Exercise 11b Date: 26/03/2025

Single Layer Perceptron - Non-Linearly Separable Dataset

Aim: To build a Single Layer Perceptron on a non-linearly separable dataset.

Algorithm:

The Single-Layer Perceptron (SLP) is a fundamental neural network model used for binary classification.
It consists of a single neuron with an activation function that determines the output.
The perceptron updates its weights iteratively based on misclassified points until convergence, making it suitable for linearly separable data.

The learning process is formulated as:

$$w \leftarrow w + \eta(y_i - \hat{y}_i)x_i$$

Where:

- w represents the weight vector,
- x_i is the input feature vector,
- y_i is the actual class label (0 or 1),
- $\hat{y}_i$ is the predicted output,
- η is the learning rate.

The perceptron uses a step activation function, which outputs 1 if the weighted sum is non-negative and 0 otherwise. This makes it suitable for linearly separable data.

Step 1: Import Libraries

- Import necessary Python libraries including NumPy for numerical operations, Matplotlib for visualization, and scikit-learn modules for data splitting and preprocessing.

Step 2: Load the Dataset

- Generate a synthetic dataset containing 200 random feature values with 2 dimensions.
- Assign binary class labels based on whether the sum of squared feature values exceeds a threshold.
- Augment the dataset by adding squared feature terms to introduce non-linearity.

Step 3: Define the Single Layer Perceptron

- Create a class `SingleLayerPerceptron` that initializes random weights.
- Implement a step and sigmoid activation function, where step outputs 1 if the weighted sum is non-negative, otherwise 0, and sigmoid outputs probabilities.
- Implement a `train()` method that updates weights based on prediction errors using a simple perceptron learning rule.

Step 4: Train the Model

- Initialize the perceptron with 4 input features (original + squared terms).
- Train the perceptron on the dataset using the training samples.

Step 5: Evaluate the Model

- Use the trained model to make predictions on the dataset.
- Compute the accuracy by comparing predicted labels with actual labels.
- Print the model's accuracy on the dataset.

Step 6: Visualize the Results

- Define a function to scatter plot the data points with class labels.
- Overlay the decision boundary using model predictions on a grid.
- Display the scatter plot to observe how well the perceptron classifies the data.

Import the libraries

In [15]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

Load the dataset

In [5]:
```
X_nonlinear = np.random.randn(200, 2)
y_nonlinear = (X_nonlinear[:, 0]**2 + X_nonlinear[:, 1]**2 > 1).astype(int)
X_nonlinear = np.c_[X_nonlinear, X_nonlinear[:, 0]**2, X_nonlinear[:, 1]**2]
```

Single Layer Perceptron

In [6]:
```
class SingleLayerPerceptron:
    def __init__(self, input_dim, learning_rate=0.01, epochs=1000, activation='step'):
        self.weights = np.random.randn(input_dim + 1)
        self.learning_rate = learning_rate
        self.epochs = epochs
        self.activation_function = activation

    def activation(self, x):
        if self.activation_function == 'step':
            return 1 if x >= 0 else 0
        elif self.activation_function == 'sigmoid':
            return 1 / (1 + np.exp(-x)) > 0.5

    def predict(self, X):
        X_bias = np.c_[np.ones((X.shape[0], 1)), X]
        return np.array([self.activation(np.dot(self.weights, x)) for x in X_bias])

    def train(self, X, y):
        X_bias = np.c_[np.ones((X.shape[0], 1)), X]
        for _ in range(self.epochs):
            for i in range(X.shape[0]):
                prediction = self.activation(np.dot(self.weights, X_bias[i]))
                error = y[i] - prediction
                self.weights += self.learning_rate * error * X_bias[i]
```

Training the model

In [7]:
```
perceptron_nonlinear = SingleLayerPerceptron(input_dim=4, activation='sigmoid')
perceptron_nonlinear.train(X_nonlinear, y_nonlinear)
```

Evaluating the model

In [8]:
```
y_pred_nonlin = perceptron_nonlinear.predict(X_nonlinear)
acc_nonlin = np.mean(y_pred_nonlin == y_nonlinear)
print(f"Accuracy on Non-Linearly Separable Data: {acc_nonlin:.4f}")
```

```
Accuracy on Non-Linearly Separable Data: 1.0000
```

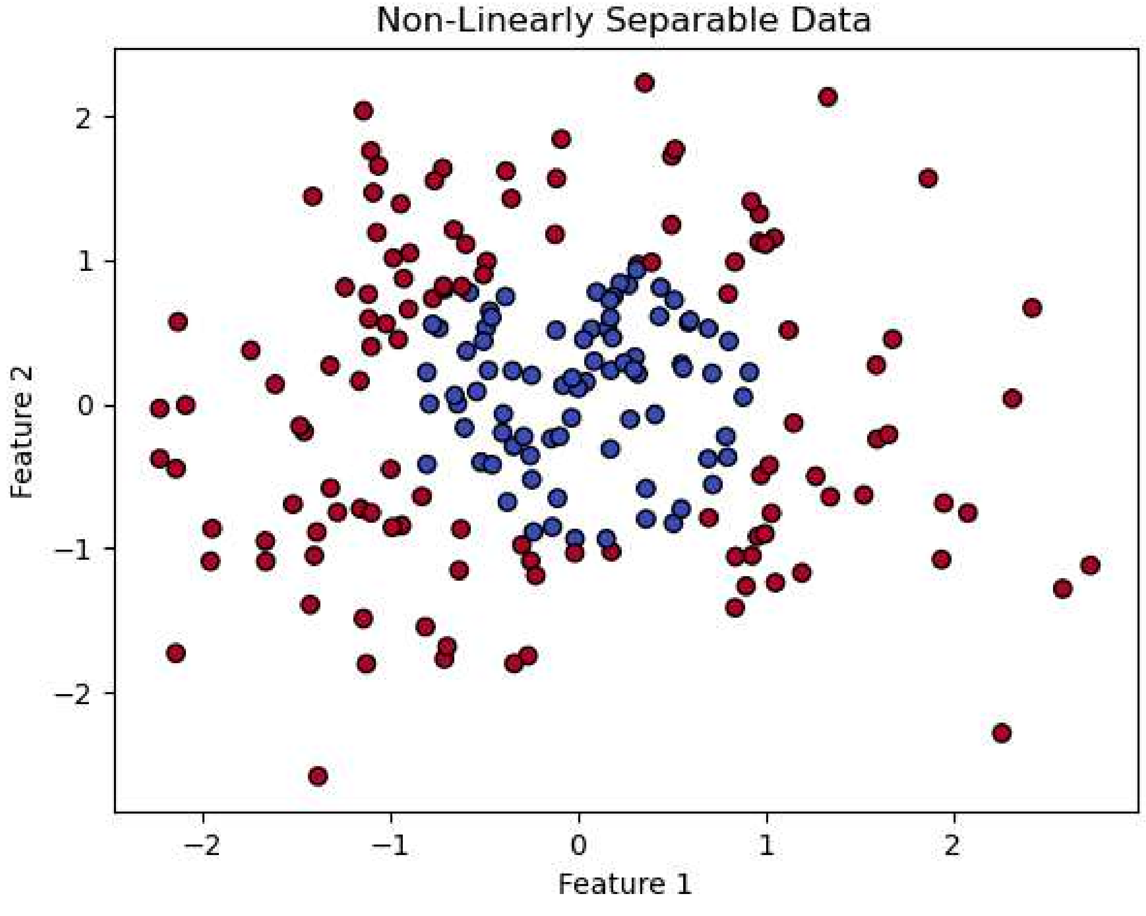
Visualisation

In [9]:
```
def plot_data_og(X, y, title, model=None):
    plt.scatter(X[:, 0], X[:, 1], c=y, cmap='coolwarm', edgecolors='k')
    plt.title(title)
    plt.xlabel("Feature 1")
    plt.ylabel("Feature 2")

    if model is not None:
        xx, yy = np.meshgrid(np.linspace(X[:, 0].min(), X[:, 0].max(), 100),
                             np.linspace(X[:, 1].min(), X[:, 1].max(), 100))
        Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
        Z = Z.reshape(xx.shape)
        plt.contour(xx, yy, Z, levels=[0.5], colors='black')

    plt.show()
```

In [11]:
```
plot_data_og(X_nonlinear, y_nonlinear, "Non-Linearly Separable Data")
```

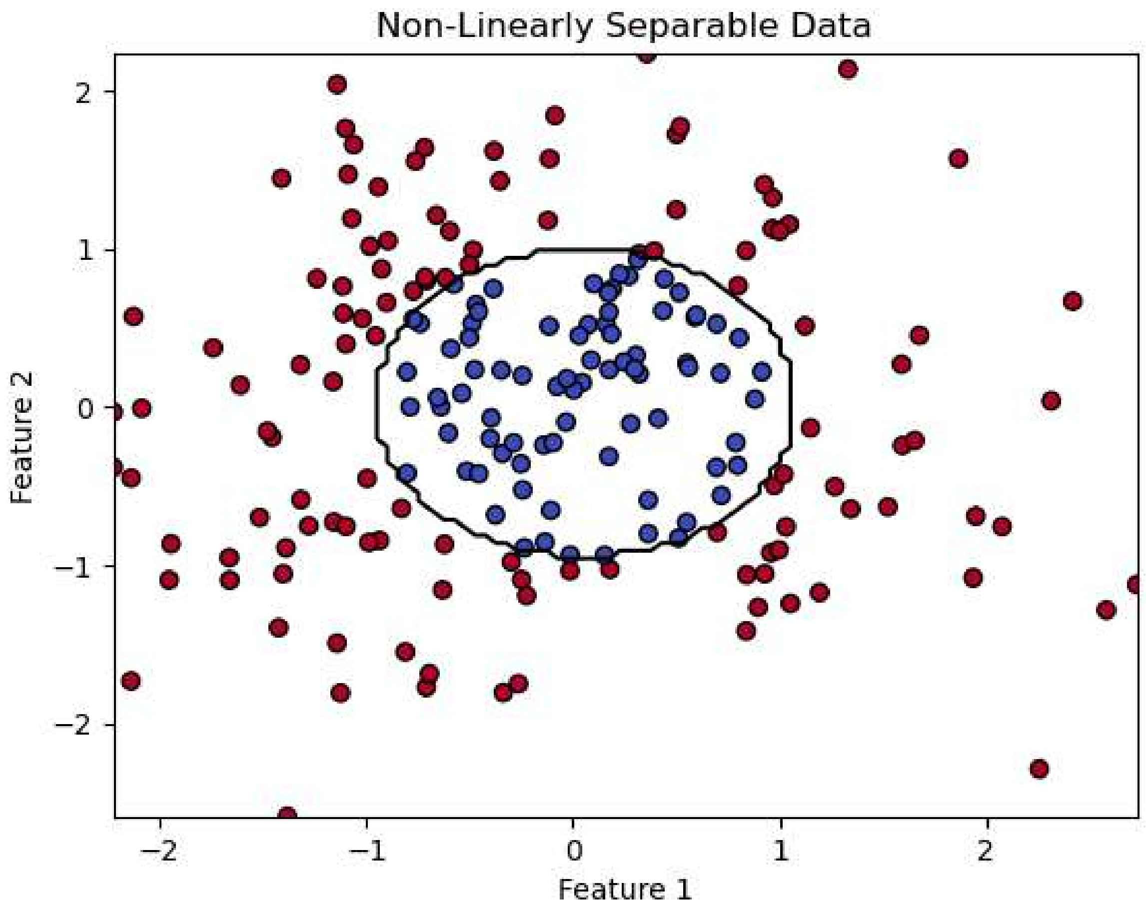


In [12]:
```
def plot_data(X, y, title, model):
    plt.scatter(X[:, 0], X[:, 1], c=y, cmap='coolwarm', edgecolors='k')
    plt.title(title)
    plt.xlabel("Feature 1")
    plt.ylabel("Feature 2")

    xx, yy = np.meshgrid(np.linspace(X[:, 0].min(), X[:, 0].max(), 100),
                         np.linspace(X[:, 1].min(), X[:, 1].max(), 100))
    Z = model.predict(np.c_[xx.ravel(), yy.ravel(), xx.ravel()**2, yy.ravel()**2])
    Z = Z.reshape(xx.shape)
    plt.contour(xx, yy, Z, levels=[0.5], colors='black')

    plt.show()
```

In [13]:
```
plot_data(X_nonlinear, y_nonlinear, "Non-Linearly Separable Data", perceptron_nonlinear)
```



Result

A Single-Layer Perceptron (SLP) model was successfully implemented for classifying a non-linearly separable dataset.
