Problem a-Synthetic 1 data set

Using the first two features-f1 and f2

In [12]:

```
import numpy as np
from numpy import genfromtxt
import math as mt
import matplotlib.pyplot as plt
from scipy.spatial.distance import cdist
```

plotDecBoundaries function given

In [13]:

```
def plotDecBoundaries(training, label_train, sample_mean):

    #Plot the decision boundaries and data points for minimum distance to
    #class mean classifier
    #
    # training: traning data
    # label_train: class lables correspond to training data
    # sample_mean: mean vector for each class
    #
    # Total number of classes
    nclass =  max(np.unique(label_train))

    # Set the feature range for ploting
    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 = 0.005

    # 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

    # 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

    # distance measure evaluations for each (x,y) pair.
    dist_mat = cdist(xy, sample_mean)
    pred_label = np.argmin(dist_mat, axis=1)

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

    #show the image, give each coordinate a color according to its class label
    plt.imshow(decisionmap, extent=[xrange[0], xrange[1], yrange[0], yrange[1]], origi

    # 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')
    if nclass == 3:
        plt.plot(training[label_train == 3, 0],training[label_train == 3, 1], 'b*')

    # include legend for training data
    if nclass == 3:
        l = plt.legend(('Class 1', 'Class 2', 'Class 3'), loc=2)
    else:
        l = plt.legend(('Class 1', 'Class 2'), loc=2)
    plt.gca().add_artist(l)

    # plot the class mean vector.
    m1, = plt.plot(sample_mean[0,0], sample_mean[0,1], 'rd', markersize=12, markerface
    m2, = plt.plot(sample_mean[1,0], sample_mean[1,1], 'gd', markersize=12, markerface
```

```
    if nclass == 3:
        m3, = plt.plot(sample_mean[2,0], sample_mean[2,1], 'bd', markersize=12, marker

    # include legend for class mean vector
    if nclass == 3:
        l1 = plt.legend([m1,m2,m3],['Class 1 Mean', 'Class 2 Mean', 'Class 3 Mean'], l
    else:
        l1 = plt.legend([m1,m2], ['Class 1 Mean', 'Class 2 Mean'], loc=4)

    plt.gca().add_artist(l1)

    plt.show()

```

The classifier function which calculates the sample mean, error rate and success rate for both training and testing data

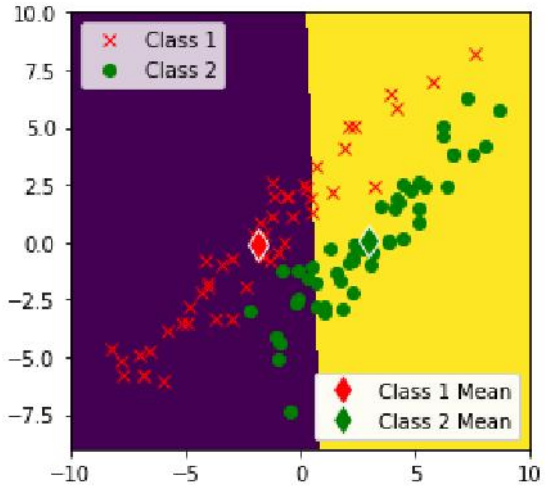
In [16]:

```
def classifier_synthetic1(sytrain_data, sytest_data):
    sy_train_len = len(sytrain_data);
    sy_test_len = len(sytest_data);
    label = sy_train_len-1;

    class1_f1 = [sytrain_data[i,0] for i in range (0,sy_train_len) if sytrain_data[i,2
    class2_f1 = [sytrain_data[i,0] for i in range (0,sy_train_len) if sytrain_data[i,2
    class1_f2 = [sytrain_data[i,1] for i in range (0,sy_train_len) if sytrain_data[i,2
    class2_f2 = [sytrain_data[i,1] for i in range (0,sy_train_len) if sytrain_data[i,2

    class1_f1_mean = np.mean(class1_f1)
    class2_f1_mean = np.mean(class2_f1)
    class1_f2_mean = np.mean(class1_f2)
    class2_f2_mean = np.mean(class2_f2)


    #Calculating the sample mean for the training data set
    sample_mean = np.array([[class1_f1_mean,class1_f2_mean],
                        [class2_f1_mean,class2_f2_mean]])

    euc_array =[];

    #Calculating the Euclidian distance of each data point from the respective class me

    for i in range (0,sy_train_len):
        x1 = np.array([sytrain_data[i,0]-sample_mean[0,0]]);
        y1 = np.array([sytrain_data[i,1]-sample_mean[0,1]]);
        x2 = np.array([sytrain_data[i,0]-sample_mean[1,0]]);
        y2 = np.array([sytrain_data[i,1]-sample_mean[1,1]]);

        euc_dist_1 = np.sqrt((x1**2)+(y1**2));
        euc_dist_2 = np.sqrt((x2**2)+(y2**2));



    #Calculating the minimum Euclidean distance and assigning it to the class with min
        if(euc_dist_1>euc_dist_2):
            euc_array.append(2);
        else:
            euc_array.append(1);

        euc_dist_1 = 0;
        euc_dist_2 = 0;

    #Calculating the Error rates and Success rates for the training data set
    success_cnt = 0;
    error_cnt = 0;
    for i in range(0,sy_train_len):
        if(euc_array[i] == sytrain_data[i,2]):
            success_cnt = success_cnt+1;
        else:
            error_cnt = error_cnt+1;

    error_rate = (error_cnt/sy_train_len)*100;
    success_rate = (success_cnt/sy_train_len)*100;

     #Calculating the Euclidian distance of each data point from the respective class m
    euc_array_test =[];
    for i in range (0,sy_test_len):
```

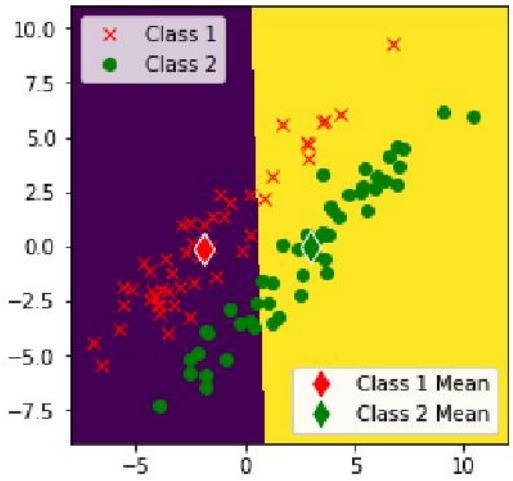
```
        x1_test = np.array([sytest_data[i,0]-sample_mean[0,0]]);
        y1_test = np.array([sytest_data[i,1]-sample_mean[0,1]]);
        x2_test = np.array([sytest_data[i,0]-sample_mean[1,0]]);
        y2_test = np.array([sytest_data[i,1]-sample_mean[1,1]]);

        euc_dist_1_test = np.sqrt((x1_test**2)+(y1_test**2));
        euc_dist_2_test = np.sqrt((x2_test**2)+(y2_test**2));



    #Calculating the minimum Euclidean distance and assigning it to the class with min
        if(euc_dist_1_test>euc_dist_2_test):
            euc_array_test.append(2);
        else:
            euc_array_test.append(1);

        euc_dist_1_test = 0;
        euc_dist_2_test = 0;

     #Calculating the Error rates and Success rates for the testing data set
    success_cnt_test = 0;
    error_cnt_test = 0;
    for i in range(0,sy_test_len):
        if(euc_array_test[i] == sytest_data[i,2]):
            success_cnt_test = success_cnt_test+1;
        else:
            error_cnt_test = error_cnt_test+1;

    error_rate_test = (error_cnt_test/sy_test_len)*100;
    success_rate_test = (success_cnt_test/sy_test_len)*100;

    return sample_mean,error_rate,success_rate,error_rate_test,success_rate_test;

```

In [17]:

```
sytrain_data = np.genfromtxt('synthetic1_train.csv',delimiter=',')
sytest_data = np.genfromtxt('synthetic1_test.csv', delimiter=',')
[sample_mean,error_rate,success_rate,error_rate_test,success_rate_test] = classifier_sy
plotDecBoundaries(sytrain_data,sytrain_data[:,2],sample_mean)
print("The error rate for Synthetic 1 Training data set is ",error_rate)
print("The success rate for Sythetic 1 Training data set is ",success_rate)
plotDecBoundaries(sytest_data,sytest_data[:,2],sample_mean)
print("The error rate for Synthetic 1 Testing data set is ",error_rate_test)
print("The success rate for Sythetic 1 Testing data set is ",success_rate_test)
```




```
The error rate for Synthetic 1 Training data set is  21.0
The success rate for Sythetic 1 Training data set is  79.0
```




```
The error rate for Synthetic 1 Testing data set is  24.0
The success rate for Sythetic 1 Testing data set is  76.0
```

In [ ]:

```

```

Problem a-Synthetic 2 data set

Using the first two features-f1 and f2

In [5]:

```
import numpy as np
from numpy import genfromtxt
import math as mt
import matplotlib.pyplot as plt
from scipy.spatial.distance import cdist
```

plotDecBoundaries function given

In [6]:

```
def plotDecBoundaries(training, label_train, sample_mean):

    #Plot the decision boundaries and data points for minimum distance to
    #class mean classifier
    #
    # training: traning data
    # label_train: class lables correspond to training data
    # sample_mean: mean vector for each class
    #
    # Total number of classes
    nclass =  max(np.unique(label_train))

    # Set the feature range for ploting
    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 = 0.005

    # 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

    # 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

    # distance measure evaluations for each (x,y) pair.
    dist_mat = cdist(xy, sample_mean)
    pred_label = np.argmin(dist_mat, axis=1)

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

    #show the image, give each coordinate a color according to its class label
    plt.imshow(decisionmap, extent=[xrange[0], xrange[1], yrange[0], yrange[1]], origi

    # 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')
    if nclass == 3:
        plt.plot(training[label_train == 3, 0],training[label_train == 3, 1], 'b*')

    # include legend for training data
    if nclass == 3:
        l = plt.legend(('Class 1', 'Class 2', 'Class 3'), loc=2)
    else:
        l = plt.legend(('Class 1', 'Class 2'), loc=2)
    plt.gca().add_artist(l)

    # plot the class mean vector.
    m1, = plt.plot(sample_mean[0,0], sample_mean[0,1], 'rd', markersize=12, markerface
    m2, = plt.plot(sample_mean[1,0], sample_mean[1,1], 'gd', markersize=12, markerface
```

```
    if nclass == 3:
        m3, = plt.plot(sample_mean[2,0], sample_mean[2,1], 'bd', markersize=12, marker
    # include legend for class mean vector
    if nclass == 3:
        l1 = plt.legend([m1,m2,m3],['Class 1 Mean', 'Class 2 Mean', 'Class 3 Mean'], l
    else:
        l1 = plt.legend([m1,m2], ['Class 1 Mean', 'Class 2 Mean'], loc=4)

    plt.gca().add_artist(l1)

    plt.show()

```

The classifier function which calculates the sample mean, error rate and success rate for both training and testing data

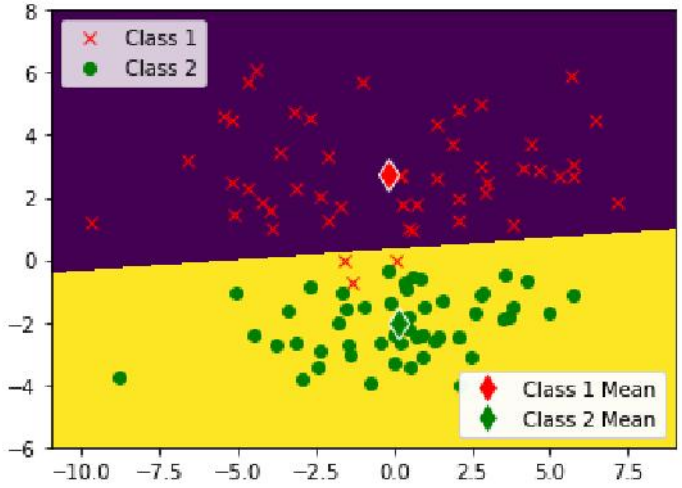
In [7]:

```
def classifier_synthetic1(sytrain_data, sytest_data):
    sy_train_len = len(sytrain_data);
    sy_test_len = len(sytest_data);
    label = sy_train_len-1;

    class1_f1 = [sytrain_data[i,0] for i in range (0,sy_train_len) if sytrain_data[i,2
    class2_f1 = [sytrain_data[i,0] for i in range (0,sy_train_len) if sytrain_data[i,2
    class1_f2 = [sytrain_data[i,1] for i in range (0,sy_train_len) if sytrain_data[i,2
    class2_f2 = [sytrain_data[i,1] for i in range (0,sy_train_len) if sytrain_data[i,2

    class1_f1_mean = np.mean(class1_f1)
    class2_f1_mean = np.mean(class2_f1)
    class1_f2_mean = np.mean(class1_f2)
    class2_f2_mean = np.mean(class2_f2)


    #Calculating the sample mean for the training data set
    sample_mean = np.array([[class1_f1_mean,class1_f2_mean],
                        [class2_f1_mean,class2_f2_mean]])

    euc_array =[];

    #Calculating the Euclidian distance of each data point from the respective class me

    for i in range (0,sy_train_len):
        x1 = np.array([sytrain_data[i,0]-sample_mean[0,0]]);
        y1 = np.array([sytrain_data[i,1]-sample_mean[0,1]]);
        x2 = np.array([sytrain_data[i,0]-sample_mean[1,0]]);
        y2 = np.array([sytrain_data[i,1]-sample_mean[1,1]]);

        euc_dist_1 = np.sqrt((x1**2)+(y1**2));
        euc_dist_2 = np.sqrt((x2**2)+(y2**2));



    #Calculating the minimum Euclidean distance and assigning it to the class with min
        if(euc_dist_1>euc_dist_2):
            euc_array.append(2);
        else:
            euc_array.append(1);

        euc_dist_1 = 0;
        euc_dist_2 = 0;

    #Calculating the Error rates and Success rates for the training data set
    success_cnt = 0;
    error_cnt = 0;
    for i in range(0,sy_train_len):
        if(euc_array[i] == sytrain_data[i,2]):
            success_cnt = success_cnt+1;
        else:
            error_cnt = error_cnt+1;

    error_rate = (error_cnt/sy_train_len)*100;
    success_rate = (success_cnt/sy_train_len)*100;

     #Calculating the Euclidian distance of each data point from the respective class m
    euc_array_test =[];
    for i in range (0,sy_test_len):
```

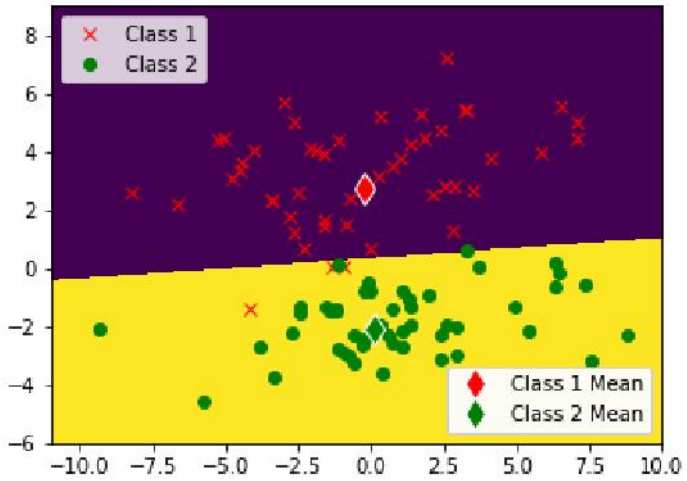
```
        x1_test = np.array([sytest_data[i,0]-sample_mean[0,0]]);
        y1_test = np.array([sytest_data[i,1]-sample_mean[0,1]]);
        x2_test = np.array([sytest_data[i,0]-sample_mean[1,0]]);
        y2_test = np.array([sytest_data[i,1]-sample_mean[1,1]]);

        euc_dist_1_test = np.sqrt((x1_test**2)+(y1_test**2));
        euc_dist_2_test = np.sqrt((x2_test**2)+(y2_test**2));



    #Calculating the minimum Euclidean distance and assigning it to the class with min
        if(euc_dist_1_test>euc_dist_2_test):
            euc_array_test.append(2);
        else:
            euc_array_test.append(1);

        euc_dist_1_test = 0;
        euc_dist_2_test = 0;

     #Calculating the Error rates and Success rates for the testing data set
    success_cnt_test = 0;
    error_cnt_test = 0;
    for i in range(0,sy_test_len):
        if(euc_array_test[i] == sytest_data[i,2]):
            success_cnt_test = success_cnt_test+1;
        else:
            error_cnt_test = error_cnt_test+1;

    error_rate_test = (error_cnt_test/sy_test_len)*100;
    success_rate_test = (success_cnt_test/sy_test_len)*100;

    return sample_mean,error_rate,success_rate,error_rate_test,success_rate_test;

```

In [8]:

```
sytrain_data = np.genfromtxt('synthetic2_train.csv',delimiter=',')
sytest_data = np.genfromtxt('synthetic2_test.csv', delimiter=',')
[sample_mean,error_rate,success_rate,error_rate_test,success_rate_test] = classifier_sy
plotDecBoundaries(sytrain_data,sytrain_data[:,2],sample_mean)
print("The error rate for Synthetic 2 Training data set is ",error_rate)
print("The success rate for Sythetic 2 Training data set is ",success_rate)
plotDecBoundaries(sytest_data,sytest_data[:,2],sample_mean)
print("The error rate for Synthetic 2 Testing data set is ",error_rate_test)
print("The success rate for Sythetic 2 Testing data set is ",success_rate_test)
```




```
The error rate for Synthetic 2 Training data set is  3.0
The success rate for Sythetic 2 Training data set is  97.0
```




```
The error rate for Synthetic 2 Testing data set is  4.0
The success rate for Sythetic 2 Testing data set is  96.0
```

In [ ]:

```
1
```

In [ ]:

```
1
```

Problem c-Wine data set

Using the first two features

In [18]:

```
import numpy as np
from numpy import genfromtxt
import math as mt
import matplotlib.pyplot as plt
from scipy.spatial.distance import cdist
```

plotDecBoundaries function given

In [19]:

```
def plotDecBoundaries(training, label_train, sample_mean):

    #Plot the decision boundaries and data points for minimum distance to
    #class mean classifier
    #
    # training: traning data
    # label_train: class lables correspond to training data
    # sample_mean: mean vector for each class
    #
    # Total number of classes
    nclass =  max(np.unique(label_train))

    # Set the feature range for ploting
    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 = 0.005

    # 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

    # 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

    # distance measure evaluations for each (x,y) pair.
    dist_mat = cdist(xy, sample_mean)
    pred_label = np.argmin(dist_mat, axis=1)

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

    #show the image, give each coordinate a color according to its class label
    plt.imshow(decisionmap, extent=[xrange[0], xrange[1], yrange[0], yrange[1]], origi

    # 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')
    if nclass == 3:
        plt.plot(training[label_train == 3, 0],training[label_train == 3, 1], 'b*')

    # include legend for training data
    if nclass == 3:
        l = plt.legend(('Class 1', 'Class 2', 'Class 3'), loc=2)
    else:
        l = plt.legend(('Class 1', 'Class 2'), loc=2)
    plt.gca().add_artist(l)

    # plot the class mean vector.
    m1, = plt.plot(sample_mean[0,0], sample_mean[0,1], 'rd', markersize=12, markerface
    m2, = plt.plot(sample_mean[1,0], sample_mean[1,1], 'gd', markersize=12, markerface
```

```
    if nclass == 3:
        m3, = plt.plot(sample_mean[2,0], sample_mean[2,1], 'bd', markersize=12, marker

    # include legend for class mean vector
    if nclass == 3:
        l1 = plt.legend([m1,m2,m3],['Class 1 Mean', 'Class 2 Mean', 'Class 3 Mean'], l
    else:
        l1 = plt.legend([m1,m2], ['Class 1 Mean', 'Class 2 Mean'], loc=4)

    plt.gca().add_artist(l1)

    plt.show()

```

The classifier function which calculates the sample mean, error rate and success rate for both training nad testing data

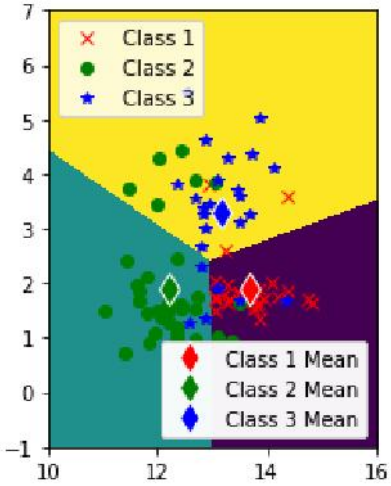
In [20]:

```
def classifier_wine(sytrain_data, sytest_data):
    sy_train_len = len(sytrain_data);
    sy_test_len = len(sytest_data);
    label = sy_train_len-1;

    class1_f1 = [sytrain_data[i,0] for i in range (0,sy_train_len) if sytrain_data[i,1
    class2_f1 = [sytrain_data[i,0] for i in range (0,sy_train_len) if sytrain_data[i,1
    class3_f1 = [sytrain_data[i,0] for i in range (0,sy_train_len) if sytrain_data[i,1
    class1_f2 = [sytrain_data[i,1] for i in range (0,sy_train_len) if sytrain_data[i,1
    class2_f2 = [sytrain_data[i,1] for i in range (0,sy_train_len) if sytrain_data[i,1
    class3_f2 = [sytrain_data[i,1] for i in range (0,sy_train_len) if sytrain_data[i,1

    class1_f1_mean = np.mean(class1_f1)
    class2_f1_mean = np.mean(class2_f1)
    class3_f1_mean = np.mean(class3_f1)
    class1_f2_mean = np.mean(class1_f2)
    class2_f2_mean = np.mean(class2_f2)
    class3_f2_mean = np.mean(class3_f2)

    #Calculating the sample mean for the training data set
    sample_mean = np.array([[class1_f1_mean,class1_f2_mean],
                        [class2_f1_mean,class2_f2_mean],
                        [class3_f1_mean,class3_f2_mean]])

    euc_array =[];

    #Calculating the Euclidian distance of each data point from the respective class m
    for i in range(0,sy_train_len):
        x1 = np.array([sytrain_data[i,0]-sample_mean[0,0]]);
        y1 = np.array([sytrain_data[i,1]-sample_mean[0,1]]);
        x2 = np.array([sytrain_data[i,0]-sample_mean[1,0]]);
        y2 = np.array([sytrain_data[i,1]-sample_mean[1,1]]);
        x3 = np.array([sytrain_data[i,0]-sample_mean[2,0]]);
        y3 = np.array([sytrain_data[i,1]-sample_mean[2,1]]);


        euc_dist_1 = np.sqrt((x1**2)+(y1**2));
        euc_dist_2 = np.sqrt((x2**2)+(y2**2));
        euc_dist_3 = np.sqrt((x3**2)+(y3**2));

    #Calculating the minimum Euclidean distance and assigning it to the class with min
        if((euc_dist_1<euc_dist_2) & (euc_dist_1<euc_dist_3) ):
            euc_array.append(1);
        elif((euc_dist_2<euc_dist_1) & (euc_dist_2<euc_dist_3) ):
            euc_array.append(2);
        elif((euc_dist_3<euc_dist_1) & (euc_dist_3<euc_dist_2) ):
            euc_array.append(3);

        euc_dist_1 = 0;
        euc_dist_2 = 0;
        euc_dist_3 = 0;

    #Calculating the Error rates and Success rates for the training data set
    success_cnt = 0;
    error_cnt = 0;
    for i in range(0,sy_train_len):
        if(euc_array[i] == sytrain_data[i,13]):
            success_cnt = success_cnt+1;
        else:
```

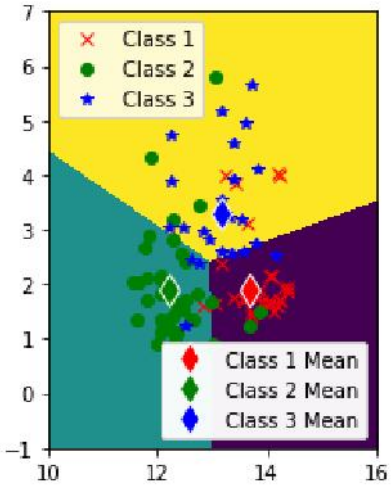
```
            error_cnt = error_cnt+1;

    error_rate = (error_cnt/sy_train_len)*100;
    success_rate = (success_cnt/sy_train_len)*100;

     #Calculating the Euclidian distance of each data point from the respective class
    euc_array_test =[];
    for i in range (0,sy_train_len):
        x1_test = np.array([sytest_data[i,0]-sample_mean[0,0]]);
        y1_test = np.array([sytest_data[i,1]-sample_mean[0,1]]);
        x2_test = np.array([sytest_data[i,0]-sample_mean[1,0]]);
        y2_test = np.array([sytest_data[i,1]-sample_mean[1,1]]);
        x3_test = np.array([sytest_data[i,0]-sample_mean[2,0]]);
        y3_test = np.array([sytest_data[i,1]-sample_mean[2,1]]);

        euc_dist_1_test = np.sqrt((x1_test**2)+(y1_test**2));
        euc_dist_2_test = np.sqrt((x2_test**2)+(y2_test**2));
        euc_dist_3_test = np.sqrt((x3_test**2)+(y3_test**2));

    #Calculating the minimum Euclidean distance and assigning it to the class with min
        if(euc_dist_1_test == min(euc_dist_1_test,euc_dist_2_test,euc_dist_3_test) ):
            euc_array_test.append(1);
        elif(euc_dist_2_test == min(euc_dist_1_test,euc_dist_2_test,euc_dist_3_test) )
            euc_array_test.append(2);
        elif(euc_dist_3_test == min(euc_dist_1_test,euc_dist_2_test,euc_dist_3_test) )
            euc_array_test.append(3);

        euc_dist_1_test = 0;
        euc_dist_2_test = 0;
        euc_dist_3_test = 0;

     #Calculating the Error rates and Success rates for the testing data set
        success_cnt_test = 0;
        error_cnt_test = 0;
    for i in range(0,sy_test_len):
        if(euc_array_test[i] == sytest_data[i,13]):
            success_cnt_test = success_cnt_test+1;
        else:
            error_cnt_test = error_cnt_test+1;

    error_rate_test = (error_cnt_test/sy_test_len)*100;
    success_rate_test = (success_cnt_test/sy_test_len)*100;

    return sample_mean,error_rate,success_rate,error_rate_test,success_rate_test;

```

In [21]:

```
sytrain_data = np.genfromtxt('wine_train.csv',delimiter=',')
sytest_data = np.genfromtxt('wine_test.csv', delimiter=',')
[sample_mean,error_rate,success_rate,error_rate_test,success_rate_test] = classifier_w
plotDecBoundaries(sytrain_data,sytrain_data[:,13],sample_mean)
print("The error rate for Wine Training data set is ",error_rate)
print("The success rate for Wine Training data set is ",success_rate)
plotDecBoundaries(sytest_data,sytest_data[:,13],sample_mean)
print("The error rate for Wine Testing data set is ",error_rate_test)
print("The success rate for Wine Testing data set is ",success_rate_test)
```




```
The error rate for Wine Training data set is  20.224719101123593
The success rate for Wine Training data set is  79.7752808988764
```




```
The error rate for Wine Testing data set is  22.47191011235955
The success rate for Wine Testing data set is  77.52808988764045
```

In [ ]:

```

```

Problem e-Wine data set

Using two features for all combination from f1 to f13

In [23]:

```
import numpy as np
from numpy import genfromtxt
import math as mt
import matplotlib.pyplot as plt
from scipy.spatial.distance import cdist
```

plotDecBoundaries function given

In [24]:

```
def plotDecBoundaries(training, label_train, sample_mean):

    #Plot the decision boundaries and data points for minimum distance to
    #class mean classifier
    #
    # training: traning data
    # label_train: class lables correspond to training data
    # sample_mean: mean vector for each class
    #
    # Total number of classes
    nclass =  max(np.unique(label_train))

    # Set the feature range for ploting
    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 = 0.005

    # 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

    # 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

    # distance measure evaluations for each (x,y) pair.
    dist_mat = cdist(xy, sample_mean)
    pred_label = np.argmin(dist_mat, axis=1)

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

    #show the image, give each coordinate a color according to its class label
    plt.imshow(decisionmap, extent=[xrange[0], xrange[1], yrange[0], yrange[1]], origi

    # 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')
    if nclass == 3:
        plt.plot(training[label_train == 3, 0],training[label_train == 3, 1], 'b*')

    # include legend for training data
    if nclass == 3:
        l = plt.legend(('Class 1', 'Class 2', 'Class 3'), loc=2)
    else:
        l = plt.legend(('Class 1', 'Class 2'), loc=2)
    plt.gca().add_artist(l)

    # plot the class mean vector.
    m1, = plt.plot(sample_mean[0,0], sample_mean[0,1], 'rd', markersize=12, markerface
    m2, = plt.plot(sample_mean[1,0], sample_mean[1,1], 'gd', markersize=12, markerface
```

```
    if nclass == 3:
        m3, = plt.plot(sample_mean[2,0], sample_mean[2,1], 'bd', markersize=12, marker

    # include legend for class mean vector
    if nclass == 3:
        l1 = plt.legend([m1,m2,m3],['Class 1 Mean', 'Class 2 Mean', 'Class 3 Mean'], l
    else:
        l1 = plt.legend([m1,m2], ['Class 1 Mean', 'Class 2 Mean'], loc=4)

    plt.gca().add_artist(l1)

    plt.show()

```

The classifier function which calculates the sample mean, error rate and success rate for both training and testing data

In [25]:

```
def classifier_wine(sytrain_data, sytest_data):
    sy_train_len = len(sytrain_data);
    sy_test_len = len(sytest_data);
    label = sy_train_len-1;

    class1_f1 = [sytrain_data[i,0] for i in range (0,sy_train_len) if sytrain_data[i,2
    class2_f1 = [sytrain_data[i,0] for i in range (0,sy_train_len) if sytrain_data[i,2
    class3_f1 = [sytrain_data[i,0] for i in range (0,sy_train_len) if sytrain_data[i,2
    class1_f2 = [sytrain_data[i,1] for i in range (0,sy_train_len) if sytrain_data[i,2
    class2_f2 = [sytrain_data[i,1] for i in range (0,sy_train_len) if sytrain_data[i,2
    class3_f2 = [sytrain_data[i,1] for i in range (0,sy_train_len) if sytrain_data[i,2

    class1_f1_mean = np.mean(class1_f1)
    class2_f1_mean = np.mean(class2_f1)
    class3_f1_mean = np.mean(class3_f1)
    class1_f2_mean = np.mean(class1_f2)
    class2_f2_mean = np.mean(class2_f2)
    class3_f2_mean = np.mean(class3_f2)

    #Calculating the sample mean for the training data set
    sample_mean = np.array([[class1_f1_mean,class1_f2_mean],
                        [class2_f1_mean,class2_f2_mean],
                        [class3_f1_mean,class3_f2_mean]])

    euc_array =[];

    #Calculating the Euclidian distance of each data point from the respective class m
    for i in range(0,sy_train_len):
        x1 = np.array([sytrain_data[i,0]-sample_mean[0,0]]);
        y1 = np.array([sytrain_data[i,1]-sample_mean[0,1]]);
        x2 = np.array([sytrain_data[i,0]-sample_mean[1,0]]);
        y2 = np.array([sytrain_data[i,1]-sample_mean[1,1]]);
        x3 = np.array([sytrain_data[i,0]-sample_mean[2,0]]);
        y3 = np.array([sytrain_data[i,1]-sample_mean[2,1]]);


        euc_dist_1 = np.sqrt((x1**2)+(y1**2));
        euc_dist_2 = np.sqrt((x2**2)+(y2**2));
        euc_dist_3 = np.sqrt((x3**2)+(y3**2));

    #Calculating the minimum Euclidean distance and assigning it to the class with min
        if((euc_dist_1<euc_dist_2) & (euc_dist_1<euc_dist_3) ):
            euc_array.append(1);
        elif((euc_dist_2<euc_dist_1) & (euc_dist_2<euc_dist_3) ):
            euc_array.append(2);
        elif((euc_dist_3<euc_dist_1) & (euc_dist_3<euc_dist_2) ):
            euc_array.append(3);

        euc_dist_1 = 0;
        euc_dist_2 = 0;
        euc_dist_3 = 0;

    #Calculating the Error rates and Success rates for the training data set
    success_cnt = 0;
    error_cnt = 0;
    for i in range(0,sy_train_len):
        if(euc_array[i] == sytrain_data[i,2]):
            success_cnt = success_cnt+1;
        else:
```

```
            error_cnt = error_cnt+1;

    error_rate = (error_cnt/sy_train_len)*100;
    success_rate = (success_cnt/sy_train_len)*100;

     #Calculating the Euclidian distance of each data point from the respective class
    euc_array_test =[];
    for i in range (0,sy_train_len):
        x1_test = np.array([sytest_data[i,0]-sample_mean[0,0]]);
        y1_test = np.array([sytest_data[i,1]-sample_mean[0,1]]);
        x2_test = np.array([sytest_data[i,0]-sample_mean[1,0]]);
        y2_test = np.array([sytest_data[i,1]-sample_mean[1,1]]);
        x3_test = np.array([sytest_data[i,0]-sample_mean[2,0]]);
        y3_test = np.array([sytest_data[i,1]-sample_mean[2,1]]);

        euc_dist_1_test = np.sqrt((x1_test**2)+(y1_test**2));
        euc_dist_2_test = np.sqrt((x2_test**2)+(y2_test**2));
        euc_dist_3_test = np.sqrt((x3_test**2)+(y3_test**2));

    #Calculating the minimum Euclidean distance and assigning it to the class with min
        if(euc_dist_1_test == min(euc_dist_1_test,euc_dist_2_test,euc_dist_3_test) ):
            euc_array_test.append(1);
        elif(euc_dist_2_test == min(euc_dist_1_test,euc_dist_2_test,euc_dist_3_test) )
            euc_array_test.append(2);
        elif(euc_dist_3_test == min(euc_dist_1_test,euc_dist_2_test,euc_dist_3_test) )
            euc_array_test.append(3);

        euc_dist_1_test = 0;
        euc_dist_2_test = 0;
        euc_dist_3_test = 0;

     #Calculating the Error rates and Success rates for the testing data set
        success_cnt_test = 0;
        error_cnt_test = 0;
    for i in range(0,sy_test_len):
        if(euc_array_test[i] == sytest_data[i,2]):
            success_cnt_test = success_cnt_test+1;
        else:
            error_cnt_test = error_cnt_test+1;

    error_rate_test = (error_cnt_test/sy_test_len)*100;
    success_rate_test = (success_cnt_test/sy_test_len)*100;

    return sample_mean,error_rate,success_rate,error_rate_test,success_rate_test;

```

In [26]:

```
sytrain_data = np.genfromtxt('wine_train.csv',delimiter=',')
sytest_data = np.genfromtxt('wine_test.csv', delimiter=',')
for i in range(0,13):
    for j in range(i+1,13):
        sytrain_data_to_col1 = sytrain_data[:,i];
        sytrain_data_to_col2 = sytrain_data[:,j];
        sytrain_data_to_col3 = sytrain_data[:,13];
        sytrain_data_to = np.column_stack((sytrain_data_to_col1,sytrain_data_to_col2,sy
        sytest_data_to_col1 = sytest_data[:,i];
        sytest_data_to_col2 = sytest_data[:,j];
        sytest_data_to_col3 = sytest_data[:,13];
        sytest_data_to = np.column_stack((sytest_data_to_col1,sytest_data_to_col2,syte
        [sample_mean,error_rate,success_rate,error_rate_test,success_rate_test] = clas
        #plotDecBoundaries(sytrain_data,sytrain_data[:,13],sample_mean)
        print("The error rate of f1 = ",i+1," f2 =",j+1," of Wine Training data set is
        #print("The success rate of f1 = ",i+1," f2 =",j+1," of Wine Training data set
        #plotDecBoundaries(sytest_data,sytest_data[:,13],sample_mean)
        print("The error rate of f1 = ",i+1," f2 =",j+1," of Wine Testing data set is
        #print("The success rate of f1 = ",i+1," f2 =",j+1," of Wine Testing data set
```

```
The error rate of f1 =  1  f2 = 2  of Wine Training data set is  20.224719
101123593
The error rate of f1 =  1  f2 = 2  of Wine Testing data set is  22.4719101
1235955
The error rate of f1 =  1  f2 = 3  of Wine Training data set is  31.460674
15730337
The error rate of f1 =  1  f2 = 3  of Wine Testing data set is  28.0898876
4044944
The error rate of f1 =  1  f2 = 4  of Wine Training data set is  44.943820
2247191
The error rate of f1 =  1  f2 = 4  of Wine Testing data set is  40.4494382
0224719
The error rate of f1 =  1  f2 = 5  of Wine Training data set is  56.179775
28089888
The error rate of f1 =  1  f2 = 5  of Wine Testing data set is  44.9438202
247191
The error rate of f1 =  1  f2 = 6  of Wine Training data set is  14.606741
573033707
The error rate of f1 =  1  f2 = 6  of Wine Testing data set is  15.7303370
```

In [ ]:

```
1
```

Problem d-Wine data set

Using the best two features-f1 and f12

In [9]:

```
import numpy as np
from numpy import genfromtxt
import math as mt
import matplotlib.pyplot as plt
from scipy.spatial.distance import cdist
```

plotDecBoundaries function given

In [10]:

```
def plotDecBoundaries(training, label_train, sample_mean):

    #Plot the decision boundaries and data points for minimum distance to
    #class mean classifier
    #
    # training: traning data
    # label_train: class lables correspond to training data
    # sample_mean: mean vector for each class
    #
    # Total number of classes
    nclass =  max(np.unique(label_train))

    # Set the feature range for ploting
    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 = 0.005

    # 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

    # 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

    # distance measure evaluations for each (x,y) pair.
    dist_mat = cdist(xy, sample_mean)
    pred_label = np.argmin(dist_mat, axis=1)

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

    #show the image, give each coordinate a color according to its class label
    plt.imshow(decisionmap, extent=[xrange[0], xrange[1], yrange[0], yrange[1]], origi

    # 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')
    if nclass == 3:
        plt.plot(training[label_train == 3, 0],training[label_train == 3, 1], 'b*')

    # include legend for training data
    if nclass == 3:
        l = plt.legend(('Class 1', 'Class 2', 'Class 3'), loc=2)
    else:
        l = plt.legend(('Class 1', 'Class 2'), loc=2)
    plt.gca().add_artist(l)

    # plot the class mean vector.
    m1, = plt.plot(sample_mean[0,0], sample_mean[0,1], 'rd', markersize=12, markerface
    m2, = plt.plot(sample_mean[1,0], sample_mean[1,1], 'gd', markersize=12, markerface
```

```
    if nclass == 3:
        m3, = plt.plot(sample_mean[2,0], sample_mean[2,1], 'bd', markersize=12, marker

    # include legend for class mean vector
    if nclass == 3:
        l1 = plt.legend([m1,m2,m3],['Class 1 Mean', 'Class 2 Mean', 'Class 3 Mean'], l
    else:
        l1 = plt.legend([m1,m2], ['Class 1 Mean', 'Class 2 Mean'], loc=4)

    plt.gca().add_artist(l1)

    plt.show()

```

The classifier function which calculates the sample mean, error rate and success rate for both training and testing data

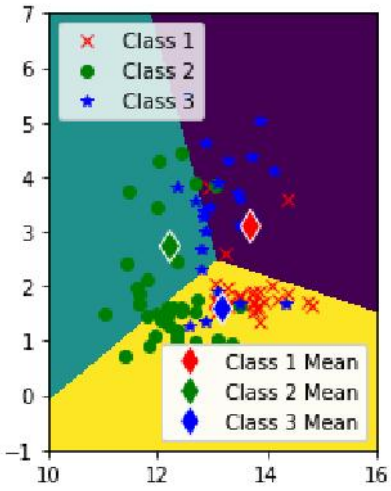
In [11]:

```
def classifier_wine(sytrain_data, sytest_data):
    sy_train_len = len(sytrain_data);
    sy_test_len = len(sytest_data);
    label = sy_train_len-1;

    class1_f1 = [sytrain_data[i,0] for i in range (0,sy_train_len) if sytrain_data[i,1
    class2_f1 = [sytrain_data[i,0] for i in range (0,sy_train_len) if sytrain_data[i,1
    class3_f1 = [sytrain_data[i,0] for i in range (0,sy_train_len) if sytrain_data[i,1
    class1_f2 = [sytrain_data[i,11] for i in range (0,sy_train_len) if sytrain_data[i,
    class2_f2 = [sytrain_data[i,11] for i in range (0,sy_train_len) if sytrain_data[i,
    class3_f2 = [sytrain_data[i,11] for i in range (0,sy_train_len) if sytrain_data[i,

    class1_f1_mean = np.mean(class1_f1)
    class2_f1_mean = np.mean(class2_f1)
    class3_f1_mean = np.mean(class3_f1)
    class1_f2_mean = np.mean(class1_f2)
    class2_f2_mean = np.mean(class2_f2)
    class3_f2_mean = np.mean(class3_f2)

    #Calculating the sample mean for the training data set
    sample_mean = np.array([[class1_f1_mean,class1_f2_mean],
                        [class2_f1_mean,class2_f2_mean],
                        [class3_f1_mean,class3_f2_mean]])

    euc_array =[];

    #Calculating the Euclidian distance of each data point from the respective class m
    for i in range(0,sy_train_len):
        x1 = np.array([sytrain_data[i,0]-sample_mean[0,0]]);
        y1 = np.array([sytrain_data[i,11]-sample_mean[0,1]]);
        x2 = np.array([sytrain_data[i,0]-sample_mean[1,0]]);
        y2 = np.array([sytrain_data[i,11]-sample_mean[1,1]]);
        x3 = np.array([sytrain_data[i,0]-sample_mean[2,0]]);
        y3 = np.array([sytrain_data[i,11]-sample_mean[2,1]]);


        euc_dist_1 = np.sqrt((x1**2)+(y1**2));
        euc_dist_2 = np.sqrt((x2**2)+(y2**2));
        euc_dist_3 = np.sqrt((x3**2)+(y3**2));

    #Calculating the minimum Euclidean distance and assigning it to the class with min
        if((euc_dist_1<euc_dist_2) & (euc_dist_1<euc_dist_3) ):
            euc_array.append(1);
        elif((euc_dist_2<euc_dist_1) & (euc_dist_2<euc_dist_3) ):
            euc_array.append(2);
        elif((euc_dist_3<euc_dist_1) & (euc_dist_3<euc_dist_2) ):
            euc_array.append(3);

        euc_dist_1 = 0;
        euc_dist_2 = 0;
        euc_dist_3 = 0;

    #Calculating the Error rates and Success rates for the training data set
    success_cnt = 0;
    error_cnt = 0;
    for i in range(0,sy_train_len):
        if(euc_array[i] == sytrain_data[i,13]):
            success_cnt = success_cnt+1;
        else:
```

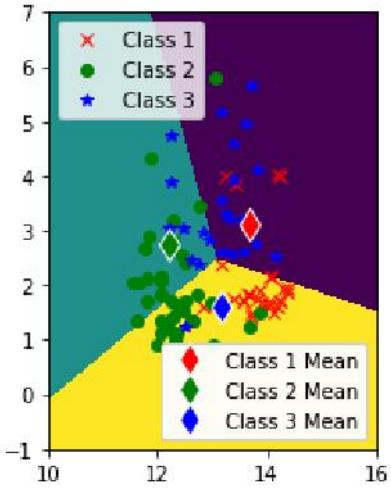
```
            error_cnt = error_cnt+1;

    error_rate = (error_cnt/sy_train_len)*100;
    success_rate = (success_cnt/sy_train_len)*100;

     #Calculating the Euclidian distance of each data point from the respective class
    euc_array_test =[];
    for i in range (0,sy_train_len):
        x1_test = np.array([sytest_data[i,0]-sample_mean[0,0]]);
        y1_test = np.array([sytest_data[i,11]-sample_mean[0,1]]);
        x2_test = np.array([sytest_data[i,0]-sample_mean[1,0]]);
        y2_test = np.array([sytest_data[i,11]-sample_mean[1,1]]);
        x3_test = np.array([sytest_data[i,0]-sample_mean[2,0]]);
        y3_test = np.array([sytest_data[i,11]-sample_mean[2,1]]);

        euc_dist_1_test = np.sqrt((x1_test**2)+(y1_test**2));
        euc_dist_2_test = np.sqrt((x2_test**2)+(y2_test**2));
        euc_dist_3_test = np.sqrt((x3_test**2)+(y3_test**2));

    #Calculating the minimum Euclidean distance and assigning it to the class with min
        if(euc_dist_1_test == min(euc_dist_1_test,euc_dist_2_test,euc_dist_3_test) ):
            euc_array_test.append(1);
        elif(euc_dist_2_test == min(euc_dist_1_test,euc_dist_2_test,euc_dist_3_test) )
            euc_array_test.append(2);
        elif(euc_dist_3_test == min(euc_dist_1_test,euc_dist_2_test,euc_dist_3_test) )
            euc_array_test.append(3);

        euc_dist_1_test = 0;
        euc_dist_2_test = 0;
        euc_dist_3_test = 0;

     #Calculating the Error rates and Success rates for the testing data set
        success_cnt_test = 0;
        error_cnt_test = 0;
    for i in range(0,sy_test_len):
        if(euc_array_test[i] == sytest_data[i,13]):
            success_cnt_test = success_cnt_test+1;
        else:
            error_cnt_test = error_cnt_test+1;

    error_rate_test = (error_cnt_test/sy_test_len)*100;
    success_rate_test = (success_cnt_test/sy_test_len)*100;

    return sample_mean,error_rate,success_rate,error_rate_test,success_rate_test;

```

In [12]:

```
sytrain_data = np.genfromtxt('wine_train.csv',delimiter=',')
sytest_data = np.genfromtxt('wine_test.csv', delimiter=',')
[sample_mean,error_rate,success_rate,error_rate_test,success_rate_test] = classifier_w
plotDecBoundaries(sytrain_data,sytrain_data[:,13],sample_mean)
print("The error rate for Wine Training data set is ",error_rate)
print("The success rate for Wine Training data set is ",success_rate)
plotDecBoundaries(sytest_data,sytest_data[:,13],sample_mean)
print("The error rate for Wine Testing data set is ",error_rate_test)
print("The success rate for Wine Testing data set is ",success_rate_test)
```




```
The error rate for Wine Training data set is  7.865168539325842
The success rate for Wine Training data set is  92.13483146067416
```




```
The error rate for Wine Testing data set is  12.359550561797752
The success rate for Wine Testing data set is  87.64044943820225
```

In [ ]:

```

```