In [1]:
```
# import libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import fetch_openml
from sklearn.svm import LinearSVC
from sklearn import svm
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.svm import SVC
```

Multiclass Perceptron

In [2]:
```
data0 = np.loadtxt('data0.txt')
x = data0[ : , :2]
y = data0[:, 2]
```

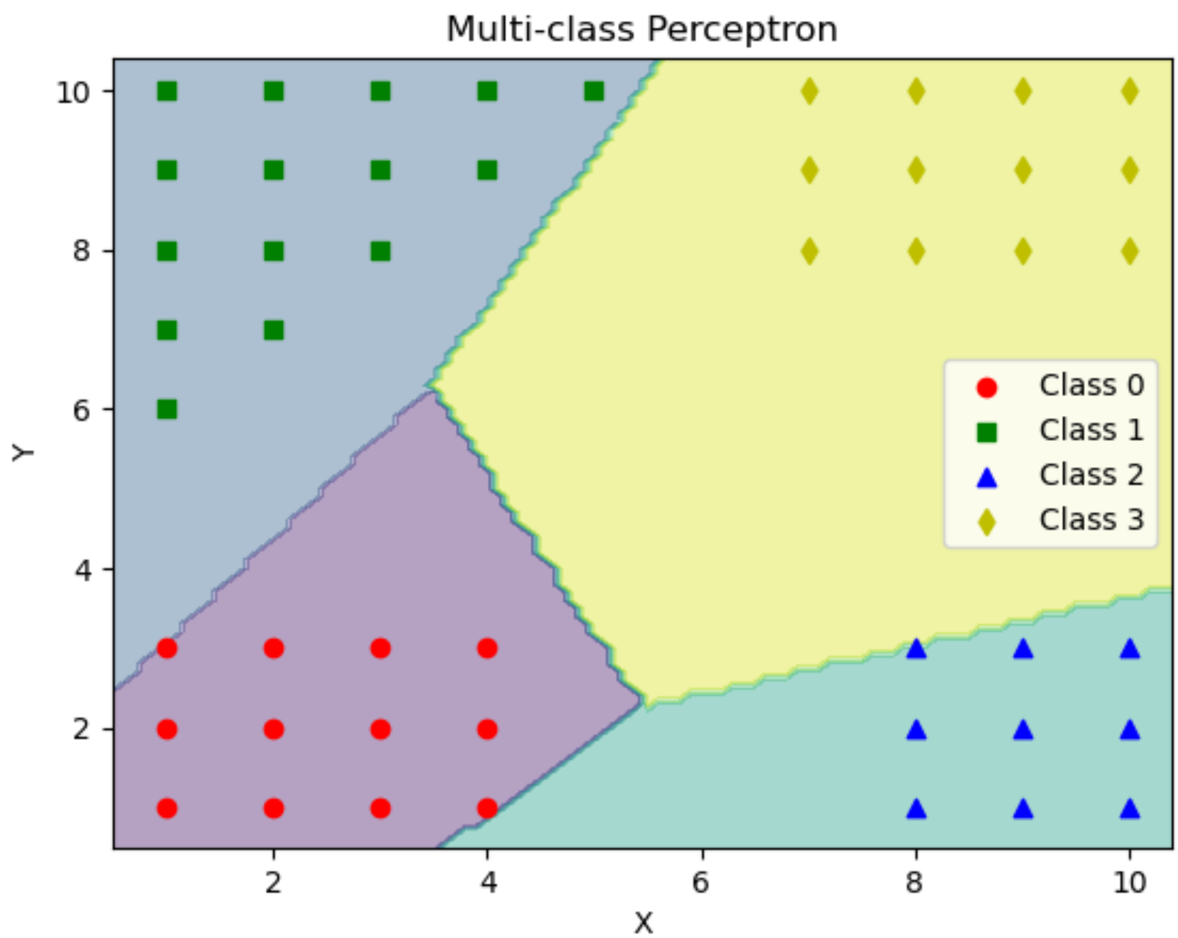
In [3]:
```
print(data0[0])
```

```
[1. 1. 0.]
```

The data0 is composed of 2-D coordinates (1st & 2nd column) and the labels (3rd column).

In [4]:
```
k = 4 ; #number of classes (0,1,2,3)
d = 2; #2-d (x1,x2)
w = np.zeros((k,d))
b = np.zeros(k)
epochs = 100
learning_rate = 0.1

for epoch in range(epochs):
    for i in range (x.shape[0]): #multiclass perceptron
        x_i = x[i]
        y_i = int(y[i])

        scores = np.dot(w, x_i) + b #scores
        y_pred = np.argmax(scores)

        if y_pred != y_i:
            w[y_i] += learning_rate * x_i
            b[y_i] += learning_rate
            w[y_pred] -= learning_rate *x_i
            b[y_pred] -= learning_rate
```

In [5]:
```
# Create plot showing data points and decision regions
x_min, x_max = x[:, 0].min() - 0.5, x[:, 0].max() + 0.5
y_min, y_max = x[:, 1].min() - 0.5, x[:, 1].max() + 0.5
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.1), np.arange(y_min, y_max, 1
Z = np.argmax(np.dot(np.c_[xx.ravel(), yy.ravel()], w.T) + b, axis=1)
Z = Z.reshape(xx.shape)
plt.contourf(xx, yy, Z, alpha=0.4)

markers = ['o', 's', '^', 'd']
colors = ['r', 'g', 'b', 'y']
for i in range(4):
    plt.scatter(x[y==i, 0], x[y==i, 1], marker=markers[i], color=colors[i],
plt.legend()
plt.xlabel('X')
plt.ylabel('Y')
plt.title('Multi-class Perceptron')
plt.show()
```



Multiclass SVM

a)

In [6]:
```
# Load MNIST data
mnist = fetch_openml('mnist_784', version=1)
x = mnist.data
y = mnist.target
```

In [ ]:

In [7]:
```
#seperate data in train & test data
x_train ,x_test, y_train, y_test =  train_test_split(x, y, test_size=10000,
```

b)

In [8]:
```
def fit_classifier(C_value=1.0):
    clf = svm.LinearSVC(C=C_value, loss='hinge', max_iter = 5000)
    clf.fit(x_train,y_train)

    ## Get predictions on training data
    train_preds = clf.predict(x_train)
    train_error = 1.0 - clf.score(x_train, y_train)

    ## Get predictions on test data
    test_preds = clf.predict(x_test)
    test_error = 1.0 - clf.score(x_test, y_test)

    return train_error, test_error
```

To find the optimal C we will compare different ones to find the best performance. The performance will be measured by lowering the training error as much as possible without significantly increasing the test error.

In [9]:
```
cvals = [0.01,0.1,1.0,10.0,100.0,1000.0,10000.0]
for c in cvals:
    train_error, test_error = fit_classifier(c)
    print ("Error rate for C = %0.2f: train %0.3f test %0.3f" % (c, train_er
```

```
/Users/marlenearredondo/opt/anaconda3/lib/python3.9/site-packages/sklearn/sv
m/_base.py:1206: ConvergenceWarning: Liblinear failed to converge, increase
the number of iterations.
  warnings.warn(
Error rate for C = 0.01: train 0.153 test 0.163
/Users/marlenearredondo/opt/anaconda3/lib/python3.9/site-packages/sklearn/sv
m/_base.py:1206: ConvergenceWarning: Liblinear failed to converge, increase
the number of iterations.
  warnings.warn(
Error rate for C = 0.10: train 0.121 test 0.131
/Users/marlenearredondo/opt/anaconda3/lib/python3.9/site-packages/sklearn/sv
m/_base.py:1206: ConvergenceWarning: Liblinear failed to converge, increase
the number of iterations.
  warnings.warn(
```

```
Error rate for C = 1.00: train 0.107 test 0.121
/Users/marlenearredondo/opt/anaconda3/lib/python3.9/site-packages/sklearn/sv
m/_base.py:1206: ConvergenceWarning: Liblinear failed to converge, increase
the number of iterations.
  warnings.warn(
Error rate for C = 10.00: train 0.166 test 0.176
/Users/marlenearredondo/opt/anaconda3/lib/python3.9/site-packages/sklearn/sv
m/_base.py:1206: ConvergenceWarning: Liblinear failed to converge, increase
the number of iterations.
  warnings.warn(
Error rate for C = 100.00: train 0.166 test 0.184
/Users/marlenearredondo/opt/anaconda3/lib/python3.9/site-packages/sklearn/sv
m/_base.py:1206: ConvergenceWarning: Liblinear failed to converge, increase
the number of iterations.
  warnings.warn(
Error rate for C = 1000.00: train 0.150 test 0.168
Error rate for C = 10000.00: train 0.117 test 0.129
/Users/marlenearredondo/opt/anaconda3/lib/python3.9/site-packages/sklearn/sv
m/_base.py:1206: ConvergenceWarning: Liblinear failed to converge, increase
the number of iterations.
  warnings.warn(
```

We will be choosing C= 1 since it is the one with the lowest training error and test error.

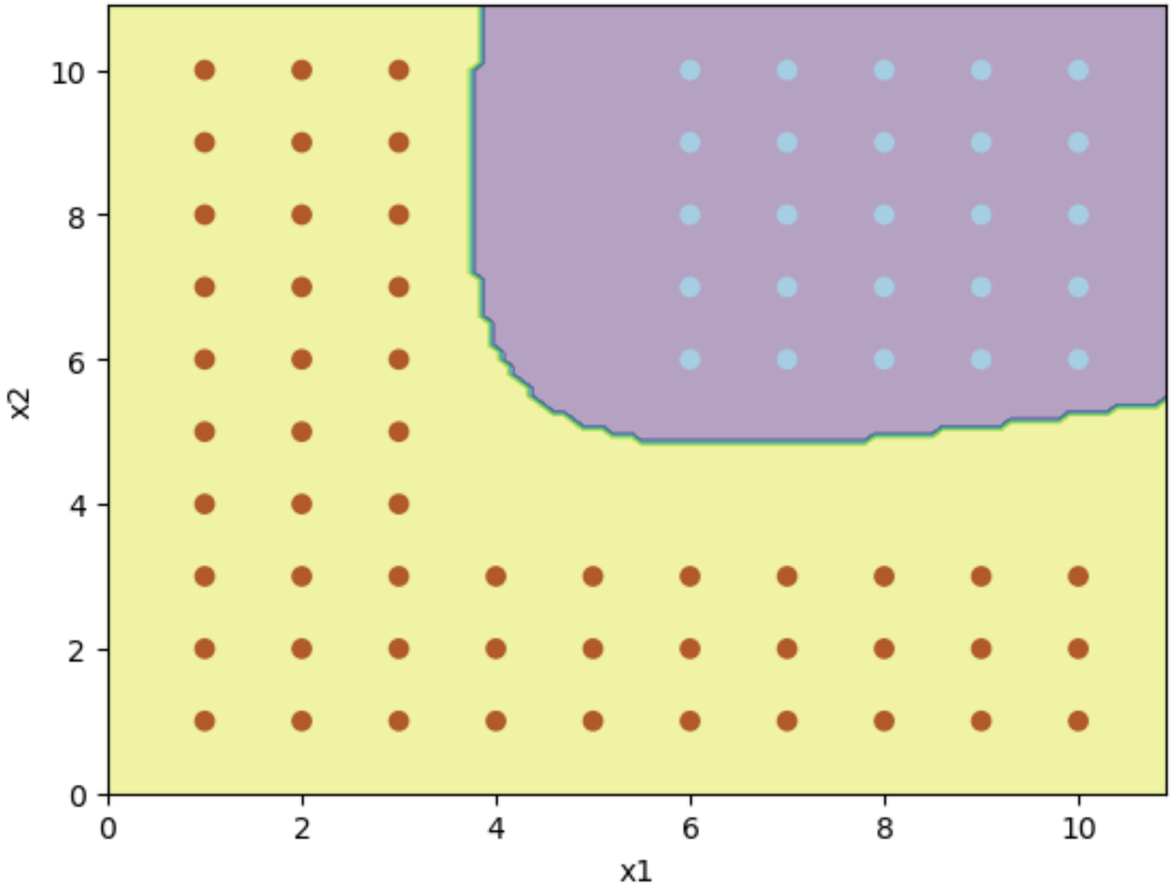
c)

The test error is 0.121 or 12.1%. The data is not linearly separable, since the highest test accuracy achieved is only 0.879.

Kernel Perceptron

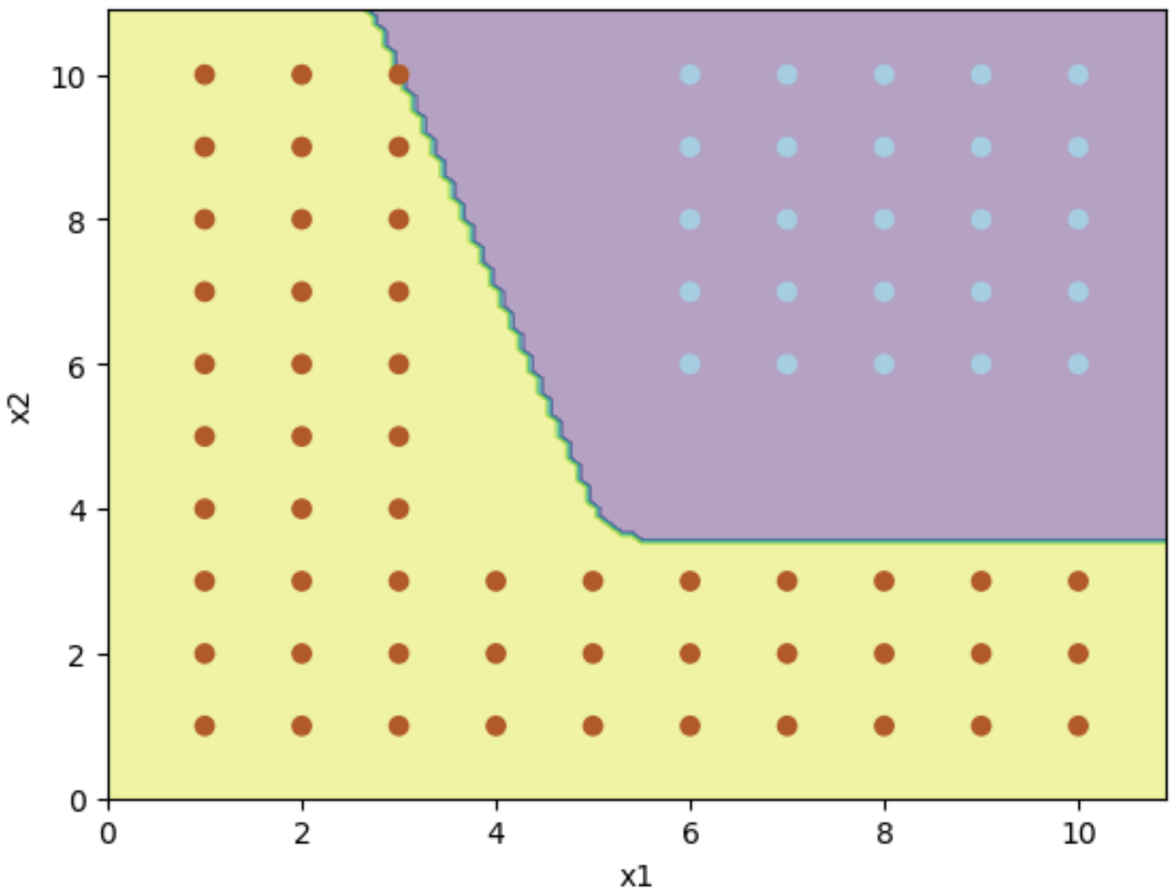
DATASET 1

In [10]:
```
# Load the data
data = np.loadtxt('data1.txt')
X = data[:, :2]
y = data[:, 2]

# Define the quadratic kernel function
def quadratic_kernel(X, Z):
    return np.square(np.dot(X, Z.T) + 1)

# Define the kernel Perceptron algorithm
def kernel_perceptron(X, y, kernel_func, kernel_params, num_iters=100):
    n = X.shape[0]
    alpha = np.zeros(n)
    K = kernel_func(X, X, *kernel_params)
    for _ in range(num_iters):
        for i in range(n):
            if np.sign(np.dot(K[i], alpha * y)) != y[i]:
                alpha[i] += 1
    return alpha

# Run the kernel Perceptron algorithm with the quadratic kernel
alpha = kernel_perceptron(X, y, quadratic_kernel, [])

# Define the plot_decision_boundary function
def plot_decision_boundary(X, y, alpha, kernel_func, kernel_params):
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.1),
                         np.arange(y_min, y_max, 0.1))
    meshgrid = np.c_[xx.ravel(), yy.ravel()]
    K = kernel_func(X, meshgrid, *kernel_params)
    preds = np.sign(np.dot(K.T, alpha * y))
    plt.contourf(xx, yy, preds.reshape(xx.shape), alpha=0.4)
    plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.Paired)
    plt.xlabel('x1')
    plt.ylabel('x2')
    plt.show()

# Plot the decision boundary
plot_decision_boundary(X, y, alpha, quadratic_kernel, [])
```

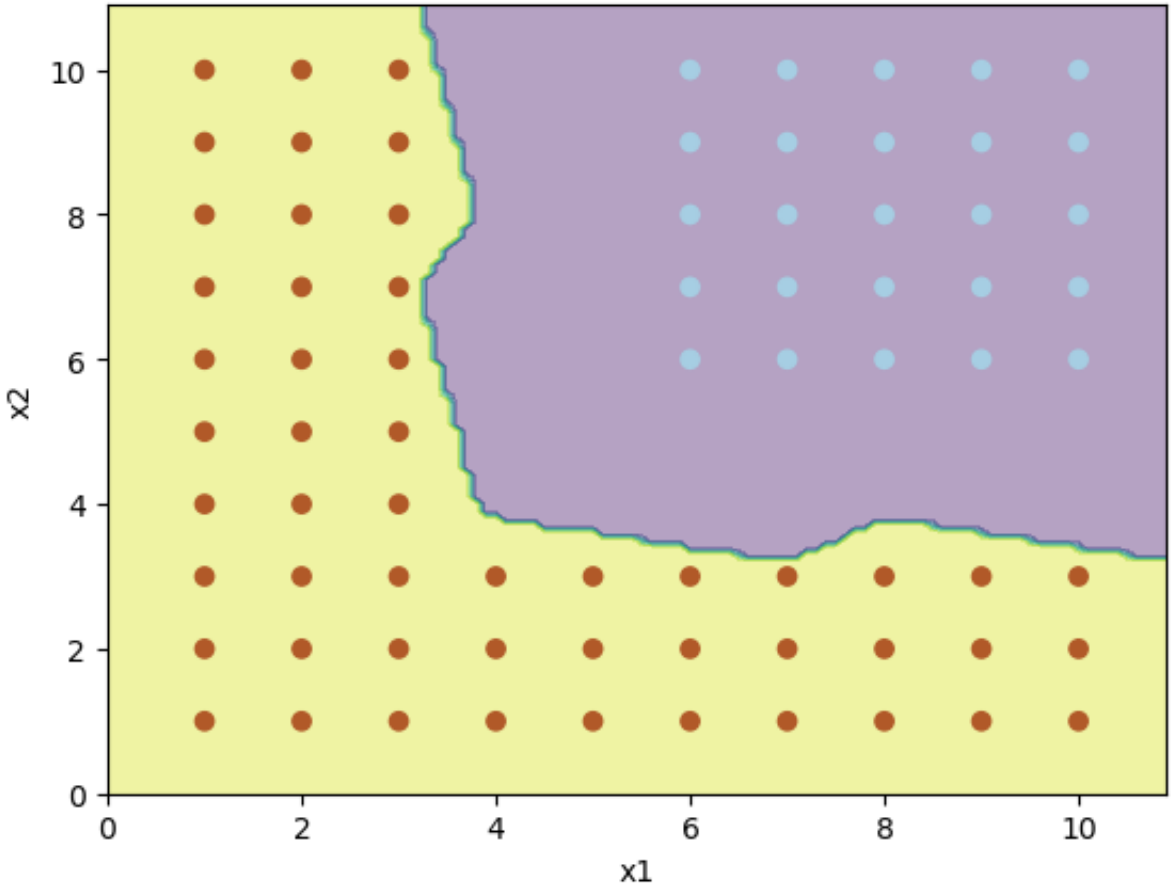


RBF

In [11]:
```
# Define the RBF kernel function
def rbf_kernel(X, Z, sigma=1):
    dist_sq = np.sum(X**2, axis=1, keepdims=True) + np.sum(Z**2, axis=1) - 2
    return np.exp(-dist_sq / (2 * sigma ** 2))

# Run the kernel Perceptron algorithm with the RBF kernel
alpha = kernel_perceptron(X, y, rbf_kernel, [1])

# Plot the decision boundary
plot_decision_boundary(X, y, alpha, rbf_kernel, [1])
```

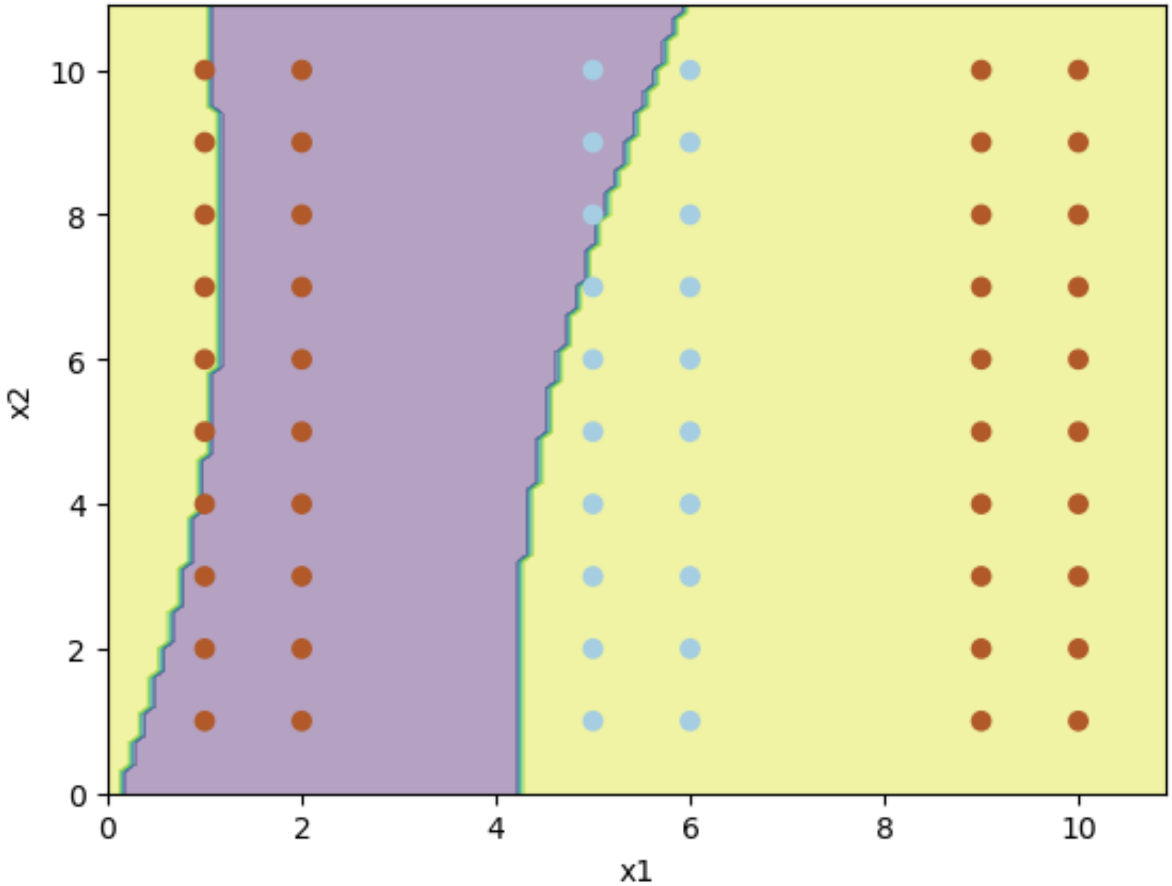


In [12]:
```
# Run the kernel Perceptron algorithm with the RBF kernel
alpha = kernel_perceptron(X, y, rbf_kernel, [0.1])

# Plot the decision boundary
plot_decision_boundary(X, y, alpha, rbf_kernel, [0.1])
```

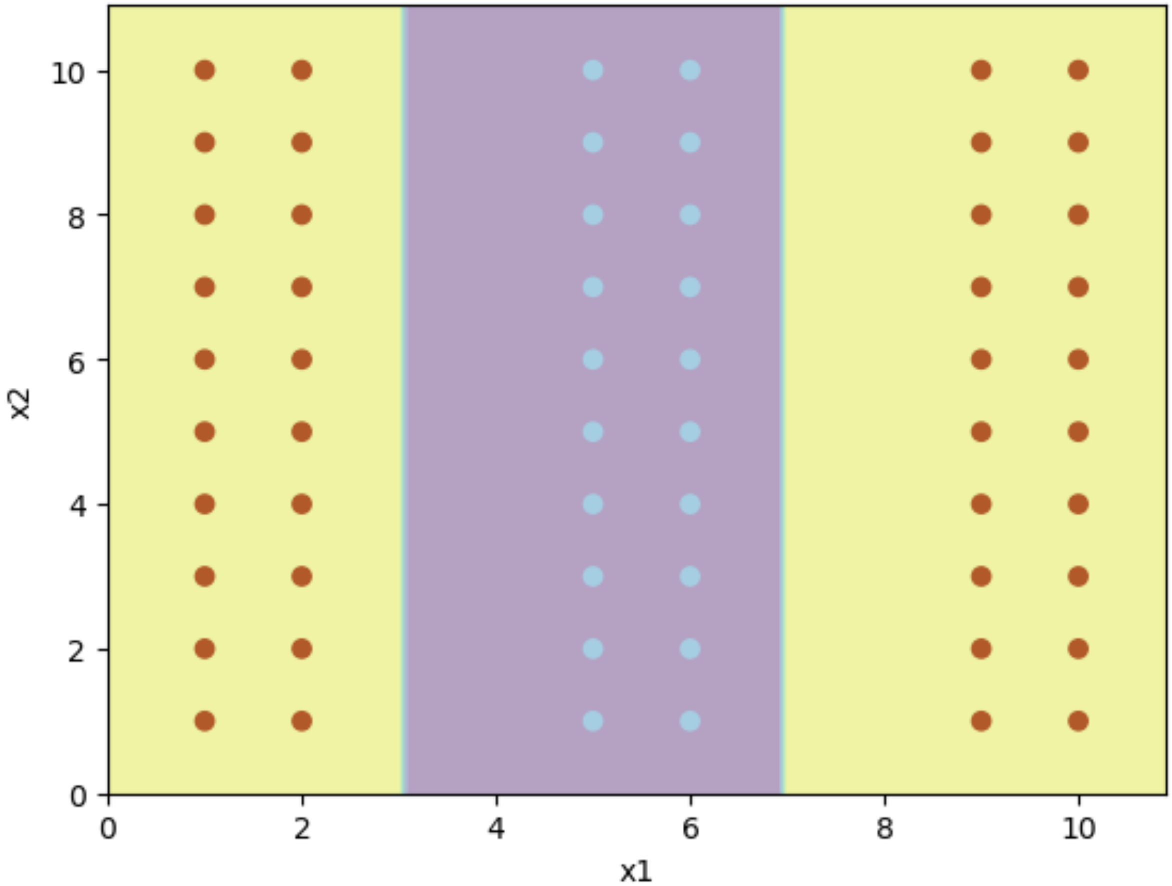


DATASET 2

In [13]:
```
# Load the data
data2 = np.loadtxt('data2.txt')
X = data2[:, :2]
y = data2[:, 2]
# Run the kernel Perceptron algorithm with the quadratic kernel
alpha = kernel_perceptron(X, y, quadratic_kernel, [])
# Plot the decision boundary
plot_decision_boundary(X, y, alpha, quadratic_kernel, [])
```



In [14]:
```
# Run the kernel Perceptron algorithm with the RBF kernel
alpha = kernel_perceptron(X, y, rbf_kernel, [1])

# Plot the decision boundary
plot_decision_boundary(X, y, alpha, rbf_kernel, [1])
```



Multiclass SVM

In [15]:
```
x = mnist.data
y = mnist.target
```

In [16]:
```
x_train ,x_test, y_train, y_test =  train_test_split(x, y, test_size=10000,
```

In [17]:
```
# Define list of tradeoff parameter values
C_values = [0.01, 0.1, 1.0, 10.0, 100.0]

# Iterate over C values and train Linear SVM
for C in C_values:
    svm = LinearSVC(C=C, loss='hinge', random_state=42)
    svm.fit(x_train, y_train)

    # Calculate training and test accuracy
    train_acc = svm.score(x_train, y_train)
    test_acc = svm.score(x_test, y_test)

    # Print results
    print(f'C = {C}: training accuracy = {train_acc:.2f}, test accuracy = {t
```

```
/Users/marlenearredondo/opt/anaconda3/lib/python3.9/site-packages/sklearn/sv
m/_base.py:1206: ConvergenceWarning: Liblinear failed to converge, increase
the number of iterations.
  warnings.warn(
C = 0.01: training accuracy = 0.86, test accuracy = 0.85
/Users/marlenearredondo/opt/anaconda3/lib/python3.9/site-packages/sklearn/sv
m/_base.py:1206: ConvergenceWarning: Liblinear failed to converge, increase
the number of iterations.
  warnings.warn(
C = 0.1: training accuracy = 0.86, test accuracy = 0.85
/Users/marlenearredondo/opt/anaconda3/lib/python3.9/site-packages/sklearn/sv
m/_base.py:1206: ConvergenceWarning: Liblinear failed to converge, increase
the number of iterations.
  warnings.warn(
C = 1.0: training accuracy = 0.86, test accuracy = 0.85
/Users/marlenearredondo/opt/anaconda3/lib/python3.9/site-packages/sklearn/sv
m/_base.py:1206: ConvergenceWarning: Liblinear failed to converge, increase
the number of iterations.
  warnings.warn(
C = 10.0: training accuracy = 0.86, test accuracy = 0.85
C = 100.0: training accuracy = 0.86, test accuracy = 0.85
/Users/marlenearredondo/opt/anaconda3/lib/python3.9/site-packages/sklearn/sv
m/_base.py:1206: ConvergenceWarning: Liblinear failed to converge, increase
the number of iterations.
  warnings.warn(
```

In [18]:

```
# Train Kernel SVM with quadratic kernel
svm = SVC(kernel='poly', degree=2, C=1.0, random_state=42)
svm.fit(x_train, y_train)

# Calculate training and test accuracy
train_acc = svm.score(x_train, y_train)
test_acc = svm.score(x_test, y_test)

# Calculate number of support vectors
n_support = svm.n_support_

# Print results
print(f'Quadratic Kernel SVM: training accuracy = {train_acc:.2f}, test accu
```

```
Quadratic Kernel SVM: training accuracy = 0.99, test accuracy = 0.97, number
of support vectors = 10073
```

In [ ]: