

hw4_4

February 23, 2023

```
[99]: import numpy as np
import random as rm
import pandas as pd
import csv
import sys
from sklearn import preprocessing
# from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt
```

```
[21]: def getData(fname, dimension):
    # create a new array to store the data
    data = np.empty([0,dimension])
    label = []
    with open(fname, mode='r') as file:
        # reading the CSV file
        csvFile = csv.reader(file)

        # displaying the contents of the CSV file
        for lines in csvFile:
            data = np.row_stack((data,[float(lines[0]), float(lines[1])]))
            label.append(float(lines[2]))
    label = np.array(label)
    return (data, label)
```

```
[24]: # Obtain data
xdata1_train, ydata1_train = getData('dataset1_train.csv', 2)
xdata1_test, ydata1_test = getData('dataset1_test.csv', 2)

xdata2_train, ydata2_train = getData('dataset2_train.csv', 2)
xdata2_test, ydata2_test = getData('dataset2_test.csv', 2)

xdata3_train, ydata3_train = getData('dataset3_train.csv', 2)
xdata3_test, ydata3_test = getData('dataset3_test.csv', 2)

## preprocessing
# Standarize
```

```

scaler1_train = preprocessing.StandardScaler().fit(xdata1_train)
scaler2_train = preprocessing.StandardScaler().fit(xdata2_train)
scaler3_train = preprocessing.StandardScaler().fit(xdata3_train)

xdata1_train_scaled = scaler1_train.transform(xdata1_train)
xdata1_test_scaled = scaler1_train.transform(xdata1_test)

xdata2_train_scaled = scaler2_train.transform(xdata2_train)
xdata2_test_scaled = scaler2_train.transform(xdata2_test)

xdata3_train_scaled = scaler3_train.transform(xdata3_train)
xdata3_test_scaled = scaler3_train.transform(xdata3_test)

```

```

[101]: #####
## EE559 HW4, Prof. Chugg
#####

def plotDecBoundaries_Nonlinear(feature, labels, weight, non_linear_trans,
    ↪ predictor, fsize=(6,4), legend_on = False):

    '''
    Plot the decision boundaries and data points for any binary classifiers

    feature: origianl2D feautre, N x 2 array:
        N: number of data points
        2: number of features
    labels: class lables correspond to feature, N x 1 array: [0,0,1,1,0,0,...]
        N: number of data points
    legend_on: add the legend in the plot. potentially slower for datasets with
    ↪ large number of clases and data points
    -----
    You need to write the following two functions

    non_linear_trans: your custom non-linear transforation function.
        <feature_nonlinear> = non_linear_trans(<feature_original>),
        Input: <feature_original>, Nx2 array,
        Output: <feature_nonlinear>: Nx? array.
        if no nonlinear transformation performs, then,
        let non_linear_trans = lambda x:x, which just output your original
    ↪ feature

    predictor: your custom predictor.
        <predictions> = predictor(<feature>)
        Input: <feature> Nx? array.
        Output: <predictions> binary labels, i.e., array ([0,1,0,0,1...])
    '''

```

If you don't want write custom functions, you can modify this plot function, based on your need, do non-linear transformation and class prediction inside this plot function.

```

'''
labels = labels.astype(int)

# Set the feature range for plotting
max_x = np.ceil(max(feature[:, 0])) + 1
min_x = np.floor(min(feature[:, 0])) - 1
max_y = np.ceil(max(feature[:, 1])) + 1
min_y = np.floor(min(feature[:, 1])) - 1

xrange = (min_x, max_x)
yrange = (min_y, max_y)

# step size for how finely you want to visualize the decision boundary.
inc = 0.05

# generate grid coordinates. this will be the basis of the decision
# boundary visualization.
(x, y) = np.meshgrid(np.arange(xrange[0], xrange[1]+inc/100, inc), np.
    ↳arange(yrange[0], yrange[1]+inc/100, inc))

# size of the (x, y) image, which will also be the size of the
# decision boundary image that is used as the plot background.
image_size = x.shape
xy = np.hstack( (x.reshape(x.shape[0]*x.shape[1], 1, order='F'), y.
    ↳reshape(y.shape[0]*y.shape[1], 1, order='F')) ) # make (x,y) pairs as a
    ↳bunch of row vectors.

'''
You should write the custom functions, non_linear_trans and predictor
'''

# apply non-linear transformation to all points in the map (not only data
    ↳points)
xy = non_linear_trans(xy)
# predict the class of all points in the map
# pred_label = predictor(xy)

pred_label = predictor(xy, weight)

# reshape the idx (which contains the class label) into an image.
decisionmap = pred_label.reshape(image_size, order='F')

```

```

# documentation: https://matplotlib.org/stable/api/_as_gen/matplotlib.
↳pyplot.plot.html
symbols_ar = np.array(['rx', 'bo', 'ms', '
↳'cd', 'gp', 'y*', 'kx', 'gP', 'r+', 'bh'])
#show the image, give each coordinate a color according to its class label
plt.figure(figsize=fsize)

plt.imshow(decisionmap, extent=[xrange[0], xrange[1], yrange[0],
↳yrange[1]], origin='lower', aspect='auto')

# plot the class data.
plot_index = 0
class_list = []
class_list_name = [] #for legend
for cur_label in np.unique(labels):
    # print(cur_label, plot_index, np.sum(label_train == cur_label))
    d1, = plt.plot(feature[labels == cur_label, 0], feature[labels ==
↳cur_label, 1], symbols_ar[plot_index])

    if legend_on:
        class_list.append(d1)
        class_list_name.append('Class '+str(plot_index))
        l = plt.legend(class_list, class_list_name, loc=2)
        plt.gca().add_artist(l)

    plot_index = plot_index + 1

plt.show()

```

```

[20]: #####
## EE559 HW1, Prof. Jenkins
## Created by Arindam Jati
## Tested in Python 3.6.3, OSX El Capitan, and subsequent versions
#####

import numpy as np
import matplotlib.pyplot as plt
from scipy.spatial.distance import cdist

def plotDecBoundaries(training, label_train, w):
    # Plot the decision boundaries and data points for perceptron learning
    ↳classification result
    # training: training data
    # label_train: class labels correspond to training data
    # w: weight vector
    nclass = max(np.unique(label_train))

```

```

# Set the feature range for plotting
max_x = np.ceil(max(training[:, 0])) + 1
min_x = np.floor(min(training[:, 0])) - 1
max_y = np.ceil(max(training[:, 1])) + 1
min_y = np.floor(min(training[:, 1])) - 1

xrange = (min_x, max_x)
yrange = (min_y, max_y)

# step size for how finely you want to visualize the decision boundary.
inc = 0.01

# generate grid coordinates. this will be the basis of the decision
# boundary visualization.
(x, y) = np.meshgrid(np.arange(xrange[0], xrange[1] + inc / 100, inc),
                     np.arange(yrange[0], yrange[1] + inc / 100, inc))

# size of the (x, y) image, which will also be the size of the
# decision boundary image that is used as the plot background.
image_size = x.shape
xy = np.hstack((x.reshape(x.shape[0] * x.shape[1], 1, order='F'),
                y.reshape(y.shape[0] * y.shape[1], 1, order='F'))) # make_
↪(x,y) pairs as a bunch of row vectors.

# distance measure evaluations for each (x,y) pair.
aug = np.zeros(np.shape(xy)[0]) + 1
xy_aug = np.concatenate((aug[:, None], xy), axis=1)
pred_label = np.zeros(np.shape(xy)[0])

for i in range(np.shape(xy)[0]):
    if w.T @ xy_aug[i] > 0:
        pred_label[i] = 1
    else:
        pred_label[i] = 2

decisionmap = pred_label.reshape(image_size, order='F')

plt.imshow(decisionmap, extent=[xrange[0], xrange[1], yrange[0],
↪yrange[1]], origin='lower')

# plot the class training data.
plt.plot(training[label_train == 1, 0], training[label_train == 1, 1], 'rx')
plt.plot(training[label_train == 2, 0], training[label_train == 2, 1], 'go')

l = plt.legend(('Class 1', 'Class 2'), loc=2)
plt.gca().add_artist(l)

```

```
# plot the class mean vector.
plt.show()
```

```
[50]: def shuffle(xdata, ydata):
    newX = np.copy(xdata)
    newY = np.copy(ydata)
    N = len(newX)
    shuff = np.random.permutation(N)
    for i in range(N):
        newX[i] = xdata[shuff[i]]
        newY[i] = ydata[shuff[i]]
    return (newX, newY)
```

```
[41]: def perceptronLearning(data, label, w0, eta = 1, maxEpochs = 100):
    """
    data: (N, D) data array, non-augmented format with labels(1.0, 2.0)
    eta: learning rate (constant)
    maxEpochs: max number of passes through the data. Halts if reach the max_
    ↪ epoch
    """

    N, D = data.shape
    wHat = np.copy(w0)

    minJ = sys.float_info.max
    finalWHat = np.copy(w0)
    i1 = False

    for m in range(1, maxEpochs + 1):
        # 1. shuffle
        shuffledData, shuffledLabel = shuffle(data, label)

        # 2. Augment and reflected
        z = (-1.0) ** (shuffledLabel + 1)
        # z = shuffledLabel
        dataAug = np.ones((N, D + 1))
        dataAug[:, 1:] = shuffledData
        zData = (dataAug.T * z).T

        for n in range(0, N):
            condition = np.dot(wHat, zData[n])
            index = (m - 1) * N + n

            # compute new J(w)
            gx_matrix = np.dot(wHat, zData.T)
```

```

        gx_matrix = gx_matrix * -1
        loss = np.sum(gx_matrix > 0)
        if( loss < minJ ):
            minJ = loss
            finalWHat = np.copy(wHat)

        if(condition <= 0):
            wHat = wHat + eta * zData[n]

    if loss == 0:
        print("i1 reach. Data is linearly separable")
        i1 = True
        break
    if(not i1):
        print("i2 reach.")
    print("Weight matrix is:" , finalWHat)
    print("Min J is:" , minJ)

    return finalWHat

```

```

[134]: def accuracy(data, label, wHat):
        '''
        data: non augment data
        '''
        N, D = data.shape
        z = (-1.0) ** (label + 1)
        wHat = np.copy(wHat)
        dataAug = np.ones((N, D + 1))
        dataAug[:, 1:] = data
        zData = (dataAug.T * z).T

        count = 0
        for i in range(N):
            if np.dot(wHat ,zData[i]) > 0:
                count += 1
        return (count) / N * 100

```

```

[42]: def linear_classification():
        weight = np.ones(3)

        weight1 = perceptronLearning(xdata1_train_scaled, ydata1_train, weight)

        plotDecBoundaries(xdata1_train_scaled, ydata1_train, weight1)
        acc_train1 = accuracy(xdata1_train_scaled, ydata1_train, weight1)
        print("Accuracy for training data 1 is: ", acc_train1, "%")

        plotDecBoundaries(xdata1_test_scaled, ydata1_test, weight1)

```

```

acc_test1 = accuracy(xdata1_test_scaled, ydata1_test, weight1)
print("Accuracy for testing data 1 is: ", acc_test1, "%")

weight2 = perceptronLearning(xdata2_train_scaled, ydata2_train, weight)

plotDecBoundaries(xdata2_train_scaled, ydata2_train, weight2)
acc_train2 = accuracy(xdata2_train_scaled, ydata2_train, weight2)
print("Accuracy for training data 2 is: ", acc_train2, "%")

plotDecBoundaries(xdata2_test_scaled, ydata2_test, weight2)
acc_test2 = accuracy(xdata2_test_scaled, ydata2_test, weight2)
print("Accuracy for testing data 2 is: ", acc_test2, "%")

weight3 = perceptronLearning(xdata3_train_scaled, ydata3_train, weight)

plotDecBoundaries(xdata3_train_scaled, ydata3_train, weight3)
acc_train3 = accuracy(xdata3_train_scaled, ydata3_train, weight3)
print("Accuracy for training data 3 is: ", acc_train3, "%")

plotDecBoundaries(xdata3_test_scaled, ydata3_test, weight3)
acc_test3 = accuracy(xdata3_test_scaled, ydata3_test, weight3)
print("Accuracy for testing data 3 is: ", acc_test3, "%")

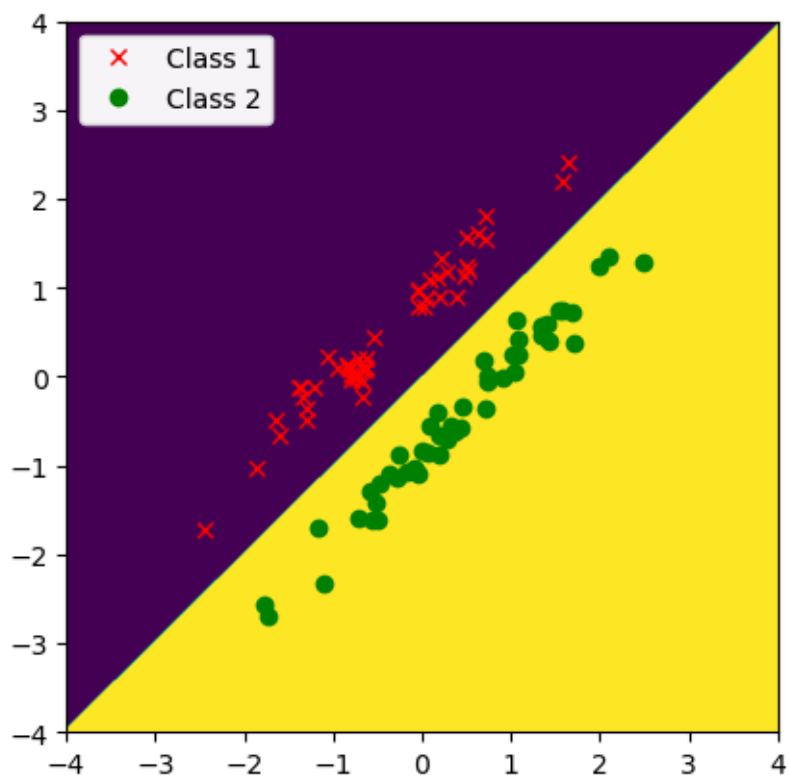
```

```
[49]: linear_classification()
```

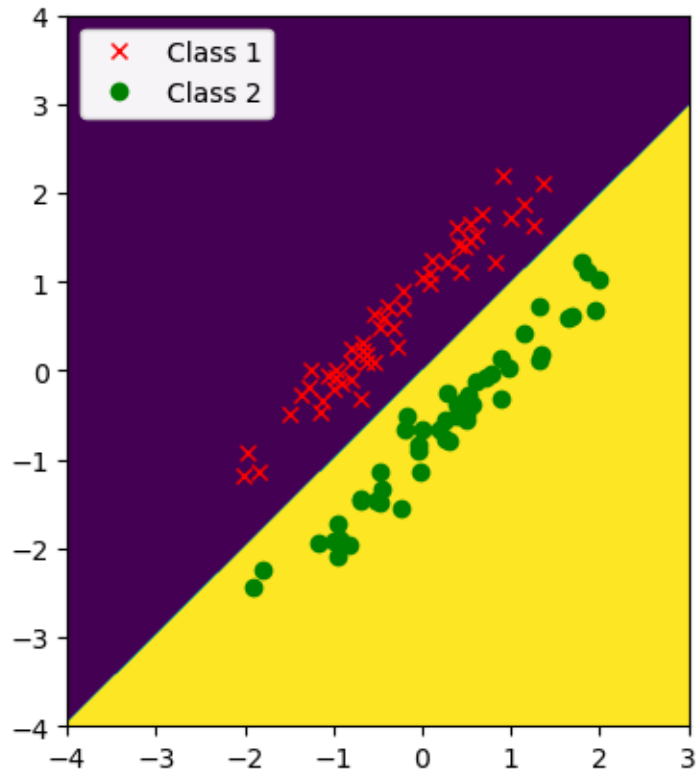
```

i1 reach. Data is linearly separable
Weight matrix is: [ 0.          -2.09552505  2.11386957]
Min J is: 0

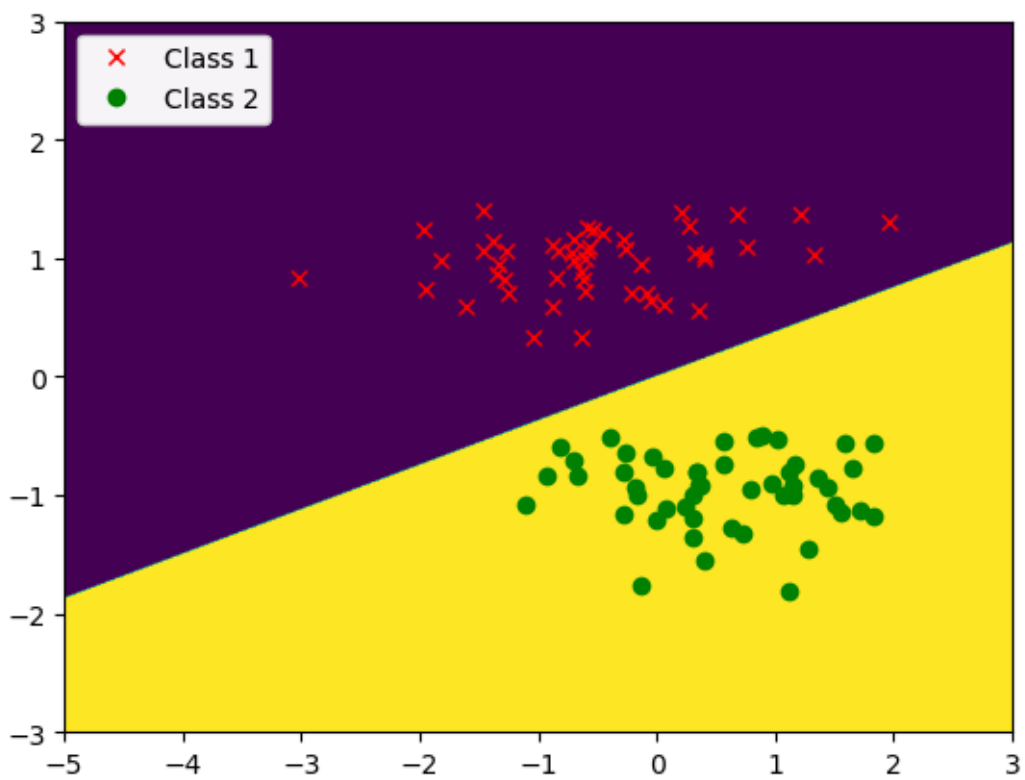
```

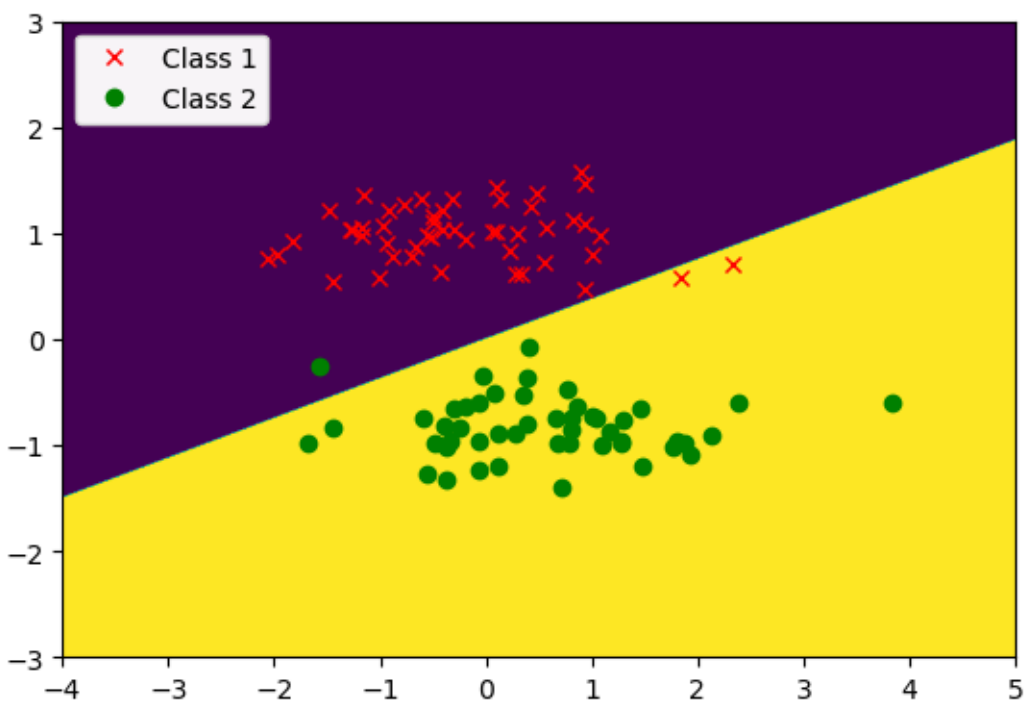
Accuracy for training data 1 is: 100.0 %



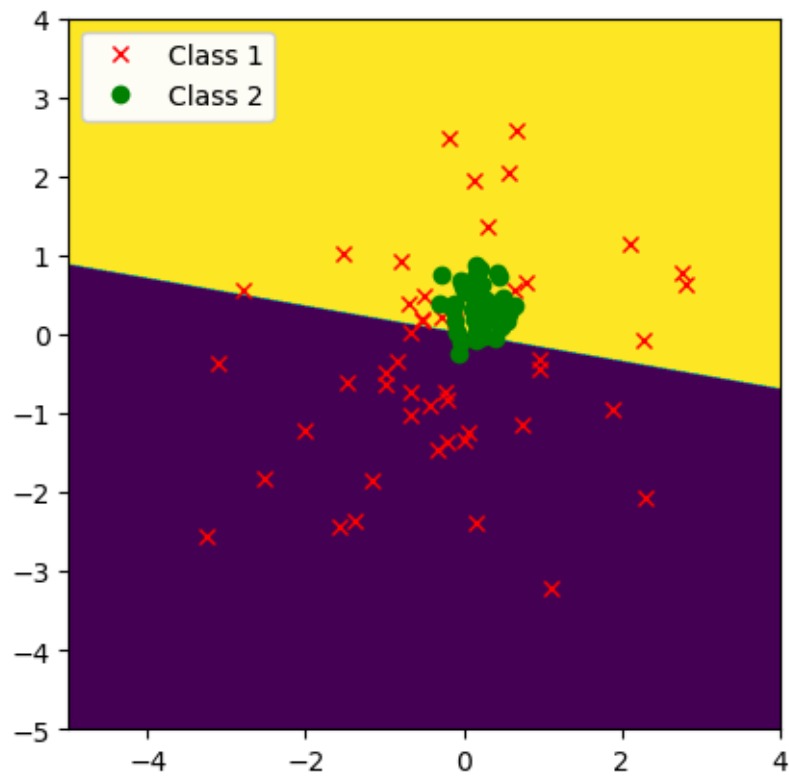
```
Accuracy for testing data 1 is: 100.0 %  
i1 reach. Data is linearly separable  
Weight matrix is: [ 0.          -0.58388862  1.55803305]  
Min J is: 0
```



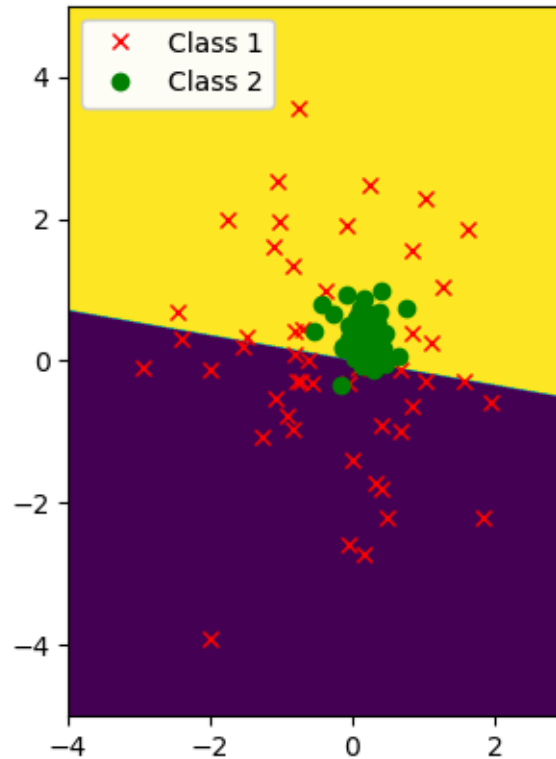
Accuracy for training data 2 is: 100.0 %



Accuracy for testing data 2 is: 96.96969696969697 %
i2 reach.
Weight matrix is: [0. -0.00949884 -0.05430716]
Min J is: 1.0980766158415043



Accuracy for training data 3 is: 74.74747474747475 %



Accuracy for testing data 3 is: 75.75757575757575 %

```
[46]: def run_10_linear():
    weight = np.ones(3)

    accuracy_train1 = np.zeros(10)
    accuracy_test1 = np.zeros(10)

    accuracy_train2 = np.zeros(10)
    accuracy_test2 = np.zeros(10)

    accuracy_train3 = np.zeros(10)
    accuracy_test3 = np.zeros(10)

    for i in range(10):

        weight1 = perceptronLearning(xdata1_train_scaled, ydata1_train, weight)
        accuracy_train1[i] = accuracy(xdata1_train_scaled, ydata1_train,
        ↪weight1)
        accuracy_test1[i] = accuracy(xdata1_test_scaled, ydata1_test, weight1)

        weight2 = perceptronLearning(xdata2_train_scaled, ydata2_train, weight)
```

```

        accuracy_train2[i] = accuracy(xdata2_train_scaled, ydata2_train,
↪weight2)
        accuracy_test2[i] = accuracy(xdata2_test_scaled, ydata2_test, weight2)

        weight3 = perceptronLearning(xdata3_train_scaled, ydata3_train, weight)

        accuracy_train3[i] = accuracy(xdata3_train_scaled, ydata3_train,
↪weight3)
        accuracy_test3[i] = accuracy(xdata3_test_scaled, ydata3_test, weight3)

        print("The mean of accuracy for the training data 1 is :", np.
↪mean(accuracy_train1), "%")
        print("The mean of accuracy for the testing data 1 is :", np.
↪mean(accuracy_test1), "%")
        print("The std of accuracy for the training data 1 is :", np.
↪std(accuracy_train1))
        print("The std of accuracy for the testing data 1 is :", np.
↪std(accuracy_test1))
        print("\n")
        print("The mean of accuracy for the training data 2 is :", np.
↪mean(accuracy_train2), "%")
        print("The mean of accuracy for the testing data 2 is :", np.
↪mean(accuracy_test2), "%")
        print("The std of accuracy for the training data 2 is :", np.
↪std(accuracy_train2))
        print("The std of accuracy for the testing data 2 is :", np.
↪std(accuracy_test2))
        print("\n")
        print("The mean of accuracy for the training data 3 is :", np.
↪mean(accuracy_train3), "%")
        print("The mean of accuracy for the testing data 3 is :", np.
↪mean(accuracy_test3), "%")
        print("The std of accuracy for the training data 3 is :", np.
↪std(accuracy_train3))
        print("The std of accuracy for the testing data 3 is :", np.
↪std(accuracy_test3))

```

[47]: run_10_linear()

```

i1 reach. Data is linearly separable
Weight matrix is: [ 0.          -3.52481552  3.50225845]
Min J is: 0
Accuracy: 100.0 %
Accuracy: 100.0 %
i1 reach. Data is linearly separable

```

Weight matrix is: [0. -0.153983 1.91133793]
 Min J is: 0
 Accuracy: 100.0 %
 Accuracy: 100.0 %
 i2 reach.
 Weight matrix is: [0. -0.03543798 0.02314264]
 Min J is: 1.225349059469708
 Accuracy: 53.535353535353536 %
 Accuracy: 46.464646464646464 %
 i1 reach. Data is linearly separable
 Weight matrix is: [0. -0.70940556 0.61468782]
 Min J is: 0
 Accuracy: 100.0 %
 Accuracy: 100.0 %
 i1 reach. Data is linearly separable
 Weight matrix is: [1. -0.00874213 2.01612548]
 Min J is: 0
 Accuracy: 100.0 %
 Accuracy: 94.94949494949495 %
 i2 reach.
 Weight matrix is: [0. 0.01542985 -0.04514293]
 Min J is: 0.9244808153892415
 Accuracy: 67.67676767676768 %
 Accuracy: 67.67676767676768 %
 i1 reach. Data is linearly separable
 Weight matrix is: [0. -1.82931335 2.21360549]
 Min J is: 0
 Accuracy: 100.0 %
 Accuracy: 100.0 %
 i1 reach. Data is linearly separable
 Weight matrix is: [0. -0.0247717 1.52412521]
 Min J is: 0
 Accuracy: 100.0 %
 Accuracy: 100.0 %
 i2 reach.
 Weight matrix is: [0. -0.00547731 -0.03518379]
 Min J is: 0.7079176600142785
 Accuracy: 74.74747474747475 %
 Accuracy: 76.76767676767676 %
 i1 reach. Data is linearly separable
 Weight matrix is: [0. -2.62304313 2.08564099]
 Min J is: 0
 Accuracy: 100.0 %
 Accuracy: 98.98989898989899 %
 i1 reach. Data is linearly separable
 Weight matrix is: [0. 0.26978371 2.32760393]
 Min J is: 0
 Accuracy: 100.0 %

```

Accuracy: 100.0 %
i2 reach.
Weight matrix is: [ 0.          -0.02013134 -0.08456868]
Min J is: 1.7473778985840835
Accuracy: 74.74747474747475 %
Accuracy: 75.75757575757575 %
i1 reach. Data is linearly separable
Weight matrix is: [ 0.          -3.15542044  3.06719648]
Min J is: 0
Accuracy: 100.0 %
Accuracy: 100.0 %
i1 reach. Data is linearly separable
Weight matrix is: [ 0.          -0.11943942  2.8135676 ]
Min J is: 0
Accuracy: 100.0 %
Accuracy: 100.0 %
i2 reach.
Weight matrix is: [0.          0.00075788 0.00378839]
Min J is: 0.21160802058431658
Accuracy: 25.25252525252523 %
Accuracy: 25.25252525252523 %
i1 reach. Data is linearly separable
Weight matrix is: [-1.          -3.04622363  2.69569424]
Min J is: 0
Accuracy: 100.0 %
Accuracy: 97.97979797979798 %
i1 reach. Data is linearly separable
Weight matrix is: [ 0.          -0.16897988  1.73812694]
Min J is: 0
Accuracy: 100.0 %
Accuracy: 100.0 %
i2 reach.
Weight matrix is: [ 0.          -0.00152637 -0.03085155]
Min J is: 0.6063426545733005
Accuracy: 72.72727272727273 %
Accuracy: 72.72727272727273 %
i1 reach. Data is linearly separable
Weight matrix is: [ 0.          -0.8561001  1.17339924]
Min J is: 0
Accuracy: 100.0 %
Accuracy: 100.0 %
i1 reach. Data is linearly separable
Weight matrix is: [ 0.          -0.71908314  2.12224397]
Min J is: 0
Accuracy: 100.0 %
Accuracy: 96.96969696969697 %
i2 reach.
Weight matrix is: [ 0.          -0.06606927  0.04531855]

```


Min J is: 2.3581858834547638
 Accuracy: 51.5151515151516 %
 Accuracy: 44.4444444444444 %
 i1 reach. Data is linearly separable
 Weight matrix is: [0. -2.9573079 2.78736034]
 Min J is: 0
 Accuracy: 100.0 %
 Accuracy: 100.0 %
 i1 reach. Data is linearly separable
 Weight matrix is: [0. -0.8240739 3.03591588]
 Min J is: 0
 Accuracy: 100.0 %
 Accuracy: 98.9898989898989 %
 i2 reach.
 Weight matrix is: [0. 0.0035151 0.02225743]
 Min J is: 1.2207465410559901
 Accuracy: 25.2525252525253 %
 Accuracy: 23.2323232323232 %
 i1 reach. Data is linearly separable
 Weight matrix is: [0. -2.55001648 1.93274305]
 Min J is: 0
 Accuracy: 100.0 %
 Accuracy: 96.9696969696969 %
 i1 reach. Data is linearly separable
 Weight matrix is: [0. -0.55098496 2.14489367]
 Min J is: 0
 Accuracy: 100.0 %
 Accuracy: 98.9898989898989 %
 i2 reach.
 Weight matrix is: [0. 0.00923827 0.00011089]
 Min J is: 0.3852259545630488
 Accuracy: 31.3131313131315 %
 Accuracy: 34.3434343434343 %
 i1 reach. Data is linearly separable
 Weight matrix is: [0. -3.53397421 3.33131186]
 Min J is: 0
 Accuracy: 100.0 %
 Accuracy: 100.0 %
 i1 reach. Data is linearly separable
 Weight matrix is: [0. -0.8279724 2.17683291]
 Min J is: 0
 Accuracy: 100.0 %
 Accuracy: 96.9696969696969 %
 i2 reach.
 Weight matrix is: [0. -0.05487033 -0.06343353]
 Min J is: 1.8042490928978543
 Accuracy: 77.7777777777779 %
 Accuracy: 76.7676767676767 %

The mean of accuracy for the training data 1 is : 100.0 %
The mean of accuracy for the testing data 1 is : 99.3939393939394 %
The std of accuracy for the training data 1 is : 0.0
The std of accuracy for the testing data 1 is : 1.0301049522409669

The mean of accuracy for the training data 2 is : 100.0 %
The mean of accuracy for the testing data 2 is : 98.68686868686868 %
The std of accuracy for the training data 2 is : 0.0
The std of accuracy for the testing data 2 is : 1.693237839822244

The mean of accuracy for the training data 3 is : 55.45454545454546 %
The mean of accuracy for the testing data 3 is : 54.34343434343434 %
The std of accuracy for the training data 3 is : 20.28996011281075
The std of accuracy for the testing data 3 is : 20.866838683353016

```
[46]: def nonlinear_quadratic_mapping(xdata):
    """
    xdata : nonagument data N * D
    """
    N, D = xdata.shape
    D_prime = 1/2 * (D ** 2 + 3 * D)
    xdata_mapping = []

    # Augment xdata
    xdata_aug = np.ones((N, D + 1))
    xdata_aug[:,1:] = xdata

    for i in range(D + 1):
        for j in range(i, D + 1):
            xdata_mapping.append(xdata_aug[:,i] * xdata_aug[:,j])

    xdata_mapping = np.array(xdata_mapping).T.reshape(N, int(D_prime) + 1)

    return xdata_mapping
```

```
[53]: def predictor(xdata, weight):
    """
    xdata : augment data N * D_primes + 1 matrix
    weight : augmnet weight D_primes + 1 vector
    """
    gx = np.dot(xdata, weight)
    predict_label = np.ones(len(gx))
    for i in range(len(gx)):
        if gx[i] < 0:
            predict_label[i] += 1
```

```
return predict_label
```

```
[115]: # Preprocessing: create an initial weight and get the transformed data
N, D = xdata1_train_scaled.shape
D_prime = 1/2 * (D ** 2 + 3 * D)
weight_prime = np.ones(int(D_prime) + 1)

xdata1_train_mapping = nonlinear_quadratic_mapping(xdata1_train_scaled)
xdata1_test_mapping = nonlinear_quadratic_mapping(xdata1_test_scaled)

xdata2_train_mapping = nonlinear_quadratic_mapping(xdata2_train_scaled)
xdata2_test_mapping = nonlinear_quadratic_mapping(xdata2_test_scaled)

xdata3_train_mapping = nonlinear_quadratic_mapping(xdata3_train_scaled)
xdata3_test_mapping = nonlinear_quadratic_mapping(xdata3_test_scaled)

[117]: def quadratic_classification():
    weight_prime_1 = perceptronLearning(xdata1_train_mapping[:,1:],
    ↪ ydata1_train, weight_prime)
    plotDecBoundaries_Nonlinear(xdata1_train_scaled, ydata1_train,
    ↪ weight_prime_1, nonlinear_quadratic_mapping, predictor)
    print("Accuracy rate for training data 1 :", accuracy(xdata1_train_mapping[:,1:],
    ↪ ydata1_train, weight_prime_1), "%")

    plotDecBoundaries_Nonlinear(xdata1_test_scaled, ydata1_test,
    ↪ weight_prime_1, nonlinear_quadratic_mapping, predictor)
    print("Accuracy rate for testing data 1 :", accuracy(xdata1_test_mapping[:,1:],
    ↪ ydata1_test, weight_prime_1), "%")
    print("\n")

    weight_prime_2 = perceptronLearning(xdata2_train_mapping[:,1:],
    ↪ ydata2_train, weight_prime)
    plotDecBoundaries_Nonlinear(xdata2_train_scaled, ydata2_train,
    ↪ weight_prime_2, nonlinear_quadratic_mapping, predictor)
    print("Accuracy rate for training data 2 :", accuracy(xdata2_train_mapping[:,1:],
    ↪ ydata2_train, weight_prime_2), "%")

    plotDecBoundaries_Nonlinear(xdata2_test_scaled, ydata2_test,
    ↪ weight_prime_2, nonlinear_quadratic_mapping, predictor)
    print("Accuracy rate for testing data 2 :", accuracy(xdata2_test_mapping[:,1:],
    ↪ ydata2_test, weight_prime_2), "%")
    print("\n")

    weight_prime_3 = perceptronLearning(xdata3_train_mapping[:,1:],
    ↪ ydata3_train, weight_prime)
```

```

    plotDecBoundaries_Nonlinear(xdata3_train_scaled, ydata3_train,
↪weight_prime_3, nonlinear_quadratic_mapping,predictor)
    print("Accuracy rate for training data 3 :", accuracy(xdata3_train_mapping[:
↪,1:], ydata3_train, weight_prime_3), "%")

    plotDecBoundaries_Nonlinear(xdata3_test_scaled, ydata3_test,
↪weight_prime_3, nonlinear_quadratic_mapping,predictor)
    print("Accuracy rate for testing data 2 :",accuracy(xdata3_test_mapping[:
↪,1:], ydata3_test, weight_prime_3), "%")

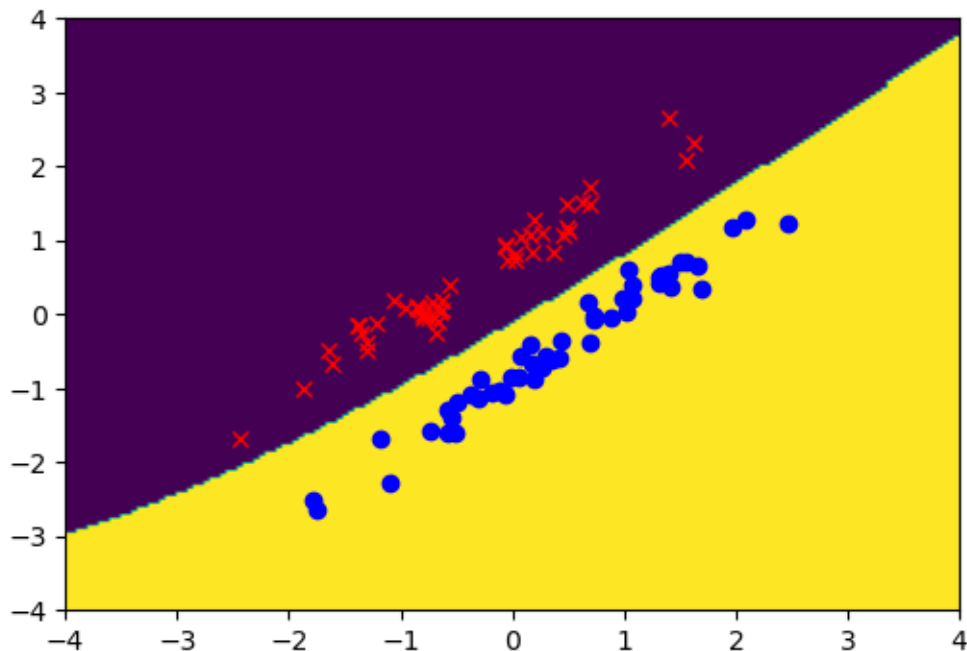
quadratic_classification()

```

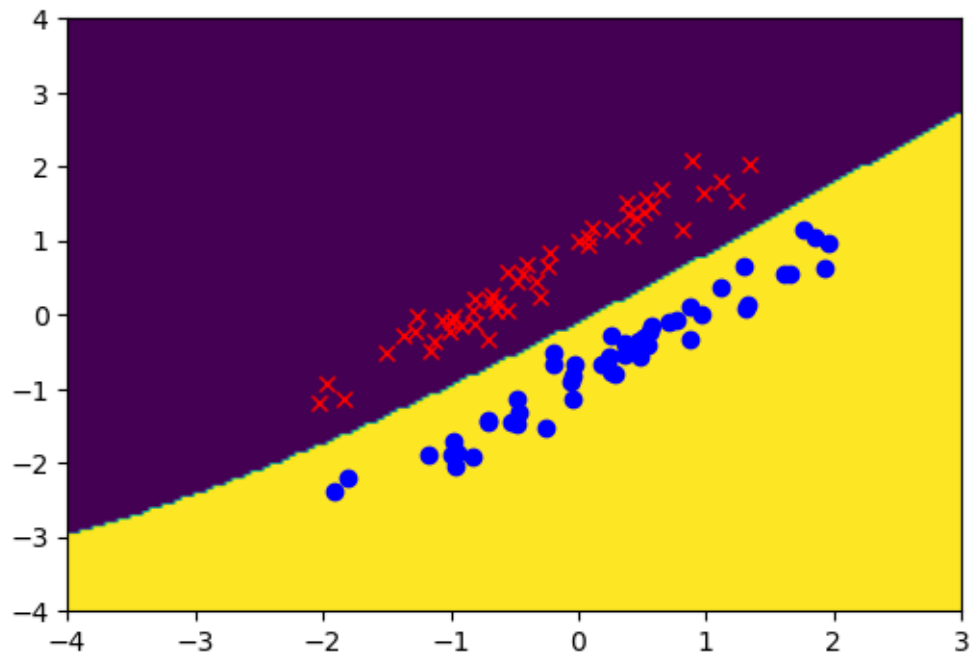
i1 reach. Data is linearly separable

Weight matrix is: [1. -7.66323998 8.7350138 -0.6293219 0.07998281
0.39826552]

Min J is: 0



Accuracy rate for training data 1 : 100.0 %

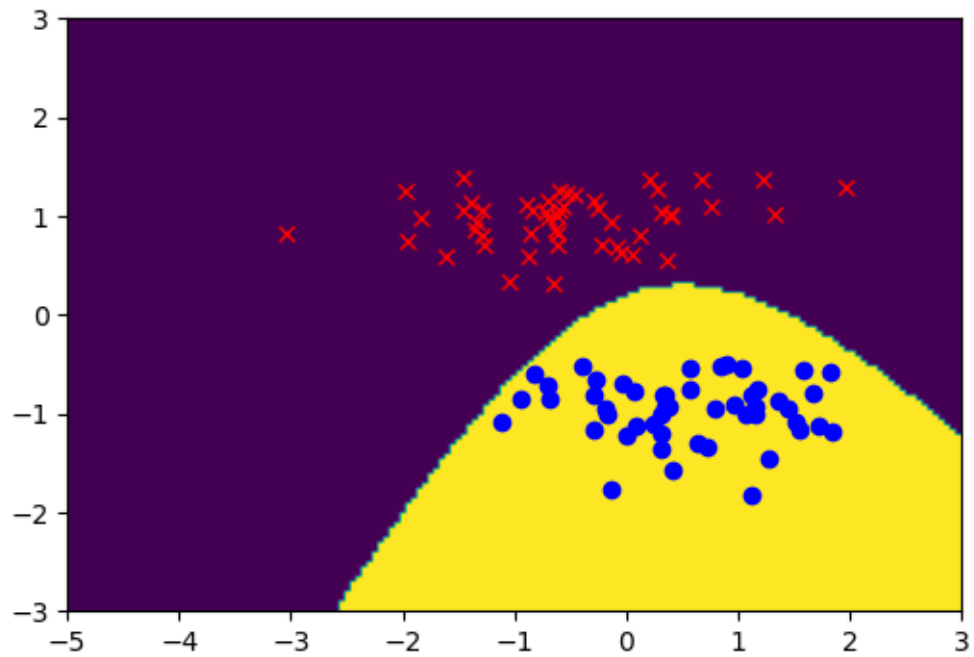


Accuracy rate for testing data 1 : 100.0 %

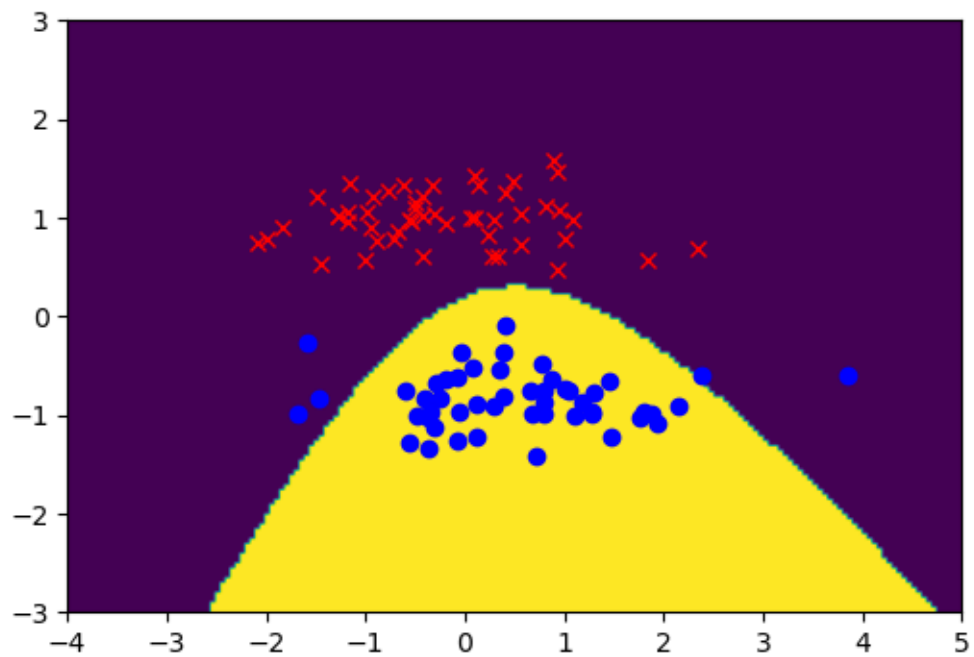
i1 reach. Data is linearly separable

Weight matrix is: [-1. -2.23120865 5.25456869 1.97562164 0.68693003
-0.86031339]

Min J is: 0



Accuracy rate for training data 2 : 100.0 %

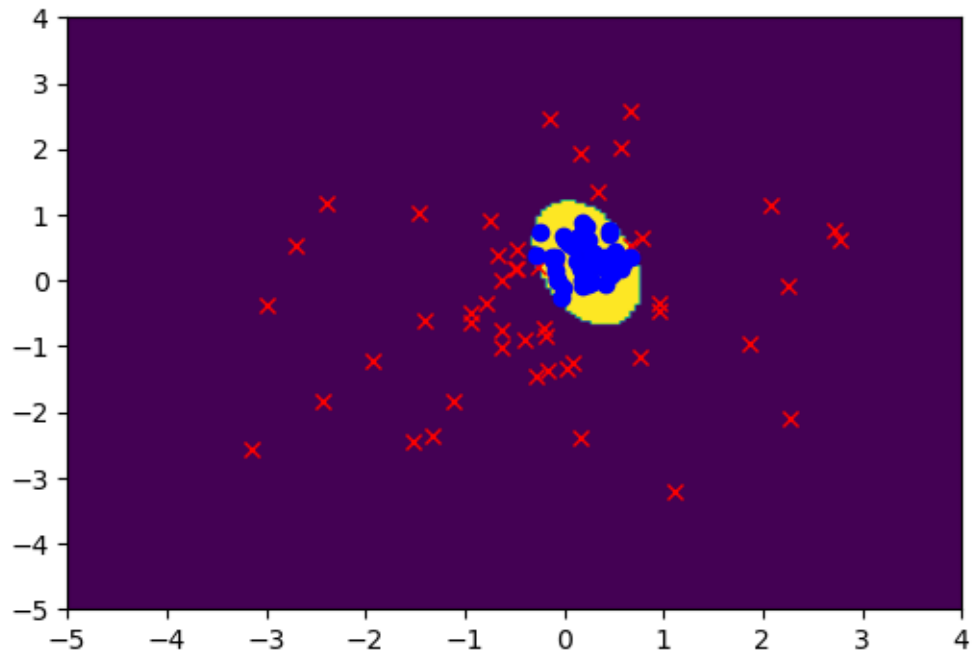


Accuracy rate for testing data 2 : 95.0 %

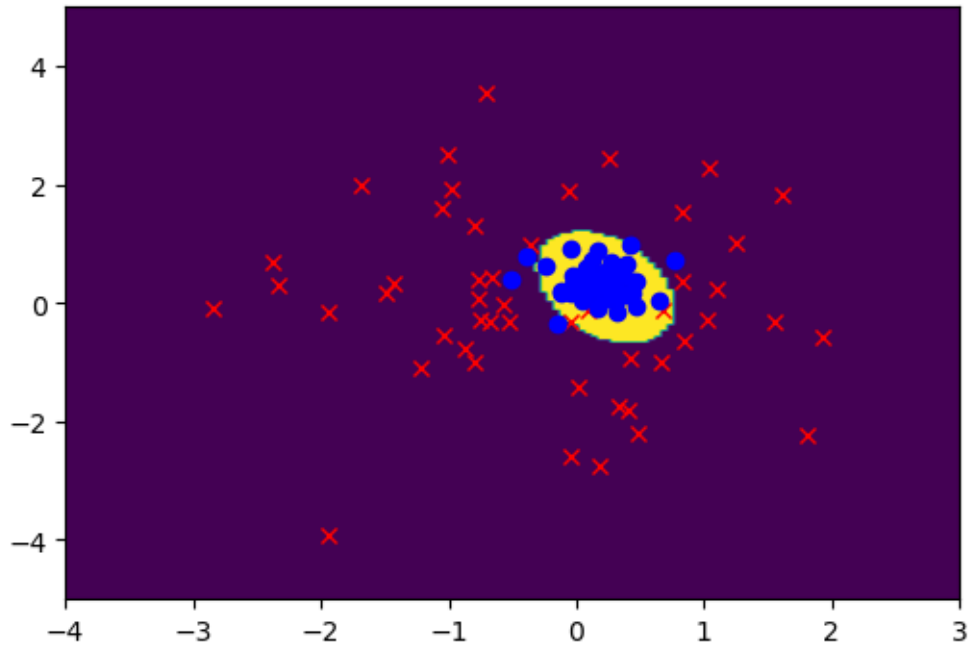
i2 reach.

Weight matrix is: [-1. -3.86819942 -1.75654489 6.64888088 2.73306438
2.14209377]

Min J is: 2



Accuracy rate for training data 3 : 98.0 %



Accuracy rate for testing data 2 : 93.0 %

```
[151]: def run_10_quadratic():

    accuracy_train1 = np.zeros(10)
    accuracy_test1 = np.zeros(10)

    accuracy_train2 = np.zeros(10)
    accuracy_test2 = np.zeros(10)

    accuracy_train3 = np.zeros(10)
    accuracy_test3 = np.zeros(10)

    for i in range(10):
        weight_prime_1 = perceptronLearning(xdata1_train_mapping[:,1:],
        ↪ ydata1_train, weight_prime)
        weight_prime_2 = perceptronLearning(xdata2_train_mapping[:,1:],
        ↪ ydata2_train, weight_prime)
        weight_prime_3 = perceptronLearning(xdata3_train_mapping[:,1:],
        ↪ ydata3_train, weight_prime)

        accuracy_train1[i] = accuracy(xdata1_train_mapping[:,1:], ydata1_train,
        ↪ weight_prime_1)
```



```

        accuracy_test1[i] = accuracy(xdata1_test_mapping[:,1:], ydata1_test,
↪weight_prime_1)

        accuracy_train2[i] = accuracy(xdata2_train_mapping[:,1:], ydata2_train,
↪weight_prime_2)
        accuracy_test2[i] = accuracy(xdata2_test_mapping[:,1:], ydata2_test,
↪weight_prime_2)

        accuracy_train3[i] = accuracy(xdata3_train_mapping[:,1:], ydata3_train,
↪weight_prime_3)
        accuracy_test3[i] = accuracy(xdata3_test_mapping[:,1:], ydata3_test,
↪weight_prime_3)

    print("The mean of accuracy for the training data 1 is :", np.
↪mean(accuracy_train1), "%")
    print("The mean of accuracy for the testing data 1 is :", np.
↪mean(accuracy_test1), "%")
    print("The std of accuracy for the training data 1 is :", np.
↪std(accuracy_train1))
    print("The std of accuracy for the testing data 1 is :", np.
↪std(accuracy_test1))
    print("\n")
    print("The mean of accuracy for the training data 2 is :", np.
↪mean(accuracy_train2), "%")
    print("The mean of accuracy for the testing data 2 is :", np.
↪mean(accuracy_test2), "%")
    print("The std of accuracy for the training data 2 is :", np.
↪std(accuracy_train2))
    print("The std of accuracy for the testing data 2 is :", np.
↪std(accuracy_test2))
    print("\n")
    print("The mean of accuracy for the training data 3 is :", np.
↪mean(accuracy_train3), "%")
    print("The mean of accuracy for the testing data 3 is :", np.
↪mean(accuracy_test3), "%")
    print("The std of accuracy for the training data 3 is :", np.
↪std(accuracy_train3))
    print("The std of accuracy for the testing data 3 is :", np.
↪std(accuracy_test3))

run_10_quadratic()

```

i1 reach. Data is linearly separable

Weight matrix is: [-2. -14.37311012 13.15115425 1.63934042

```

1.76400355
  -1.6091564 ]
Min J is: 0
i1 reach. Data is linearly separable
Weight matrix is: [-1.          -1.12933732  8.36800763 -0.61211116  1.35499821
0.73636262]
Min J is: 0
i2 reach.
Weight matrix is: [-1.          -3.4761001  -1.45441152  6.59679023  2.31566287
2.15822252]
Min J is: 2
i1 reach. Data is linearly separable
Weight matrix is: [  2.          -14.15847923  14.66324413   1.22354825
-0.046896
  0.4798287 ]
Min J is: 0
i1 reach. Data is linearly separable
Weight matrix is: [  0.          -1.61757337  5.627948   1.48937733  0.60085165
-2.34590643]
Min J is: 0
i2 reach.
Weight matrix is: [-1.          -4.19888817 -2.37571506  7.47137138  3.37030289
2.47273643]
Min J is: 2
i1 reach. Data is linearly separable
Weight matrix is: [ -1.          -10.17791242   9.71769867  -1.75130274
0.66391675
  1.3751295 ]
Min J is: 0
i1 reach. Data is linearly separable
Weight matrix is: [-2.          -5.60474477  9.60734888  1.6357955  -1.22020546
-0.36435115]
Min J is: 0
i2 reach.
Weight matrix is: [-1.          -3.78220458 -1.58370942  6.55332203  3.43743455
2.24246561]
Min J is: 2
i1 reach. Data is linearly separable
Weight matrix is: [  0.          -12.84893434  13.18091731  -0.78387109
0.33967744
  -0.83122035]
Min J is: 0
i1 reach. Data is linearly separable
Weight matrix is: [-1.          -0.03200588  3.47644623  0.29721   0.50448169
0.12918827]
Min J is: 0
i2 reach.
Weight matrix is: [-1.          -4.14520293 -1.8375234   6.85592478  3.101276

```

```

2.24553155]
Min J is: 2
i1 reach. Data is linearly separable
Weight matrix is: [ -3.          -11.92021152  16.67762127   1.02970221
0.76314312
-1.20286702]
Min J is: 0
i1 reach. Data is linearly separable
Weight matrix is: [-1.          -3.45807899  9.29396772  2.60828993  1.5040015
0.0470346 ]
Min J is: 0
i2 reach.
Weight matrix is: [-1.          -3.91056102 -1.95672937  6.96921311  3.15700443
2.67384104]
Min J is: 2
i1 reach. Data is linearly separable
Weight matrix is: [ 0.          -9.27378006 10.14364156  0.45487066 -0.79106118
-1.13223094]
Min J is: 0
i1 reach. Data is linearly separable
Weight matrix is: [ 0.          -1.23995226  3.58775562  0.31774209  1.05814183
-0.70960256]
Min J is: 0
i2 reach.
Weight matrix is: [-1.          -3.83404934 -1.88671455  7.14938155  2.85661554
2.82642925]
Min J is: 2
i1 reach. Data is linearly separable
Weight matrix is: [ 2.          -8.54532424  9.82140665 -0.74031233 -0.12017084
1.15082343]
Min J is: 0
i1 reach. Data is linearly separable
Weight matrix is: [-3.          -4.79736521  9.02599743  2.88052234 -0.60488379
-0.15334878]
Min J is: 0
i2 reach.
Weight matrix is: [-1.          -4.24271236 -2.27383013  7.28186138  2.8860311
2.93026998]
Min J is: 2
i1 reach. Data is linearly separable
Weight matrix is: [ -1.          -18.69572391 15.94743714 -2.34073218
1.37407603
0.86961898]
Min J is: 0
i1 reach. Data is linearly separable
Weight matrix is: [-3.          -3.81391111  7.72988259  3.82916627  1.41079492
-1.3838321 ]
Min J is: 0

```

```

i2 reach.
Weight matrix is: [-1.          -4.0317986  -2.25217464  7.39099968  3.3326658
2.70004509]
Min J is: 2
i1 reach. Data is linearly separable
Weight matrix is: [ -2.          -10.73828749   8.91720104  -0.55826933
1.11305686
-0.20159828]
Min J is: 0
i1 reach. Data is linearly separable
Weight matrix is: [ 0.          -2.31644654  6.58540398 -0.0937659  -0.3247009
1.07754565]
Min J is: 0
i2 reach.
Weight matrix is: [-1.          -3.98268869 -1.94221411  7.27036853  3.09007418
2.79148462]
Min J is: 2
i1 reach. Data is linearly separable
Weight matrix is: [ -1.          -19.69255506  19.54385602   0.29374475
1.51574983
-2.10647268]
Min J is: 0
i1 reach. Data is linearly separable
Weight matrix is: [-1.          -5.24444232 11.06485385  2.12370647  0.44534914
-2.22376636]
Min J is: 0
i2 reach.
Weight matrix is: [-1.          -4.4563076  -2.49453314  8.5177586  2.90341355
2.48982312]
Min J is: 2
The mean of accuracy for the training data 1 is : 100.0 %
The mean of accuracy for the testing data 1 is : 99.6 %
The std of accuracy for the training data 1 is : 0.0
The std of accuracy for the testing data 1 is : 0.8

The mean of accuracy for the training data 2 is : 100.0 %
The mean of accuracy for the testing data 2 is : 96.7 %
The std of accuracy for the training data 2 is : 0.0
The std of accuracy for the testing data 2 is : 1.6155494421403513

The mean of accuracy for the training data 3 is : 98.0 %
The mean of accuracy for the testing data 3 is : 92.0 %
The std of accuracy for the training data 3 is : 0.0
The std of accuracy for the testing data 3 is : 0.6324555320336759

```

```
[140]: def nonlinear_cubic_mapping(xdata):
        """
        xdata : nonagument data N * D
        """
        N, D = xdata.shape
        D_prime = 9
        xdata_mapping = []

        # Augment xdata
        xdata_aug = np.ones((N, D + 1))
        xdata_aug[:,1:] = xdata

        for i in range(D + 1):
            for j in range(i, D + 1):
                for k in range(j, D + 1):
                    xdata_mapping.append(xdata_aug[:,i] * xdata_aug[:,j] *
↪xdata_aug[:,k])

        xdata_mapping = np.array(xdata_mapping).T.reshape(N, D_prime + 1)
        return xdata_mapping

[145]: xdata1_train_mapping_cubic = nonlinear_cubic_mapping(xdata1_train_scaled)
xdata1_test_mapping_cubic = nonlinear_cubic_mapping(xdata1_test_scaled)

xdata2_train_mapping_cubic = nonlinear_cubic_mapping(xdata2_train_scaled)
xdata2_test_mapping_cubic = nonlinear_cubic_mapping(xdata2_test_scaled)

xdata3_train_mapping_cubic = nonlinear_cubic_mapping(xdata3_train_scaled)
xdata3_test_mapping_cubic = nonlinear_cubic_mapping(xdata3_test_scaled)

weight_prime_cubic = np.ones(10)

[147]: def cubic_classification():
        weight_prime_1 = perceptronLearning(xdata1_train_mapping_cubic[:,1:],
↪ydata1_train, weight_prime_cubic)
        plotDecBoundaries_Nonlinear(xdata1_train_scaled, ydata1_train,
↪weight_prime_1, nonlinear_cubic_mapping,predictor)
        print("Accuracy rate for training data 1 :
↪",accuracy(xdata1_train_mapping_cubic[:,1:], ydata1_train, weight_prime_1),
↪"%")

        plotDecBoundaries_Nonlinear(xdata1_test_scaled, ydata1_test,
↪weight_prime_1, nonlinear_cubic_mapping,predictor)
        print("Accuracy rate for testing data 1 :
↪",accuracy(xdata1_test_mapping_cubic[:,1:], ydata1_test, weight_prime_1),
↪"%")
```

```

print("\n")

weight_prime_2 = perceptronLearning(xdata2_train_mapping_cubic[:,1:],  

↪ ydata2_train, weight_prime_cubic)
plotDecBoundaries_Nonlinear(xdata2_train_scaled, ydata2_train,  

↪ weight_prime_2, nonlinear_cubic_mapping,predictor)
print("Accuracy rate for training data 2 :  

↪ ",accuracy(xdata2_train_mapping_cubic[:,1:], ydata2_train, weight_prime_2),  

↪ "%")

plotDecBoundaries_Nonlinear(xdata2_test_scaled, ydata2_test,  

↪ weight_prime_2, nonlinear_cubic_mapping,predictor)
print("Accuracy rate for testing data 2 :  

↪ ",accuracy(xdata2_test_mapping_cubic[:,1:], ydata2_test, weight_prime_2),  

↪ "%")
print("\n")

weight_prime_3 = perceptronLearning(xdata3_train_mapping_cubic[:,1:],  

↪ ydata3_train, weight_prime_cubic)
plotDecBoundaries_Nonlinear(xdata3_train_scaled, ydata3_train,  

↪ weight_prime_3, nonlinear_cubic_mapping,predictor)
print("Accuracy rate for training data 3 :",  

↪ accuracy(xdata3_train_mapping_cubic[:,1:], ydata3_train, weight_prime_3),  

↪ "%")

plotDecBoundaries_Nonlinear(xdata3_test_scaled, ydata3_test,  

↪ weight_prime_3, nonlinear_cubic_mapping,predictor)
print("Accuracy rate for testing data 2 :  

↪ ",accuracy(xdata3_test_mapping_cubic[:,1:], ydata3_test, weight_prime_3),  

↪ "%")

cubic_classification()

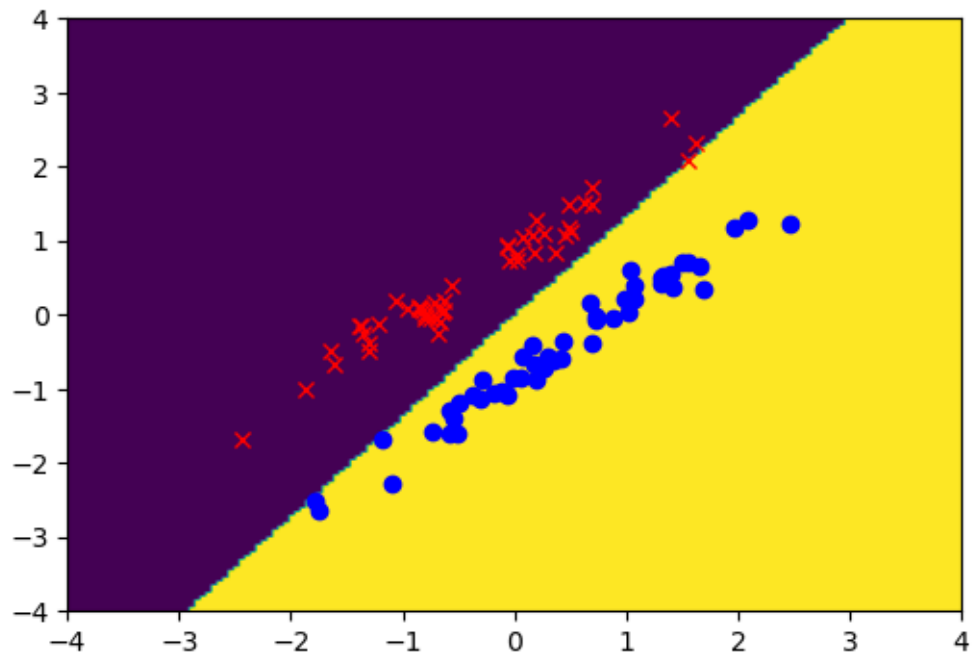
```

i1 reach. Data is linearly separable

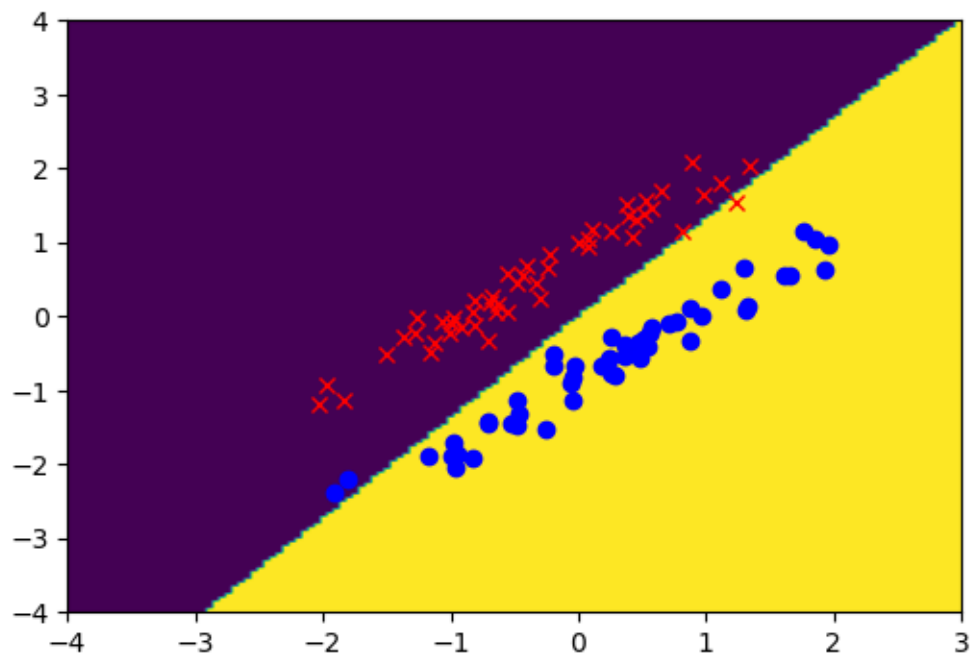
Weight matrix is: [0. -1.61543874 1.24146949 -0.28421054 0.65143742
-0.0449748

-5.97039289 -0.67180192 0.92055614 2.10049763]

Min J is: 0

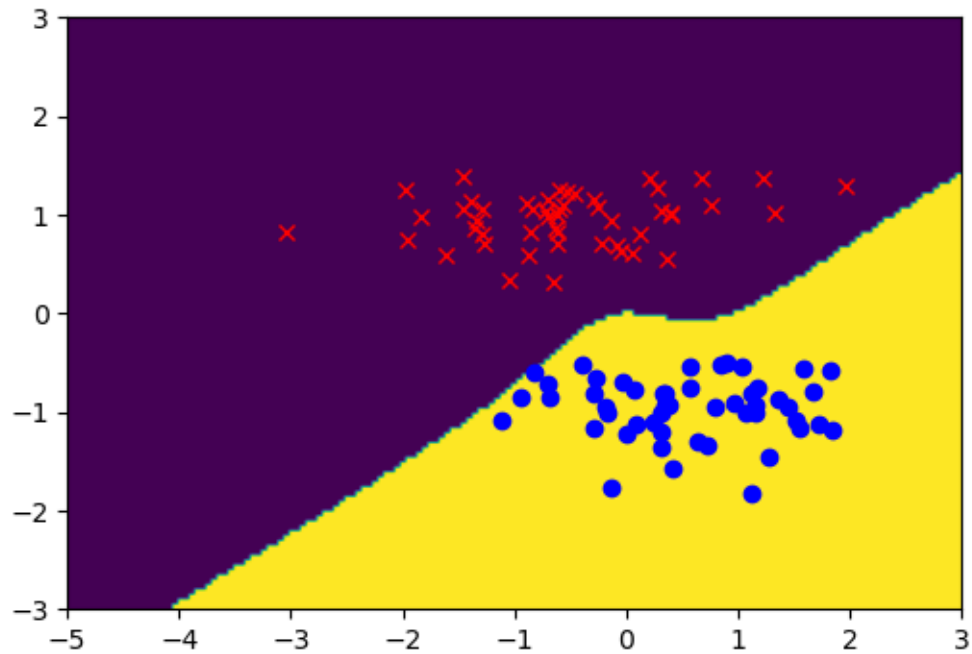


Accuracy rate for training data 1 : 100.0 %

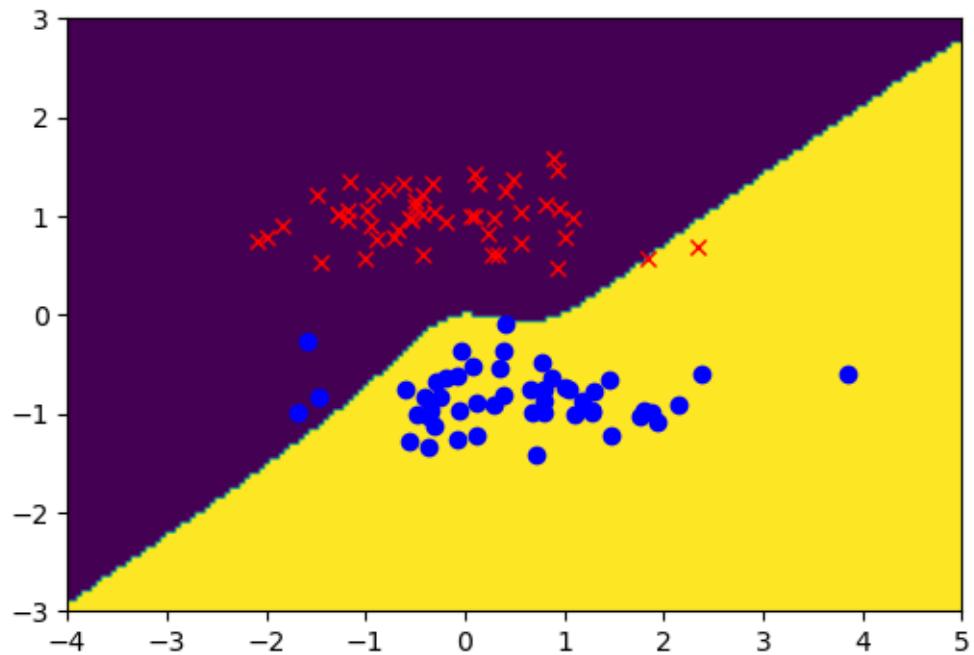


Accuracy rate for testing data 1 : 97.0 %

```
i1 reach. Data is linearly separable  
Weight matrix is: [ 0.          -0.21279609  3.29674302  3.02243026 -0.14617657  
-0.16473553  
-2.91061976  2.98469536  0.49709636  2.56366838]  
Min J is: 0
```



```
Accuracy rate for training data 2 : 100.0 %
```

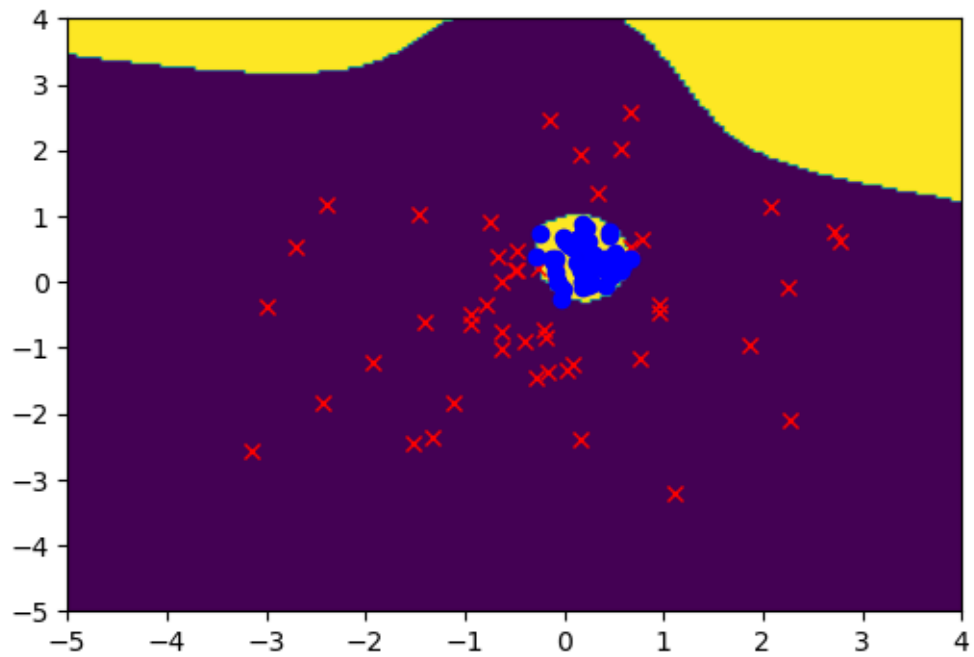
Accuracy rate for testing data 2 : 95.0 %

i2 reach.

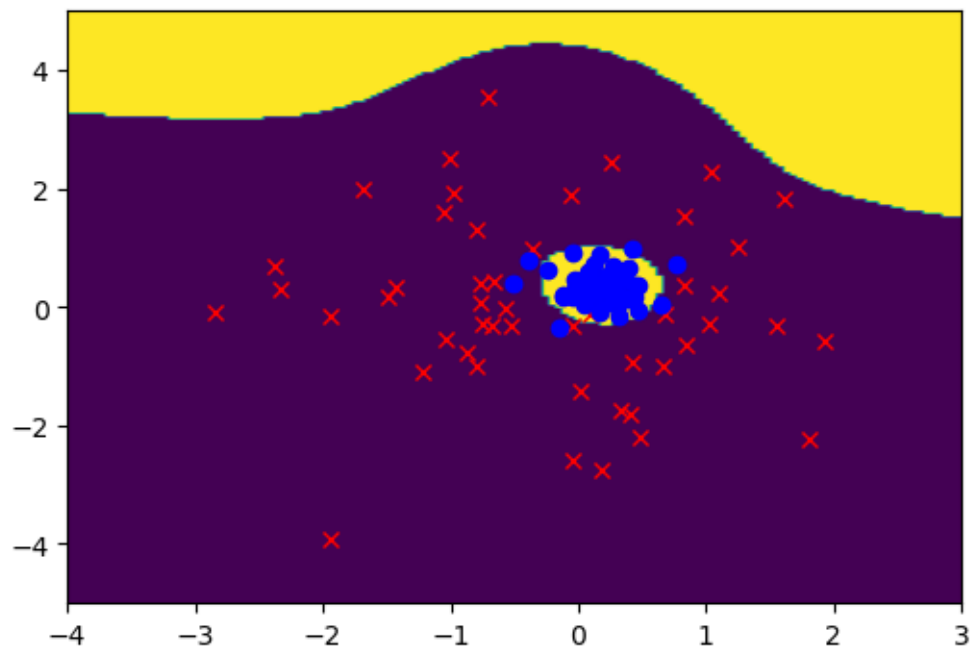
Weight matrix is: [-4. -15.39150474 -13.2885659 40.41485695
13.15847859

21.44429684 -4.17019256 -18.59555545 -3.22875261 -4.12572051]

Min J is: 3



Accuracy rate for training data 3 : 97.0 %



Accuracy rate for testing data 2 : 92.0 %

```

[152]: def run_10_cubic():

    accuracy_train1 = np.zeros(10)
    accuracy_test1 = np.zeros(10)

    accuracy_train2 = np.zeros(10)
    accuracy_test2 = np.zeros(10)

    accuracy_train3 = np.zeros(10)
    accuracy_test3 = np.zeros(10)

    for i in range(10):
        weight_prime_1 = perceptronLearning(xdata1_train_mapping_cubic[:,1:],  

        ↪ ydata1_train, weight_prime_cubic)
        weight_prime_2 = perceptronLearning(xdata2_train_mapping_cubic[:,1:],  

        ↪ ydata2_train, weight_prime_cubic)
        weight_prime_3 = perceptronLearning(xdata3_train_mapping_cubic[:,1:],  

        ↪ ydata3_train, weight_prime_cubic)

        accuracy_train1[i] = accuracy(xdata1_train_mapping_cubic[:,1:],  

        ↪ ydata1_train, weight_prime_1)
        accuracy_test1[i] = accuracy(xdata1_test_mapping_cubic[:,1:],  

        ↪ ydata1_test, weight_prime_1)

        accuracy_train2[i] = accuracy(xdata2_train_mapping_cubic[:,1:],  

        ↪ ydata2_train, weight_prime_2)
        accuracy_test2[i] = accuracy(xdata2_test_mapping_cubic[:,1:],  

        ↪ ydata2_test, weight_prime_2)

        accuracy_train3[i] = accuracy(xdata3_train_mapping_cubic[:,1:],  

        ↪ ydata3_train, weight_prime_3)
        accuracy_test3[i] = accuracy(xdata3_test_mapping_cubic[:,1:],  

        ↪ ydata3_test, weight_prime_3)

    print("The mean of accuracy for the training data 1 is :", np.  

    ↪ mean(accuracy_train1), "%")
    print("The mean of accuracy for the testing data 1 is :", np.  

    ↪ mean(accuracy_test1), "%")
    print("The std of accuracy for the training data 1 is :", np.  

    ↪ std(accuracy_train1))
    print("The std of accuracy for the testing data 1 is :", np.  

    ↪ std(accuracy_test1))
    print("\n")

```

```

    print("The mean of accuracy for the training data 2 is :", np.
↪mean(accuracy_train2), "%")
    print("The mean of accuracy for the testing data 2 is :", np.
↪mean(accuracy_test2), "%")
    print("The std of accuracy for the training data 2 is :", np.
↪std(accuracy_train2))
    print("The std of accuracy for the testing data 2 is :", np.
↪std(accuracy_test2))
    print("\n")
    print("The mean of accuracy for the training data 3 is :", np.
↪mean(accuracy_train3), "%")
    print("The mean of accuracy for the testing data 3 is :", np.
↪mean(accuracy_test3), "%")
    print("The std of accuracy for the training data 3 is :", np.
↪std(accuracy_train3))
    print("The std of accuracy for the testing data 3 is :", np.
↪std(accuracy_test3))

run_10_cubic()

```

```

i1 reach. Data is linearly separable
Weight matrix is: [-1.          -3.19325329  2.39589446 -1.20055382  3.36119949
4.26498433
-7.05501674  2.54994615  6.39661089 10.34622777]
Min J is: 0
i1 reach. Data is linearly separable
Weight matrix is: [ -1.          3.6313337   8.19029064 12.32073
1.19744872
2.39701627 -16.04666591 18.19329743  5.41832681  8.56590471]
Min J is: 0
i2 reach.
Weight matrix is: [-3.          -8.46960358 -7.13806135 20.50116809  0.26687748
13.19159749
0.60849901 -4.36401562  7.03271722 -2.80559863]
Min J is: 3
i1 reach. Data is linearly separable
Weight matrix is: [ 2.          -5.64921969  0.56262001  5.31905921
0.95184137
-2.21272912 -19.66778651 -5.53423642  3.67571804 11.93683492]
Min J is: 0
i1 reach. Data is linearly separable
Weight matrix is: [ 0.          -0.55750747  2.1550173  -1.42582952  2.79894808
-0.33406497
-2.77824761  3.80187508 -1.07781616  2.54086813]
Min J is: 0
i2 reach.
Weight matrix is: [-4.          -15.91808354 -9.14466603 38.79119612

```

```

12.57828645
  15.34553412 -3.74832756 -22.10840048  3.24412398 -3.68140448]
Min J is: 2
i1 reach. Data is linearly separable
Weight matrix is: [ 0.          -3.02101957  1.98710986 -0.36972413  0.70606103
1.03560909
-1.79173075  0.68323635  0.72181011  1.59789735]
Min J is: 0
i1 reach. Data is linearly separable
Weight matrix is: [ 2.          -0.34292846  5.02008043  4.51384146  1.72968692
1.81374949
-3.85260307 12.30376063  3.89228773  5.08513791]
Min J is: 0
i2 reach.
Weight matrix is: [-5.          -2.83157781 -6.22558909 17.38118561  0.20704133
11.78074358
-4.50066121  0.60973929  2.4032791  -0.55878507]
Min J is: 3
i1 reach. Data is linearly separable
Weight matrix is: [ 1.          0.64005397  3.76834699 -2.18061239  2.34963432
5.90976011
-9.83182478 -0.60097797  5.65682775 11.67090527]
Min J is: 0
i1 reach. Data is linearly separable
Weight matrix is: [ 0.          -0.07085133  3.22098428  2.03242838 -0.85390597
-0.14132025
-2.01863083  4.08316105  1.60179476  2.65325768]
Min J is: 0
i2 reach.
Weight matrix is: [-4.          -15.86973792 -10.41818294  31.58953964
-0.25160302
12.48664067  0.7172856  0.80188143 14.87066134 -1.07772038]
Min J is: 2
i1 reach. Data is linearly separable
Weight matrix is: [ 0.          -1.13182876  1.34379215  2.25677305  0.64389327
-1.46714345
-4.10649497 -0.92888145  1.22509077  4.5849209 ]
Min J is: 0
i1 reach. Data is linearly separable
Weight matrix is: [-1.          3.90600593  8.1814851  12.84652803
0.96545373
0.80928335 -15.22569425 17.92073856  4.19666927  8.34845111]
Min J is: 0
i2 reach.
Weight matrix is: [-5.          -18.70172676 -11.56800037 46.49445244
15.28163393
17.41366757 -5.21697907 -19.50062882 -3.11077471 -2.02951056]
Min J is: 2

```

```

i1 reach. Data is linearly separable
Weight matrix is: [ -2.          -3.94246901   3.21546696  -6.44414543
-5.02819102
  -6.67244774 -18.42215445  -2.61765968   6.93375816  15.96591516]
Min J is: 0
i1 reach. Data is linearly separable
Weight matrix is: [0.          0.02730717  1.91326997  0.05386866  1.88833116
0.16593795
  0.07970483  1.86407335  0.18871383  1.76172382]
Min J is: 0
i2 reach.
Weight matrix is: [ -5.          -19.40318163 -11.02912748  40.6493568
13.48407669
  20.67300565   0.46015311 -20.6934664   4.97114348  -3.64727937]
Min J is: 2
i1 reach. Data is linearly separable
Weight matrix is: [ -3.          -5.51054782   3.58056218  -1.10902785
-0.30612346
  -3.31618327 -21.39091947  -3.80340039   9.20881948  23.61182548]
Min J is: 0
i1 reach. Data is linearly separable
Weight matrix is: [-1.          -0.06357705   5.39249727   5.12317834   2.66272196
1.82546018
  -2.471839   10.92586178  2.95474769   4.4041211 ]
Min J is: 0
i2 reach.
Weight matrix is: [ -4.          -17.32007788  -9.83425553  31.02755758
6.80218615
  11.46717791   7.86077598  -7.86158872   5.89929833   2.187822 ]
Min J is: 2
i1 reach. Data is linearly separable
Weight matrix is: [ 0.          -4.98174567   1.63251709   0.16742558
-3.38202535
  -6.80159942 -16.31389932  -1.67852194   6.60528878  14.8436942 ]
Min J is: 0
i1 reach. Data is linearly separable
Weight matrix is: [ 0.          2.70256443   8.43848916  12.33838474
1.32326339
  2.19765038 -15.95922298  23.91317758   7.40657274   8.43858545]
Min J is: 0
i2 reach.
Weight matrix is: [ -6.          -9.56823438 -14.26393998  27.00183139
0.73618758
  24.69876161   7.07136178  -4.8830188   5.2496867   -0.38183264]
Min J is: 3
i1 reach. Data is linearly separable
Weight matrix is: [ 0.          -1.65413538  1.62578599  0.4959464   0.71093139
0.3932999

```

```

-3.43483005 -0.44417127  0.97920219  2.52965019]
Min J is: 0
i1 reach. Data is linearly separable
Weight matrix is: [ 1.          0.50646539  4.2589735   4.07942587  3.42566164
2.04938897
 0.37613022 10.13571814  3.01585047  4.06299535]
Min J is: 0
i2 reach.
Weight matrix is: [ -4.          -14.50791123  -8.68169626  27.05734734
5.20866973
 14.17115238  1.94435882  -1.42184934  4.44093119  -2.389114   ]
Min J is: 2
i1 reach. Data is linearly separable
Weight matrix is: [ 0.          -1.45829274  1.09762932 -0.21175947  0.58395041
0.36867919
 -1.80206718  0.20489922  0.53929096  1.11440734]
Min J is: 0
i1 reach. Data is linearly separable
Weight matrix is: [ 0.          0.7694296   3.86254852 -0.65941393  4.37429351
0.22175095
 -2.7856921  6.87811528 -0.06704011  4.02656855]
Min J is: 0
i2 reach.
Weight matrix is: [ -5.          -16.52750914 -11.21776605  31.56551197
7.72424132
 19.10848872  7.32112326 -12.05266387  4.33235968  -1.33057282]
Min J is: 3
The mean of accuracy for the training data 1 is : 100.0 %
The mean of accuracy for the testing data 1 is : 99.3 %
The std of accuracy for the training data 1 is : 0.0
The std of accuracy for the testing data 1 is : 1.1874342087037917

The mean of accuracy for the training data 2 is : 100.0 %
The mean of accuracy for the testing data 2 is : 96.8 %
The std of accuracy for the training data 2 is : 0.0
The std of accuracy for the testing data 2 is : 2.2271057451320084

The mean of accuracy for the training data 3 is : 97.6 %
The mean of accuracy for the testing data 3 is : 91.3 %
The std of accuracy for the training data 3 is : 0.4898979485566356
The std of accuracy for the testing data 3 is : 1.676305461424021

```

[]: