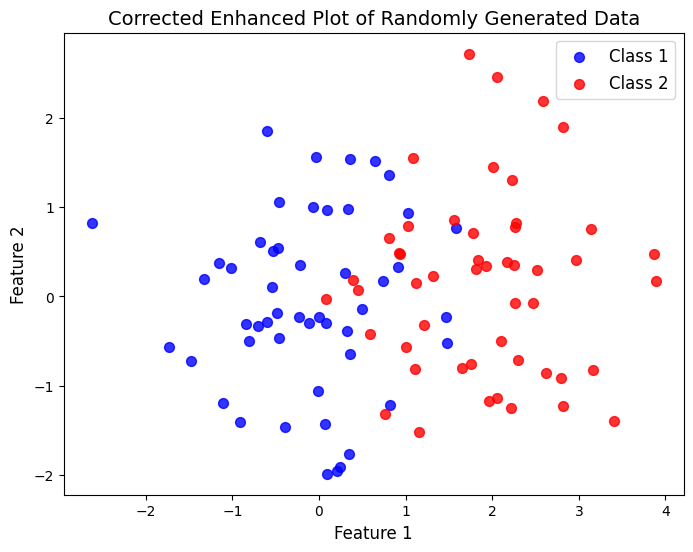
**Method 1: Using sklearn and numpy packages**

**Section 1: Data Generation and Plotting**



Explanation:

* This section generates synthetic data for two classes with different means and the same covariance matrix.
* The data is then plotted using Matplotlib, where each class is represented by a scatter plot with different colors.



**Section 2: Data Preprocessing**



Explanation:

* The data from the two classes is concatenated, and corresponding labels are created.
* The data is shuffled to ensure a random order, which is important for later training and evaluation.

**Section 3: Training and Evaluation**



Explanation:

* The code defines a function **train\_and\_evaluate** to train a Perceptron model and evaluate its performance.
* The function is called for different training set sizes (**ntr\_values**) and corresponding testing set sizes (**nts\_values**).
* The best-performing model is printed along with its weights, biases, and accuracy.
* Decision boundaries are plotted to visualize the model's classification.

**Report Summary:**

1. Data Generation and Plotting:

* Random data for two classes generated with different means.
* A corrected plot is created using Matplotlib.

1. Data Preprocessing:

* Data from both classes concatenated.
* Shuffling done for randomization.

1. Training and Evaluation:

* Perceptron model trained and evaluated.
* Performance metrics (accuracy, weights, biases) printed.
* Decision boundaries plotted for visualization.

**Method 2: Using only numpy packages**

**Section 1: Data Generation**



Explanation:

This section uses NumPy to generate random numbers from normal distributions with specified means and variances for two classes. The generated data is then labeled, creating two sets of 50 samples each. The final dataset, **'combined\_dataset'**, is a concatenation of these two classes.

**Section 2: Data Splitting Function**



Explanation:

The **split\_data\_for\_training\_and\_testing** function shuffles the dataset to ensure randomness. It then splits the data for each class into training and testing sets based on the specified number of samples per class for training. The function returns a tuple containing the training and testing data.

**Section 3: Data Splitting**



Explanation:

Here, the dataset is split into training and testing sets for three different sizes (10, 20, and 30 samples per class) using the previously defined function.

**Section 4: Single Layer Perceptron Class**



Explanation:

The **SingleLayerPerceptron** class is introduced, representing a basic single-layer perceptron. It has methods for initialization, activation, prediction, training, and accuracy calculation. This class will be used to create perceptron objects for different training set sizes.

**Section 5: Perceptron Training**



Explanation:

Perceptron objects are created for three different training set sizes (10, 20, and 30 samples per class). Each perceptron is initialized with specified parameters, and the train method is called to train the model on its respective training dataset.

**Section 6: Decision Boundary Plotting**



Explanation:

A function, **plot\_decision\_boundary**, is defined to visualize the decision boundary of a perceptron. The decision boundary is plotted using Matplotlib, showing how the perceptron classifies different regions. This function is then applied to each perceptron with their corresponding training data, providing insights into the learned decision boundaries.

