class 1) and the rest are negative (class 0).

#### Why?

This step is crucial because GDA expects the target variable to be in binary format (0 or 1) to compute probabilities for the logistic function.

#### Implementation Thought Process

1. **Iterate over each label in $y$:**
    - Assign 1 if the label is "Iris-virginica," otherwise 0.

#### Checkpoint

- Ensure you have the transformed array $y_{\text{binary}}$ with values as 0 or 1.

## Step 2: Calculate Class Prior $\phi$

#### Mathematical Explanation

To calculate the class prior $\phi$, you need to determine the proportion of samples that belong to class 1 ("Iris-virginica").

**Formula:**

$$\phi = \frac{1}{n} \sum_{i=1}^{n} \mathbb{1}\{y^{(i)} = 1\}$$

Where:

- $n$ is the total number of samples in your dataset.

- $\mathbb{1}\{y^{(i)} = 1\}$ is an indicator function that outputs 1 when $y^{(i)} = 1$ (meaning "Iris-virginica") and 0 otherwise.

This formula calculates the fraction of the dataset where the label is `1`.

#### Conceptual Steps

1. **Count the Number of Positive Class Labels:**
   - Sum the total number of samples where $y = 1$.
2. **Divide by the Total Number of Samples:**
   - This gives you the proportion of samples that are labeled as "Iris-virginica".

## Step 3: Calculate Mean Vectors $\mu_0$ and $\mu_1$

#### Objective

The mean vectors $\mu_0$ and $\mu_1$ represent the average values of the input features $x$ for each class:

- $\mu_0$ is the mean vector for class 0 (not "Iris-virginica").
- $\mu_1$ is the mean vector for class 1 ("Iris-virginica").

These mean vectors help define the Gaussian distribution for each class, which is a core part of the GDA model.

#### Mathematical Formulas

$$\mu_0 = \frac{1}{n_0} \sum_{i=1}^{n} \mathbb{1}\{y^{(i)} = 0\}\, x^{(i)}, \quad \mu_1 = \frac{1}{n_1} \sum_{i=1}^{n} \mathbb{1}\{y^{(i)} = 1\}\, x^{(i)}$$

Where:

- $n_0$ is the number of samples where $y = 0$ (not "Iris-virginica").
- $n_1$ is the number of samples where $y = 1$ ("Iris-virginica").
- $\mathbb{1}\{y^{(i)} = 0\}$ and $\mathbb{1}\{y^{(i)} = 1\}$ are indicator functions that pick out samples belonging to class 0 or class 1, respectively.
- $x^{(i)}$ is the feature vector for the $i$-th sample.

#### Conceptual Steps

1. **Filter the Data:**
   - Separate the data $x$ into two subsets: one for samples where $y = 0$ (class 0) and another for samples where $y = 1$ (class 1).
2. **Calculate the Mean for Each Subset:**
   - Compute the mean vector for each subset. The mean vector $\mu_0$ should be the average of all samples in class 0, and $\mu_1$ should be the average of all samples in class 1.

### Implementation Guidance

1. **Filter `x` by `y_binary`:**
    - Use logical indexing or masking to create two subsets: `x` where `y_binary == 0` and `x` where `y_binary == 1`.
2. **Compute the Means:**
    - Calculate the mean of each subset along the appropriate axis (e.g., along the rows if `x` is a 2D array).

## Step 4: Calculating the Covariance Matrix $\Sigma$

### Objective

To compute the shared covariance matrix $\Sigma$, which defines the shape of the Gaussian distribution for each class. This matrix captures the variance and covariance of the features across all samples, assuming the same covariance for both classes.

### Mathematical Formula

The covariance matrix $\Sigma$ is given by:

$$\Sigma = \frac{1}{n} \sum_{i=1}^{n} \left(x^{(i)} - \mu_{y^{(i)}}\right) \left(x^{(i)} - \mu_{y^{(i)}}\right)^T$$

Where:

- $n$ is the total number of samples.
- $x^{(i)}$ is the feature vector of the $i$-th sample.
- $\mu_{y^{(i)}}$ is the mean vector corresponding to the class label $y^{(i)}$:
    - If $y^{(i)} = 0$, use $\mu_0$.
    - If $y^{(i)} = 1$, use $\mu_1$.

### Conceptual Steps

1. **Compute the Deviations:**
    - For each sample $x^{(i)}$, calculate the deviation from its corresponding class mean:
        - $x^{(i)} - \mu_0$ for class 0 samples.
        - $x^{(i)} - \mu_1$ for class 1 samples.
2. **Sum the Outer Products:**
    - For each sample, compute the outer product of the deviation vector with itself and sum these matrices over all samples.
3. **Normalize by the Total Number of Samples:**

- Divide the summed matrix by $n$ (the total number of samples) to compute the final

## Implementation of the `predict` Method

The goal of the `predict` method is to use the parameters learned during the `fit` method to predict whether new input data $x$ belongs to class 1 ("Iris-virginica") or class 0 (not "Iris-virginica").

### Steps to Implement `predict`

1. **Compute the Decision Boundary:**
   - Use the parameters ($self.phi$, $self.mu0$, $self.mu1$, $self.covariance$) learned during the `fit` method to compute the decision boundary for each sample.
   - Calculate the log-likelihood ratio or the linear decision boundary for each sample to determine the class probabilities.
2. **Calculate the Log-Likelihood Ratio:**
   - For each sample, compute the log-likelihood ratio of it belonging to class 1 ("Iris-virginica") versus class 0 (not "Iris-virginica").
   - This involves using the inverse of the covariance matrix ($self.covariance$) and the mean vectors ($self.mu0$, $self.mu1$).
3. **Convert to Probabilities:**
   - Apply the logistic (sigmoid) function to the log-likelihood ratio to convert it to a probability:

$$p = \frac{1}{1 + \exp(-\text{log-likelihood})}$$

   - This gives the probability of each sample belonging to class 1.
4. **Make Predictions:**
   - For each sample, if the probability $p \geq 0.5$, classify it as "Iris-virginica" (class 1). Otherwise, classify it as not "Iris-virginica" (class 0).

### Mathematical Details for Log-Likelihood Ratio

The log-likelihood ratio $\log(p(x \mid y = 1)) - \log(p(x \mid y = 0))$ can be computed as:

$$\text{log-likelihood} = x \cdot \theta$$

where:

- $x$ is the input data (make sure to add an intercept term if needed).
- $\theta = \Sigma^{-1}(\mu_1 - \mu_0)$ is the parameter vector computed from the inverse of the covariance matrix and the difference between the mean vectors.
- $\theta_0 = -\frac{1}{2}(\mu_1^T \Sigma^{-1} \mu_1 - \mu_0^T \Sigma^{-1} \mu_0) + \log\left(\frac{\phi}{1-\phi}\right)$ is the intercept term.

### Implementation Strategy

1. **Compute the Parameter Vector ($\theta$):**
    - Use the inverse of $self.covariance$ and the difference between $self.mu1$ and $self.mu0$.
2. **Compute the Intercept ($\theta_0$):**
    - Calculate using the mean vectors, the covariance matrix, and the class prior $self.phi$.
3. **Calculate the Log-Likelihood Ratio:**
    - For each input sample, compute the linear decision boundary.
4. **Apply the Sigmoid Function:**
    - Convert the decision boundary to a probability.
5. **Classify Based on Probability:**
    - Use a threshold of 0.5 to make the final classification.

## Implementation of the `predict` Method

The goal of the `predict` method is to use the parameters learned during the `fit` method to predict whether new input data $x$ belongs to class 1 ("Iris-virginica") or class 0 (not "Iris-virginica").

### Steps to Implement `predict`

1. **Compute the Decision Boundary:**
    - Use the parameters ($self.phi$, $self.mu0$, $self.mu1$, $self.covariance$) learned during the `fit` method to compute the decision boundary for each sample.
    - Calculate the log-likelihood ratio or the linear decision boundary for each sample to determine the class probabilities.
2. **Calculate the Log-Likelihood Ratio:**
    - For each sample, compute the log-likelihood ratio of it belonging to class 1 ("Iris-virginica") versus class 0 (not "Iris-virginica").
    - This involves using the inverse of the covariance matrix ($self.covariance$) and the mean vectors ($self.mu0$, $self.mu1$).
3. **Convert to Probabilities:**
    - Apply the logistic (sigmoid) function to the log-likelihood ratio to convert it to a probability:

$$p = \frac{1}{1 + \exp(-\text{log-likelihood})}$$

    - This gives the probability of each sample belonging to class 1.
4. **Make Predictions:**
    - For each sample, if the probability $p \geq 0.5$, classify it as "Iris-virginica" (class 1). Otherwise, classify it as not "Iris-virginica" (class 0).

**Mathematical Details for Log-Likelihood Ratio**

The log-likelihood ratio $\log(p(x \mid y=1)) - \log(p(x \mid y=0))$ can be computed as:

$$\text{log-likelihood} = x \cdot \theta + \theta_0$$

where:

- $x$ is the input data (make sure to add an intercept term if needed).
- $\theta = \Sigma^{-1}(\mu_1 - \mu_0)$ is the parameter vector computed from the inverse of the covariance matrix and the difference between the mean vectors.
- $\theta_0 = -\frac{1}{2}(\mu_1^T \Sigma^{-1} \mu_1 - \mu_0^T \Sigma^{-1} \mu_0) + \log\left(\frac{\phi}{1-\phi}\right)$ is the intercept term.

**Implementation Strategy**

1. **Compute the Parameter Vector ($\theta$):**
    - Use the inverse of $self.covariance$ and the difference between $self.mu1$ and $self.mu0$.
2. **Compute the Intercept ($\theta_0$):**
    - Calculate using the mean vectors, the covariance matrix, and the class prior $self.phi$.
3. **Calculate the Log-Likelihood Ratio:**
    - For each input sample, compute the linear decision boundary.
4. **Apply the Sigmoid Function:**
    - Convert the decision boundary to a probability.
5. **Classify Based on Probability:**
    - Use a threshold of 0.5 to make the final classification. """

# Print out accuracy of the algorithm on the Iris Dataset

In [27]:
```
def main(data_path):
    """Problem: Gaussian discriminant analysis (GDA)"""
    # Load dataset
    x, y = load_dataset(data_path, add_intercept=True)

    # Split dataset into training and validation
    split_index = int(0.8 * x.shape[0])
    x_train, x_valid = x[:split_index], x[split_index:]
    y_train, y_valid = y[:split_index], y[split_index:]

    # Train GDA classifier
    clf = GDA()
    clf.fit(x_train, y_train)

    # Predict probabilities
    probabilities = clf.predict(x_valid)
    predictions = np.where(probabilities >= 0.5, 'Iris-virginica', 'Iris-setos

    # Calculate accuracy
    accuracy = np.mean(predictions == y_valid)
    print(f'Accuracy: {accuracy:.4f}')

    # Print predicted probabilities
    print("Predicted Probabilities:")
    for prob in probabilities:
        print(f"{prob:.4f}")

if __name__ == '__main__':
    main(data_path='C:/Users/Machine_Learning/Downloads/archive/iris.csv')
```

```
Accuracy: 0.8000
Predicted Probabilities:
1.0000
1.0000
0.0000
1.0000
1.0000
0.0000
1.0000
1.0000
1.0000
0.0000
0.0000
1.0000
1.0000
0.0000
0.0000
1.0000
1.0000
1.0000
1.0000
1.0000
1.0000
1.0000
1.0000
1.0000
1.0000
1.0000
1.0000
1.0000
1.0000
```

Reasons for Differences in Accuracy

Covariance Matrix Calculation:

Manual computation might introduce inaccuracies due to floating-point arithmetic errors, especially with larger datasets. Using np.cov is more reliable and efficient.