```
# Problem b-Wine data set
```

Using two features for all combination from f1 to f13

In [126]:

```
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 [189]:

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

    #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))
    nclass = 2
    # 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')
    plt.plot(training[label_train == 3, 0],training[label_train == 3, 1], 'b*')
    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','Class 3'), loc=2)
    plt.gca().add_artist(l)

    # plot the class mean vector.
    if(classno ==1):
```

```
        m1, = plt.plot(sample_mean[0,0], sample_mean[0,1], 'rd', markersize=12, marker
        m2, = plt.plot(sample_mean[1,0], sample_mean[1,1], 'yd', markersize=12, marker
    elif(classno ==2):
        m1, = plt.plot(sample_mean[0,0], sample_mean[0,1], 'cd', markersize=12, marker
        m2, = plt.plot(sample_mean[1,0], sample_mean[1,1], 'gd', markersize=12, marker
    elif(classno ==3):
        m1, = plt.plot(sample_mean[0,0], sample_mean[0,1], 'md', markersize=12, marker
        m2, = plt.plot(sample_mean[1,0], sample_mean[1,1], 'bd', markersize=12, marker

    # include legend for class mean vector
    if(classno ==1):

        l1 = plt.legend([m1,m2], ['Class 1 Mean', 'Class rest Mean'], loc=4)
    elif(classno ==2):

        l1 = plt.legend([m1,m2], ['Class rest Mean', 'Class 2 Mean'], loc=4)
    elif(classno == 3):
        l1 = plt.legend([m1,m2], ['Class rest Mean', 'Class 3 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 [190]:

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

    #Class 1 vs the rest
    class1_f1 = [sytrain_data[i,0] for i in range (0,sy_train_len) if sytrain_data[i,2]=
    classnot1_f1 = [sytrain_data[i,0] for i in range (0,sy_train_len) if (sytrain_data[i
    class1_f2 = [sytrain_data[i,1] for i in range (0,sy_train_len) if sytrain_data[i,2]=
    classnot1_f2 = [sytrain_data[i,1] for i in range (0,sy_train_len) if (sytrain_data[i

    class1_f1_mean = np.mean(class1_f1)
    classnot1_f1_mean = np.mean(classnot1_f1)
    class1_f2_mean = np.mean(class1_f2)
    classnot1_f2_mean = np.mean(classnot1_f2)

    #Class 2 vs the rest
    class2_f1 = [sytrain_data[i,0] for i in range (0,sy_train_len) if sytrain_data[i,2]=
    classnot2_f1 = [sytrain_data[i,0] for i in range (0,sy_train_len) if (sytrain_data[i
    class2_f2 = [sytrain_data[i,1] for i in range (0,sy_train_len) if sytrain_data[i,2]=
    classnot2_f2 = [sytrain_data[i,1] for i in range (0,sy_train_len) if (sytrain_data[i

    class2_f1_mean = np.mean(class2_f1)
    classnot2_f1_mean = np.mean(classnot2_f1)
    class2_f2_mean = np.mean(class2_f2)
    classnot2_f2_mean = np.mean(classnot2_f2)

     #Class 3 vs the rest
    class3_f1 = [sytrain_data[i,0] for i in range (0,sy_train_len) if sytrain_data[i,2]=
    classnot3_f1 = [sytrain_data[i,0] for i in range (0,sy_train_len) if (sytrain_data[i
    class3_f2 = [sytrain_data[i,1] for i in range (0,sy_train_len) if sytrain_data[i,2]=
    classnot3_f2 = [sytrain_data[i,1] for i in range (0,sy_train_len) if (sytrain_data[i

    class3_f1_mean = np.mean(class3_f1)
    classnot3_f1_mean = np.mean(classnot3_f1)
    class3_f2_mean = np.mean(class3_f2)
    classnot3_f2_mean = np.mean(classnot3_f2)


    #Calculating the sample mean for the training data set
    sample_mean = np.array([[class1_f1_mean,class1_f2_mean],
                            [classnot1_f1_mean,classnot1_f2_mean],
                            [class2_f1_mean,class2_f2_mean],
                            [classnot2_f1_mean,classnot2_f2_mean],
                            [class3_f1_mean,class3_f2_mean],
                            [classnot3_f1_mean,classnot3_f2_mean],])

    euc_array =[];

    #Calculating the Euclidian distance of each data point from the respective class mea
    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]]);
        xnot1 = np.array([sytrain_data[i,0]-sample_mean[1,0]]);
        ynot1 = np.array([sytrain_data[i,1]-sample_mean[1,1]]);
        x2 = np.array([sytrain_data[i,0]-sample_mean[2,0]]);
        y2 = np.array([sytrain_data[i,1]-sample_mean[2,1]]);
        xnot2 = np.array([sytrain_data[i,0]-sample_mean[3,0]]);
        ynot2 = np.array([sytrain_data[i,1]-sample_mean[3,1]]);
```

```
60        x3 = np.array([sytrain_data[i,0]-sample_mean[4,0]]);
61        y3 = np.array([sytrain_data[i,1]-sample_mean[4,1]]);
62        xnot3 = np.array([sytrain_data[i,0]-sample_mean[5,0]]);
63        ynot3 = np.array([sytrain_data[i,1]-sample_mean[5,1]]);
64
65        euc_dist_1 = np.sqrt((x1**2)+(y1**2));
66        euc_dist_not1 = np.sqrt((xnot1**2)+(ynot1**2));
67        euc_dist_2 = np.sqrt((x2**2)+(y2**2));
68        euc_dist_not2 = np.sqrt((xnot2**2)+(ynot2**2));
69        euc_dist_3 = np.sqrt((x3**2)+(y3**2));
70        euc_dist_not3 = np.sqrt((xnot3**2)+(ynot3**2));
71
72
73    #Calculating the minimum Euclidean distance and assigning it to the class with minim
74        if((euc_dist_1<euc_dist_not1) and (euc_dist_2>euc_dist_not2) and (euc_dist_3>euc
75            euc_array.append(1);
76        elif((euc_dist_2<euc_dist_not2 ) and (euc_dist_1>euc_dist_not1) and (euc_dist_3>
77            euc_array.append(2);
78        elif((euc_dist_3<euc_dist_not3 ) and (euc_dist_1>euc_dist_not1) and (euc_dist_2>
79            euc_array.append(3);
80        else:
81            euc_array.append(0);
82
83        '''print(euc_dist_1)
84        print(euc_dist_2)
85        print(euc_dist_3)
86        print(euc_dist_not1)
87        print(euc_dist_not2)
88        print(euc_dist_not3)'''
89
90
91
92        euc_dist_1 = 0;
93        euc_dist_2 = 0;
94        euc_dist_3 = 0;
95        euc_dist_not1 = 0;
96        euc_dist_not2 = 0;
97        euc_dist_not3 = 0;
98    #print(euc_array)
99    #Calculating the Error rates and Success rates for the training data set
100   success_cnt = 0;
101   error_cnt = 0;
102   for i in range(0,len(euc_array)):
103       if(euc_array[i] == sytrain_data[i,2]):
104           success_cnt = success_cnt+1;
105       else:
106           error_cnt = error_cnt+1;
107
108   error_rate = (error_cnt/sy_train_len)*100;
109   success_rate = (100 - error_rate);
110
111    #Calculating the Euclidian distance of each data point from the respective class me
112   euc_arraytest =[];
113   for i in range(0,sy_test_len):
114       x1test = np.array([sytest_data[i,0]-sample_mean[0,0]]);
115       y1test = np.array([sytest_data[i,1]-sample_mean[0,1]]);
116       xnot1test = np.array([sytest_data[i,0]-sample_mean[1,0]]);
117       ynot1test = np.array([sytest_data[i,1]-sample_mean[1,1]]);
118       x2test = np.array([sytest_data[i,0]-sample_mean[2,0]]);
119       y2test= np.array([sytest_data[i,1]-sample_mean[2,1]]);
120       xnot2test = np.array([sytest_data[i,0]-sample_mean[3,0]]);
```

```
121         ynot2test = np.array([sytest_data[i,1]-sample_mean[3,1]]);
122         x3test = np.array([sytest_data[i,0]-sample_mean[4,0]]);
123         y3test = np.array([sytest_data[i,1]-sample_mean[4,1]]);
124         xnot3test = np.array([sytest_data[i,0]-sample_mean[5,0]]);
125         ynot3test = np.array([sytest_data[i,1]-sample_mean[5,1]]);
126 
127         euc_dist_1test = np.sqrt((x1test**2)+(y1test**2));
128         euc_dist_not1test = np.sqrt((xnot1test**2)+(ynot1test**2));
129         euc_dist_2test = np.sqrt((x2test**2)+(y2test**2));
130         euc_dist_not2test = np.sqrt((xnot2test**2)+(ynot2test**2));
131         euc_dist_3test = np.sqrt((x3test**2)+(y3test**2));
132         euc_dist_not3test = np.sqrt((xnot3test**2)+(ynot3test**2));
133 
134     #Calculating the minimum Euclidean distance and assigning it to the class with minim
135         if((euc_dist_1test<euc_dist_not1test) and (euc_dist_2test>euc_dist_not2test) and
136             euc_arraytest.append(1);
137         elif((euc_dist_2test<euc_dist_not2test ) and (euc_dist_1test>euc_dist_not1test)
138             euc_arraytest.append(2);
139         elif((euc_dist_3test<euc_dist_not3test ) and (euc_dist_1test>euc_dist_not1test)
140             euc_arraytest.append(3);
141         else:
142             euc_arraytest.append(0);
143 
144         euc_dist_1test = 0;
145         euc_dist_2test = 0;
146         euc_dist_3test = 0;
147         euc_dist_not1test = 0;
148         euc_dist_not2test = 0;
149         euc_dist_not3test = 0;
150 
151 
152 
153      #Calculating the Error rates and Success rates for the testing data set
154         success_cnt_test = 0;
155         error_cnt_test = 0;
156     for i in range(0,len(euc_arraytest)):
157         if(euc_arraytest[i] == sytest_data[i,2]):
158             success_cnt_test = success_cnt_test+1;
159         else:
160             error_cnt_test = error_cnt_test+1;
161 
162     error_rate_test = (error_cnt_test/sy_test_len)*100;
163     success_rate_test = (100-error_rate_test);
164 
165     return sample_mean,error_rate,success_rate,error_rate_test,success_rate_test;
166 
```

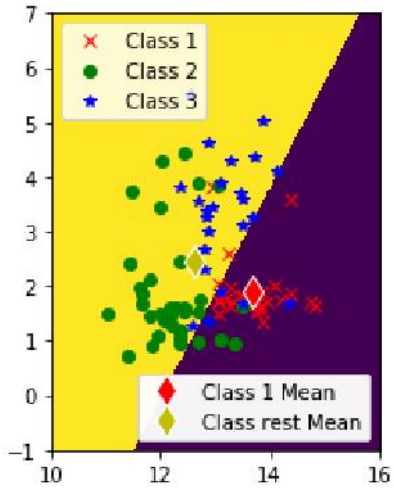
In [195]:

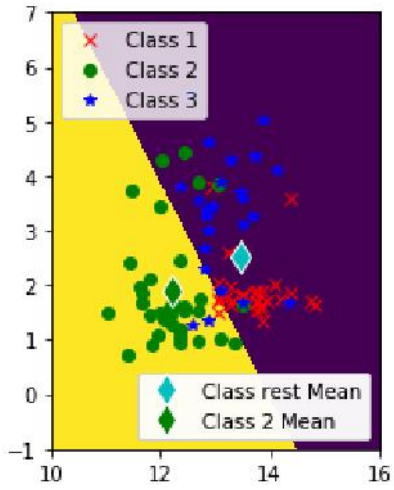
```
sytrain_data = np.genfromtxt('wine_train.csv',delimiter=',')
sytest_data = np.genfromtxt('wine_test.csv', delimiter=',')
sytrain_data_to_col1 = sytrain_data[:,0];
sytrain_data_to_col2 = sytrain_data[:,1];
sytrain_data_to_col3 = sytrain_data[:,13];
sytrain_data_to = np.column_stack((sytrain_data_to_col1,sytrain_data_to_col2,sytrain_da
sytest_data_to_col1 = sytest_data[:,0];
sytest_data_to_col2 = sytest_data[:,1];
sytest_data_to_col3 = sytest_data[:,13];
sytest_data_to = np.column_stack((sytest_data_to_col1,sytest_data_to_col2,sytest_data_
[sample_mean,error_rate,success_rate,error_rate_test,success_rate_test] = classifier_w
print("The classification accuracy of f1 &f2 of Wine Training data set is ",success_ra
print("The classification accurcy of f1 &f2 of Wine Testing data set is ",success_rate_
```

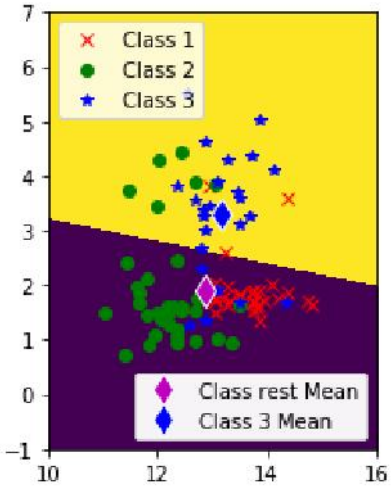
```
The classification accuracy of f1 &f2 of Wine Training data set is  74.15730
337078651
The classification accurcy of f1 &f2 of Wine Testing data set is  70.7865168
5393258
```

In [193]:

```
class_col1  = sample_mean[0:2,0:2].copy();
plotDecBoundaries(sytrain_data,sytrain_data[:,13],class_col1,1)
class_col21  = sample_mean[3,:].copy();
class_col22 = sample_mean[2,:].copy();
class_col2 = np.row_stack((class_col21,class_col22))
plotDecBoundaries(sytrain_data,sytrain_data[:,13],class_col2,2)
class_col31  = sample_mean[5,:].copy();
class_col32 = sample_mean[4,:].copy();
class_col3 = np.row_stack((class_col31,class_col32))
plotDecBoundaries(sytrain_data,sytrain_data[:,13],class_col3,3)
class_colall1  = sample_mean[0,:].copy();
class_colall2 = sample_mean[2,:].copy();
class_colall3 = sample_mean[4,:].copy();
class_col3 = np.row_stack((class_colall1,class_colall2,class_colall3))
```






Class 1
Class 2
Class 3
Class rest Mean
Class 3 Mean
7
6
5
4
3
2
1
0
−1
10
12
14
16