In [21]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from matplotlib.animation import FuncAnimation
from sklearn.datasets import make_classification
from sklearn.datasets import load_digits
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split, cross_val_score, KFold
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import Perceptron
```

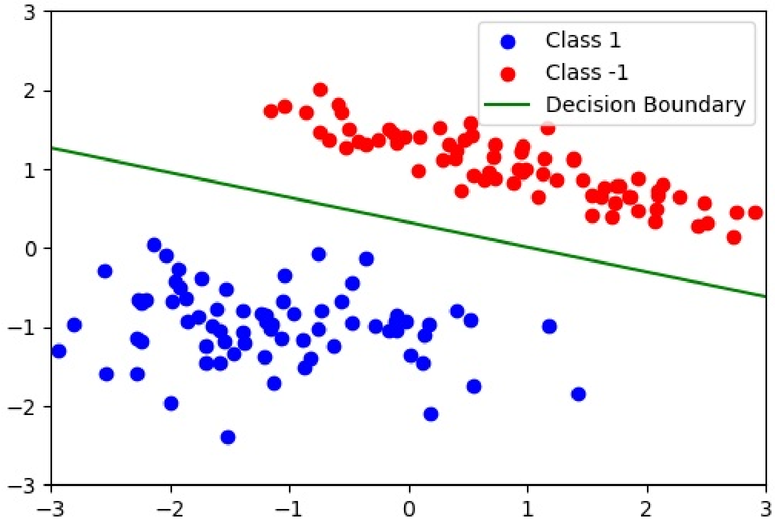
1a) Perceptron Algorithm (linearly separable)

In [53]:
```python
X = np.load('Xlin_sep.npy')
y = np.load('ylin_sep.npy')
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=50, random_state=42)
```

In [55]:
```python
#y = np.where(y == 0, -1, 1)
class Perceptron1:
    def __init__(self, input_dim, learning_rate=0.01, max_epochs=100):
        self.weights = np.zeros(input_dim + 1)
        self.learning_rate = learning_rate
        self.max_epochs = max_epochs
        self.history = []

    def predict(self, X):
        X_with_bias = np.c_[X, np.ones(X.shape[0])]
        return np.sign(X_with_bias @ self.weights)

    def fit(self, X, y):
        X_with_bias = np.c_[X, np.ones(X.shape[0])]
        for epoch in range(self.max_epochs):
            for i in range(X_with_bias.shape[0]):
                if y[i] * (X_with_bias[i] @ self.weights) <= 0:
                    self.weights += self.learning_rate * y[i] * X_with_bias[i]
            self.history.append(self.weights.copy())
            if np.all(y == self.predict(X)):
                break

input_dim = X_train.shape[1]
perceptron1 = Perceptron1(input_dim=input_dim)
perceptron1.fit(X_train, y_train)

train_accuracy = np.mean(perceptron1.predict(X_train) == y_train)
test_accuracy = np.mean(perceptron1.predict(X_test) == y_test)
print(f"Train Accuracy: {train_accuracy * 100:.2f}%")
print(f"Test Accuracy: {test_accuracy * 100:.2f}%")
```

```
Train Accuracy: 100.00%
Test Accuracy: 100.00%
```

In [56]:
```python
def plot_decision_boundary(weights, X, y, ax):
    ax.clear()
    ax.scatter(X[y == 1][:, 0], X[y == 1][:, 1], color='blue', label='Class 1')
    ax.scatter(X[y == -1][:, 0], X[y == -1][:, 1], color='red', label='Class -1')

    x_min, x_max = ax.get_xlim()
    x_vals = np.linspace(x_min, x_max, 100)
    y_vals = -(weights[0] * x_vals + weights[2]) / weights[1]
    ax.plot(x_vals, y_vals, color='green', label='Decision Boundary')

    ax.legend()
    ax.set_xlim(-3, 3)
    ax.set_ylim(-3, 3)

fig, ax = plt.subplots(figsize=(6, 4))
plot_decision_boundary(perceptron1.history[-1], X_train, y_train, ax)
plt.show()
```



In [15]:
```
# 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
```

1b) Perceptron Algorithm (Non-linearly separable)

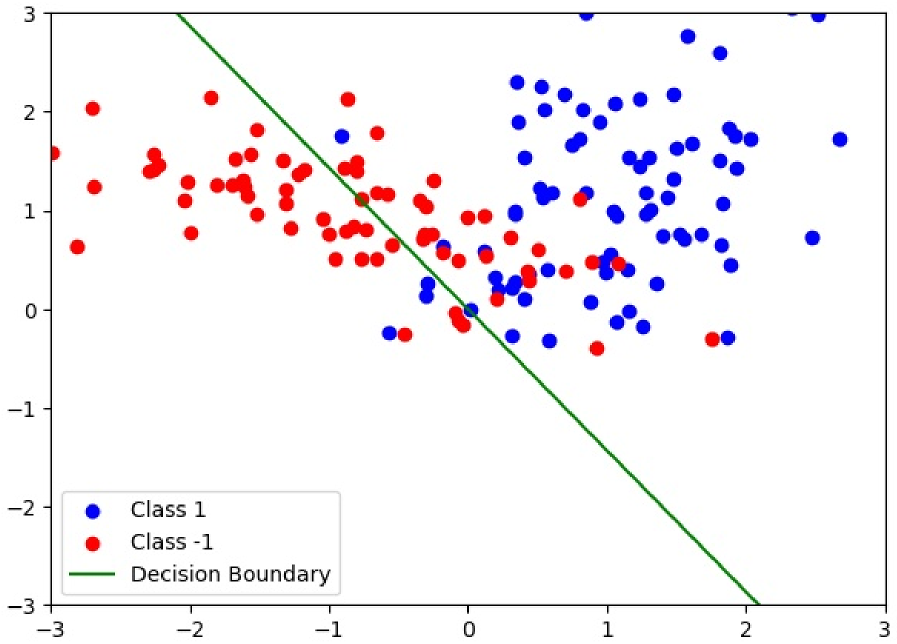
In [16]:
```
X1 = np.load('Xlinnoise_sep.npy')
y1 = np.load('ylinnoise_sep.npy')
X_train, X_test, y_train, y_test = train_test_split(X1, y1, test_size=50, random_state=42)
```

In [17]:
```
class Perceptron2:
    def __init__(self, input_dim, learning_rate=0.01, max_epochs=100):
        self.weights = np.zeros(input_dim + 1)
        self.learning_rate = learning_rate
        self.max_epochs = max_epochs
        self.history = []

    def predict(self, X):
        X_with_bias = np.c_[X, np.ones(X.shape[0])]
        return np.sign(X_with_bias @ self.weights)

    def fit(self, X, y):
        X_with_bias = np.c_[X, np.ones(X.shape[0])]
        for epoch in range(self.max_epochs):
            for i in range(X_with_bias.shape[0]):
                if y[i] * (X_with_bias[i] @ self.weights) <= 0:
                    self.weights += self.learning_rate * y[i] * X_with_bias[i]
            self.history.append(self.weights.copy())

    def score(self, X, y):
        y_pred = self.predict(X)
        return (np.mean(y_pred == y))*100


perceptron2 = Perceptron2(input_dim=2, learning_rate=0.01, max_epochs=100)
perceptron2.fit(X_train, y_train)

train_acc = perceptron2.score(X_train, y_train)
test_acc = perceptron2.score(X_test, y_test)
print(f"Train Accuracy: {train_acc:.2f}%")
print(f"Test Accuracy: {test_acc:.2f}%")
```

```
Train Accuracy: 78.00%
Test Accuracy: 68.00%
```

In [18]:
```
def plot_decision_boundary(weights, X, y, ax):
    ax.clear()
    ax.scatter(X[y == 1][:, 0], X[y == 1][:, 1], color='blue', label='Class 1')
    ax.scatter(X[y == -1][:, 0], X[y == -1][:, 1], color='red', label='Class -1')
```

```python
    # Decision boundary
    x_min, x_max = ax.get_xlim()
    x_vals = np.linspace(x_min, x_max, 100)
    y_vals = -(weights[0] * x_vals + weights[2]) / weights[1]
    ax.plot(x_vals, y_vals, color='green', label='Decision Boundary')

    ax.legend()
    ax.set_xlim(-3, 3)
    ax.set_ylim(-3, 3)

fig, ax = plt.subplots(figsize=(7, 5))
plot_decision_boundary(perceptron2.history[-1], X_train, y_train, ax)
plt.show()
```



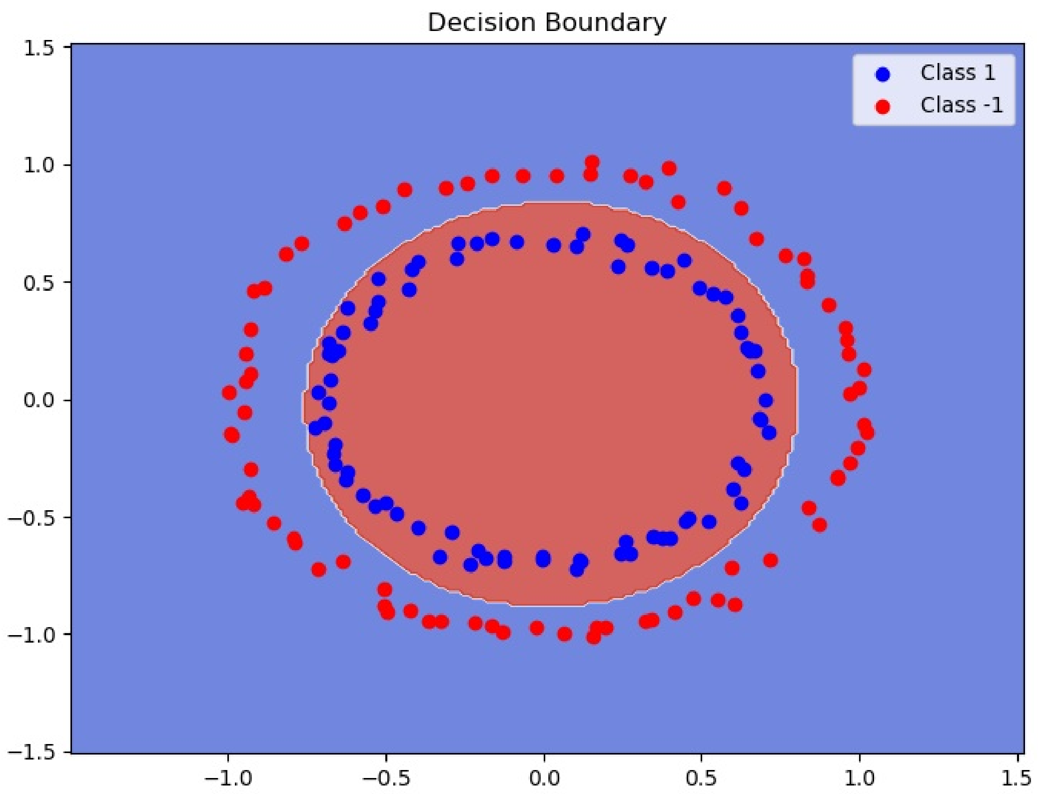
1c) Polynomial Feature Expansion

In [58]:
```python
X = np.load('circles_x.npy')
y = np.load('circles_y.npy')

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)
poly = PolynomialFeatures(degree=2, include_bias=False) # polynomial feature expansion
X_train_poly = poly.fit_transform(X_train)
X_test_poly = poly.transform(X_test)
```

In [62]:
```python
class Perceptron3:  #Using the Peceptron class (non-linear) from the previous question
    def __init__(self, input_dim, learning_rate=0.01, max_epochs=100):
        self.weights = np.zeros(input_dim + 1)
        self.learning_rate = learning_rate
        self.max_epochs = max_epochs
        self.history = []

    def predict(self, X):
        X_with_bias = np.c_[X, np.ones(X.shape[0])]
        return np.sign(X_with_bias @ self.weights)

    def fit(self, X, y):
        X_with_bias = np.c_[X, np.ones(X.shape[0])]
        for epoch in range(self.max_epochs):
            for i in range(X_with_bias.shape[0]):
                if y[i] * (X_with_bias[i] @ self.weights) <= 0:
                    self.weights += self.learning_rate * y[i] * X_with_bias[i]
            self.history.append(self.weights.copy())

    def score(self, X, y):
        y_pred = self.predict(X)
        return (np.mean(y_pred == y)) * 100

perceptron3 = Perceptron3(input_dim=X_train_poly.shape[1], learning_rate=0.01, max_epochs=100)
perceptron3.fit(X_train_poly, y_train)

train_accuracy = perceptron3.score(X_train_poly, y_train)
test_accuracy = perceptron3.score(X_test_poly, y_test)

print(f"Train Accuracy: {train_accuracy:.2f}%")
print(f"Test Accuracy: {test_accuracy:.2f}%")
```

```
Train Accuracy: 100.00%
Test Accuracy: 100.00%
```

In [60]:
```python
def plot_decision_boundary(X, y, model, poly, ax):

    x_min, x_max = X[:, 0].min() - 0.5, X[:, 0].max() + 0.5  # creating a grid of points
    y_min, y_max = X[:, 1].min() - 0.5, X[:, 1].max() + 0.5
    xx, yy = np.meshgrid(np.linspace(x_min, x_max, 200), np.linspace(y_min, y_max, 200))
    grid = np.c_[xx.ravel(), yy.ravel()]

    grid_poly = poly.transform(grid) # transform the grid using polynomial features
    zz = model.predict(grid_poly).reshape(xx.shape)

    ax.contourf(xx, yy, zz, alpha=0.8, cmap=plt.cm.coolwarm) # Plot decision boundary
    ax.scatter(X[y == 1][:, 0], X[y == 1][:, 1], color='blue', label='Class 1')
    ax.scatter(X[y == -1][:, 0], X[y == -1][:, 1], color='red', label='Class -1')
    ax.legend()
    ax.set_title("Decision Boundary")
    ax.set_xlim(x_min, x_max)
    ax.set_ylim(y_min, y_max)

fig, ax = plt.subplots(figsize=(8, 6))
plot_decision_boundary(X_train, y_train, perceptron3, poly,ax)
plt.show()

# def plot_decision_boundary(weights, X, y, ax):
#     ax.clear()
#     ax.scatter(X[y == 1][:, 0], X[y == 1][:, 1], color='blue', label='Class 1')
#     ax.scatter(X[y == -1][:, 0], X[y == -1][:, 1], color='red', label='Class -1')
#     # Decision boundary
#     x_min, x_max = ax.get_xlim()
#     x_vals = np.linspace(x_min, x_max, 100)
#     y_vals = -(weights[0] * x_vals + weights[2]) / weights[1]
#     ax.plot(x_vals, y_vals, color='green', label='Decision Boundary')

#     ax.legend()
#     ax.set_xlim(-3, 3)
#     ax.set_ylim(-3, 3)

# fig, ax = plt.subplots(figsize=(8, 6))
# plot_decision_boundary(perceptron2.history[-1], X_train, y_train, ax)
# plt.show()
```



In [ ]:
```python

```

In [46]:
```python
# from sklearn.linear_model import Perceptron

# perceptron = Perceptron(max_iter=1000, tol=1e-3, random_state=42)
# perceptron.fit(X_train_poly, y_train)

# # Evaluate the perceptron
```

```python
# y_train_pred = perceptron.predict(X_train_poly)
# y_test_pred = perceptron.predict(X_test_poly)

# train_accuracy = accuracy_score(y_train, y_train_pred)
# test_accuracy = accuracy_score(y_test, y_test_pred)

# print(f"Train Accuracy: {train_accuracy:.2f}")
# print(f"Test Accuracy: {test_accuracy:.2f}")
```

```
Train Accuracy: 1.00
Test Accuracy: 1.00
```

In [ ]:

2) Classification via Neural Networks

The task for this exercise is to develop a Neural Network model that can classify human-written digits (0 through 9). We will reuse concepts from previous exercises such as hyperparameter optimization and k-fold cross-validatio P

In [ ]:

In [19]:

```python
digits = load_digits()
X, y = digits.data, digits.target
#X_train_full, X_test, y_train_full, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=)
X_train_full, X_test, y_train_full, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
k = 5
kf = KFold(n_splits=k, shuffle=True, random_state=42)
param_grid = {
    'hidden_layer_sizes': [(50,), (100,), (50, 50), (100, 50), (100, 100)],
    'activation': ['tanh', 'relu'],
    'solver': ['sgd', 'adam'],
    'learning_rate': ['constant', 'adaptive'],
    'alpha': [1e-5, 1e-4, 1e-3, 1e-2],
    'max_iter': [200, 300, 500]
}
```

In [20]:

```python
mlp = MLPClassifier(random_state=42)

random_search = RandomizedSearchCV(
    estimator=mlp,
    param_distributions=param_grid,
    n_iter=10,  # no. of random combinations to try
    cv=kf,
    scoring='accuracy',
    random_state=42,
    n_jobs=-1
)

random_search.fit(X_train_full, y_train_full)

best_model = random_search.best_estimator_  # evaluating the best model on the hold-out test set
y_test_pred = best_model.predict(X_test)
test_accuracy = accuracy_score(y_test, y_test_pred)

print("Best Hyperparameters:", random_search.best_params_)
print(f"Test Set Accuracy: {test_accuracy:.2f}")
```

```
Best Hyperparameters: {'solver': 'adam', 'max_iter': 200, 'learning_rate': 'constant', 'hidden_layer_sizes': (10
0,), 'alpha': 0.0001, 'activation': 'tanh'}
Test Set Accuracy: 0.98
```

In [ ]:

In [ ]: