In [8]:

```python
#program2
#1BM22AI037
#DEVELOP AND IMPLEMENT A PROGRAM TO EXECUTE THE PERCEPTRON LEARNING ALGORITHM CUSTOMIZE
#TO TRAIN A SINGLE LAYER PERCEPTRON FOR BINARY CLASSIFICSTION TEST
#CREATE A ROBUST ALGORITHM THAT REFINES THE MODELS WEIGHT ITERATIVELY RESULTING
#IN A PROFFICIENT SINGLE LAYERED PERCEPTRON CAPABLE OF EFFICIENTLY HANDLING BINARY CLASSIFICATION CHALLENGES

import numpy as np
class SingleLayerPerceptron:
    def __init__(self, input_dim, learning_rate=0.01, epochs=1000):
        self.weights = np.random.rand(input_dim + 1) * 0.01
        self.learning_rate = learning_rate
        self.epochs = epochs
    def relu(self, x):
        return np.maximum(0, x)
    def sigmoid(self, x):
        return 1 / (1 + np.exp(-x))
    def predict(self, x):
        x_with_bias = np.insert(x, 0, 1)
        linear_output = np.dot(self.weights, x_with_bias)
        activated_output = self.sigmoid(linear_output)
        return activated_output
    def fit(self, X, y):
        for _ in range(self.epochs):
            for i in range(X.shape[0]):
                x_i = X[i]
                y_i = y[i]
                x_with_bias = np.insert(x_i, 0, 1)
                linear_output = np.dot(self.weights, x_with_bias)
                activated_output = self.sigmoid(linear_output)
                error = y_i - activated_output
                gradient = activated_output * (1 - activated_output)
                adjustment = self.learning_rate * error * gradient * x_with_bias
                self.weights += adjustment
X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
y = np.array([0, 0, 0, 1])
perceptron = SingleLayerPerceptron(input_dim=2)
perceptron.fit(X, y)
```

```
for x in X:
    print(perceptron.predict(x))
```

```
0.2419764207090503
0.343713922644321
0.3439504645987434
0.4624081867137969
```

In [7]:

```
import numpy as np
class Perceptron:
    def __init__(self, learning_rate=0.01, n_iter=1000):
        self.learning_rate = learning_rate
        self.n_iter = n_iter
        self.weights = None
        self.bias = None
    def fit(self, X, y):
        n_samples, n_features = X.shape
        self.weights = np.zeros(n_features)
        self.bias = 0
        for _ in range(self.n_iter):
            for idx, x_i in enumerate(X):
                linear_output = np.dot(x_i, self.weights) + self.bias
                y_predicted = self._activation_function(linear_output)
                update = self.learning_rate * (y[idx] - y_predicted)
                self.weights += update * x_i
                self.bias += update
    def _activation_function(self, x):
        return np.where(x >= 0, 1, 0)
    def predict(self, X):
        linear_output = np.dot(X, self.weights) + self.bias
        y_predicted = self._activation_function(linear_output)
        return y_predicted
if __name__ == "__main__":
    from sklearn.datasets import make_blobs
    from sklearn.model_selection import train_test_split
    from sklearn.metrics import accuracy_score
    X, y = make_blobs(n_samples=100, centers=2, random_state=42, cluster_std=1.05)
    y = np.where(y == 0, 0, 1)
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
    perceptron = Perceptron(learning_rate=0.1, n_iter=1000)
```

```
    perceptron.fit(X_train, y_train)
    predictions = perceptron.predict(X_test)
    accuracy = accuracy_score(y_test, predictions)
    print(f"Accuracy: {accuracy:.2f}")
```

```
Accuracy: 1.00
```

In [ ]: