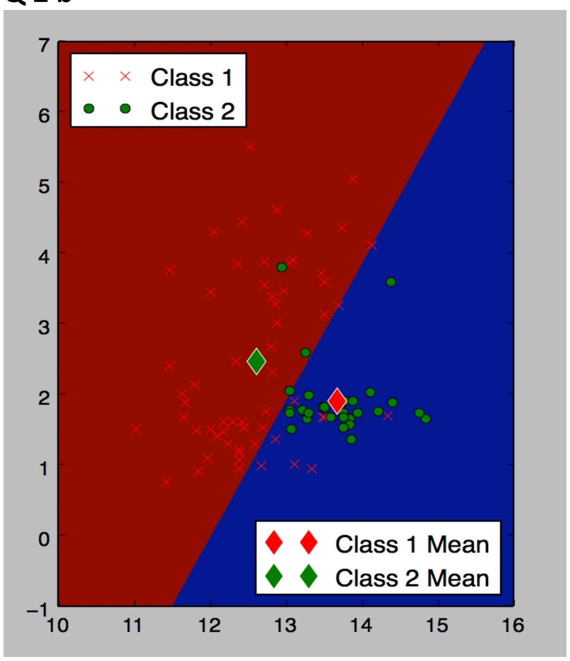
Harikrishna Prabhu 3333077042

**Q 2 a**

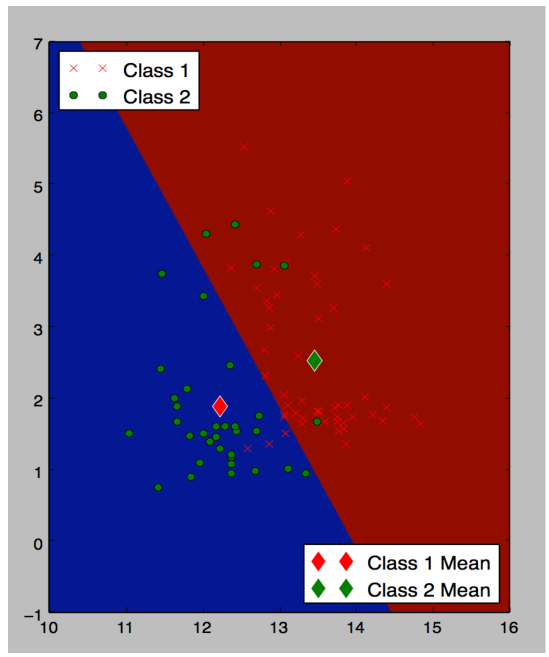
```
guest-wireless-207-151-062-167:HW3 rickerish_nah$ python q2.py
acc_TRAIN: 0.741573033708
acc_TEST: 0.707865168539
no of indeterminate_TRAIN 23
no of indeterminate_TEST 26
guest-wireless-207-151-062-167:HW3 rickerish_nah$
```

**Accuracy of Train = 74% and Test = 70%**

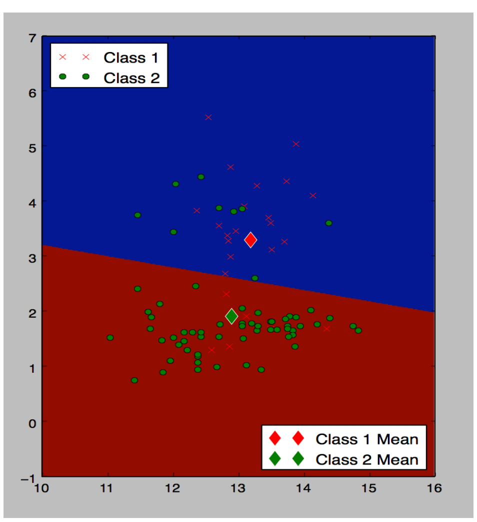
**Q 2 b**




**Class 1 = class 1 and Class 2 = class 2&3**



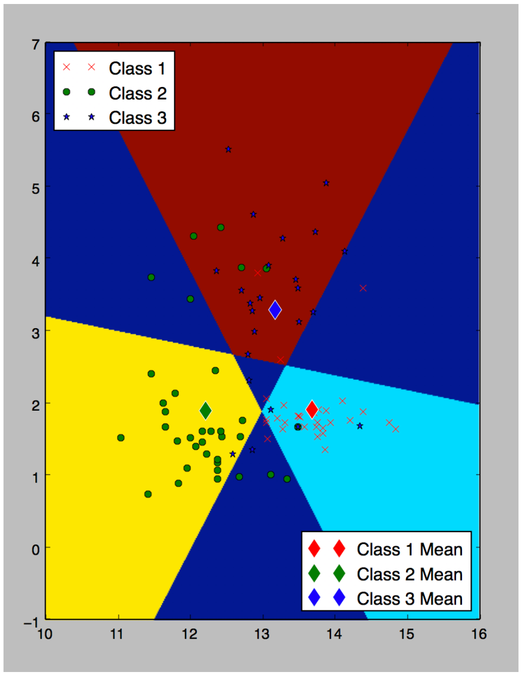

**Class 1 = class 2 and Class 2 = class 1&3**




**Class 1 = class 3 and Class 2 = class 2&1**

**The above figures show the plot for 2 class decision boundary and Region.**

Q 2 c




**Final decision boundary and region for 3 classes and indeterminate region.**

**#MAIN_PROGRAM**

```
import numpy as np
import matplotlib.pyplot as plt
import math
import plotDecisionBound as pB
import plotDec_Bound_b as Pb
import scipy

given_Data_TRAIN=np.genfromtxt("wine_train.csv", delimiter=',')
given_Data_TEST = np.genfromtxt("wine_test.csv", delimiter=',')
collen_TRAIN=len(given_Data_TRAIN)
collen_TEST=len(given_Data_TEST)
train_DATA=np.zeros((collen_TRAIN,2))
class_LABEL=np.zeros(collen_TRAIN)
#label_TRAIN=np.zeros(collen_TRAIN)
avg=np.zeros((6,2)) # 1:2:3:12:13:23
avg_1=np.zeros((2,2))
avg_2=np.zeros((2,2))
avg_3=np.zeros((2,2))
acc_TRAIN=0.0
acc_TEST=0.0
col_length_1=0
```

```
col_length_2=0
col_length_3=0
label_1=np.zeros(collen_TRAIN)
label_2=np.zeros(collen_TRAIN)
label_3=np.zeros(collen_TRAIN)
indeterminate_TRAIN=0
indeterminate_TEST=0
for i in range(0,collen_TRAIN):
	if(given_Data_TRAIN[i][13]!=1):
		label_1[i]=1
	else:
		label_1[i]=2
for i in range(0,collen_TRAIN):
	if(given_Data_TRAIN[i][13]!=2):
		label_2[i]=1
	else:
		label_2[i]=2
for i in range(0,collen_TRAIN):
	if(given_Data_TRAIN[i][13]!=3):
		label_3[i]=2
	else:
		label_3[i]=1


for i in range(0,collen_TRAIN):
	if(given_Data_TRAIN[i][13] ==1):
		col_length_1+=1
	elif(given_Data_TRAIN[i][13]==2):
		col_length_2+=1
	elif(given_Data_TRAIN[i][13]==3):
		col_length_3+=1

#print(col_length_1),;print"\t",;print(col_length_2),;print"\t",;print(col_length_3),;print"\t",;pri
nt(col_length_1+col_length_2+col_length_3)
for i in range(0,2): # because only feature 1 and 2
	for j in range(i+1,2): #nCr comination loop
		#print"i:",;print(i),;print"j:",;print(j)
		class_Feature_Average_TRAIN=np.zeros((6,2))# 1:2:3:12:13:23
		for k in range(0,collen_TRAIN): #Calculating Mean Value
			if k<col_length_1:
				#l1+=1

	class_Feature_Average_TRAIN[0][0]+=given_Data_TRAIN[k][i]/(col_length_1)  #MEAN
OF FEATURE 1 OF CLASS 1(TRAIN)
```

```
        class_Feature_Average_TRAIN[0][1]+=given_Data_TRAIN[k][j]/(col_length_1)  #MEAN
OF FEATURE 2 OF CLASS 1(TRAIN)

        class_Feature_Average_TRAIN[3][0]+=given_Data_TRAIN[k][i]/(col_length_1+col_length
_2)  #MEAN OF FEATURE 1 OF CLASS 1&2(TRAIN)

        class_Feature_Average_TRAIN[3][1]+=given_Data_TRAIN[k][j]/(col_length_1+col_length
_2)

        class_Feature_Average_TRAIN[4][0]+=given_Data_TRAIN[k][i]/(col_length_1+col_length
_3)  #MEAN OF FEATURE 1 OF CLASS 1&3(TRAIN)

        class_Feature_Average_TRAIN[4][1]+=given_Data_TRAIN[k][j]/(col_length_1+col_length
_3)

                        if k>=col_length_1 and k<(col_length_2+col_length_1):
                            #print"Hey Class 2"
                            #l2+=1

        class_Feature_Average_TRAIN[1][0]+=given_Data_TRAIN[k][i]/(col_length_2)

        class_Feature_Average_TRAIN[1][1]+=given_Data_TRAIN[k][j]/(col_length_2)

        class_Feature_Average_TRAIN[3][0]+=given_Data_TRAIN[k][i]/(col_length_1+col_length
_2)  #MEAN OF FEATURE 1 OF CLASS 1&2(TRAIN)

        class_Feature_Average_TRAIN[3][1]+=given_Data_TRAIN[k][j]/(col_length_1+col_length
_2)

        class_Feature_Average_TRAIN[5][0]+=given_Data_TRAIN[k][i]/(col_length_2+col_length
_3)  #MEAN OF FEATURE 1 OF CLASS 2&3(TRAIN)

        class_Feature_Average_TRAIN[5][1]+=given_Data_TRAIN[k][j]/(col_length_2+col_length
_3)
                        if k>=(col_length_1+col_length_2): #and k<collen_TRAIN:
                            #print(k)
                            #l3+=1

        class_Feature_Average_TRAIN[2][0]+=given_Data_TRAIN[k][i]/(col_length_3)

        class_Feature_Average_TRAIN[2][1]+=given_Data_TRAIN[k][j]/(col_length_3)
```

```
        class_Feature_Average_TRAIN[5][0]+=given_Data_TRAIN[k][i]/(col_length_2+col_length
_3)  #MEAN OF FEATURE 1 OF CLASS 2&3(TRAIN)

        class_Feature_Average_TRAIN[5][1]+=given_Data_TRAIN[k][j]/(col_length_2+col_length
_3)

        class_Feature_Average_TRAIN[4][0]+=given_Data_TRAIN[k][i]/(col_length_1+col_length
_3)  #MEAN OF FEATURE 1 OF CLASS 1&3(TRAIN)

        class_Feature_Average_TRAIN[4][1]+=given_Data_TRAIN[k][j]/(col_length_1+col_length
_3)
                k+=1
            #TRAIN
            error_RATE_TRAIN=0 # for now have changed it to success rate: change if
condition to retreat
            dist_1_TRAIN=0
            dist_2_TRAIN=0
            dist_3_TRAIN=0
            dist_12_TRAIN=0
            dist_23_TRAIN=0
            dist_13_TRAIN=0

            for k in range(0,collen_TRAIN): # (feature_1 data - Mean of Feature
1)+(feature_2 data - Mean of Feature 2)
                dist_1_TRAIN=math.sqrt(((given_Data_TRAIN[k][i]-
class_Feature_Average_TRAIN[0][0])**2)+((given_Data_TRAIN[k][j]-
class_Feature_Average_TRAIN[0][1])**2))
                dist_2_TRAIN=math.sqrt(((given_Data_TRAIN[k][i]-
class_Feature_Average_TRAIN[1][0])**2)+((given_Data_TRAIN[k][j]-
class_Feature_Average_TRAIN[1][1])**2))
                dist_3_TRAIN=math.sqrt(((given_Data_TRAIN[k][i]-
class_Feature_Average_TRAIN[2][0])**2)+((given_Data_TRAIN[k][j]-
class_Feature_Average_TRAIN[2][1])**2))
                dist_12_TRAIN=math.sqrt(((given_Data_TRAIN[k][i]-
class_Feature_Average_TRAIN[3][0])**2)+((given_Data_TRAIN[k][j]-
class_Feature_Average_TRAIN[3][1])**2))
                dist_13_TRAIN=math.sqrt(((given_Data_TRAIN[k][i]-
class_Feature_Average_TRAIN[4][0])**2)+((given_Data_TRAIN[k][j]-
class_Feature_Average_TRAIN[4][1])**2))
                dist_23_TRAIN=math.sqrt(((given_Data_TRAIN[k][i]-
class_Feature_Average_TRAIN[5][0])**2)+((given_Data_TRAIN[k][j]-
class_Feature_Average_TRAIN[5][1])**2))
```

```
                    if dist_1_TRAIN<dist_23_TRAIN and dist_13_TRAIN<dist_2_TRAIN and
dist_12_TRAIN<dist_3_TRAIN:
                        if given_Data_TRAIN[k][13]==1:
                            error_RATE_TRAIN+=1
                            class_LABEL[k]=1
                        else:
                            indeterminate_TRAIN+=1
                            #class_LABEL[k]=0
                    elif dist_2_TRAIN<dist_13_TRAIN and dist_23_TRAIN<dist_1_TRAIN and
dist_12_TRAIN<dist_3_TRAIN :
                        if given_Data_TRAIN[k][13]==2:
                            error_RATE_TRAIN+=1
                            class_LABEL[k]=2
                        else:
                            indeterminate_TRAIN+=1
                            #class_LABEL[k]=0
                    elif dist_3_TRAIN<dist_12_TRAIN and dist_23_TRAIN<dist_1_TRAIN  and
dist_13_TRAIN<dist_2_TRAIN:
                        if given_Data_TRAIN[k][13]==3:
                            error_RATE_TRAIN+=1
                            class_LABEL[k]=3
                        else:
                            indeterminate_TRAIN+=1
                            #class_LABEL[k]=0
                    else:
                        indeterminate_TRAIN+=1
                        #class_LABEL[k]=0
                    '''
                    print("Element :"),;print(k)
                    print("Class :"),;print(given_Data_TRAIN[k,13])
                    print("Distance 1 :"),;print(dist_1_TRAIN)
                    print("Distance 2 :"),;print(dist_2_TRAIN)
                    print("Distance 3 :"),;print(dist_3_TRAIN)
                    print("Distance 12 :"),;print(dist_12_TRAIN)
                    print("Distance 13 :"),;print(dist_13_TRAIN)
                    print("Distance 23 :"),;print(dist_23_TRAIN)
                    print("acc_RATE_TRAIN:"),;print(error_RATE_TRAIN)

            print("indeterminate_TRAIN"),;print(indeterminate_TRAIN),;print("\n\n\n")
                    '''
                    k+=1
                #TEST
                error_RATE_TEST=0
```

```
            dist_1_TEST=0
            dist_2_TEST=0
            dist_3_TEST=0
            dist_12_TEST=0
            dist_23_TEST=0
            dist_13_TEST=0

            for k in range(0,collen_TRAIN): # (feature_1 data - Mean of Feature 
1)+(feature_2 data - Mean of Feature 2)
                        dist_1_TEST=math.sqrt(((given_Data_TEST[k][i]-
class_Feature_Average_TRAIN[0][0])**2)+((given_Data_TEST[k][j]-
class_Feature_Average_TRAIN[0][1])**2))
                        dist_2_TEST=math.sqrt(((given_Data_TEST[k][i]-
class_Feature_Average_TRAIN[1][0])**2)+((given_Data_TEST[k][j]-
class_Feature_Average_TRAIN[1][1])**2))
                        dist_3_TEST=math.sqrt(((given_Data_TEST[k][i]-
class_Feature_Average_TRAIN[2][0])**2)+((given_Data_TEST[k][j]-
class_Feature_Average_TRAIN[2][1])**2))
                        dist_12_TEST=math.sqrt(((given_Data_TEST[k][i]-
class_Feature_Average_TRAIN[3][0])**2)+((given_Data_TEST[k][j]-
class_Feature_Average_TRAIN[3][1])**2))
                        dist_13_TEST=math.sqrt(((given_Data_TEST[k][i]-
class_Feature_Average_TRAIN[4][0])**2)+((given_Data_TEST[k][j]-
class_Feature_Average_TRAIN[4][1])**2))
                        dist_23_TEST=math.sqrt(((given_Data_TEST[k][i]-
class_Feature_Average_TRAIN[5][0])**2)+((given_Data_TEST[k][j]-
class_Feature_Average_TRAIN[5][1])**2))


                        if dist_1_TEST<dist_23_TEST and dist_13_TEST<dist_2_TEST and 
dist_12_TEST<dist_3_TEST:
                                    if given_Data_TEST[k][13]==1:
                                                error_RATE_TEST+=1
                                    else:
                                                indeterminate_TEST+=1
                        elif dist_2_TEST<dist_13_TEST and dist_23_TEST<dist_1_TEST and 
dist_12_TEST<dist_3_TEST :
                                    if given_Data_TEST[k][13]==2:
                                                error_RATE_TEST+=1
                                    else:
                                                indeterminate_TEST+=1
                        elif dist_3_TEST<dist_12_TEST and dist_23_TEST<dist_1_TEST  and 
dist_13_TEST<dist_2_TEST:
                                    if given_Data_TEST[k][13]==3:
```

```
                    error_RATE_TEST+=1
                else:
                    indeterminate_TEST+=1
            else:
                indeterminate_TEST+=1

        acc_TRAIN+=error_RATE_TRAIN
        acc_TEST+=error_RATE_TEST
        acc_TRAIN=acc_TRAIN/collen_TRAIN
        acc_TEST=acc_TEST/collen_TEST
        print("acc_TRAIN:"),;print(acc_TRAIN)
        print("acc_TEST:"),;print(acc_TEST)
        print("no of indeterminate_TRAIN"),;print(indeterminate_TRAIN)
        print("no of indeterminate_TEST"),;print(indeterminate_TEST)
        #print("Col:TRAIN:"),;print(collen_TRAIN)
        #print("Col:TEST:"),;print(collen_TEST)
        j+=1
    #print("Accuracy rate:"),;print(acc_TRAIN)
    i+=1
avg[0][0]=class_Feature_Average_TRAIN[0][0]
avg[0][1]=class_Feature_Average_TRAIN[0][1]
avg[1][0]=class_Feature_Average_TRAIN[1][0]
avg[1][1]=class_Feature_Average_TRAIN[1][1]
avg[2][0]=class_Feature_Average_TRAIN[2][0]
avg[2][1]=class_Feature_Average_TRAIN[2][1]
avg[3][0]=class_Feature_Average_TRAIN[3][0]
avg[3][1]=class_Feature_Average_TRAIN[3][1]
avg[4][0]=class_Feature_Average_TRAIN[4][0]
avg[4][1]=class_Feature_Average_TRAIN[4][1]
avg[5][0]=class_Feature_Average_TRAIN[5][0]
avg[5][1]=class_Feature_Average_TRAIN[5][1]

avg_1[0][0]=class_Feature_Average_TRAIN[0][0]
avg_1[0][1]=class_Feature_Average_TRAIN[0][1]
avg_1[1][0]=class_Feature_Average_TRAIN[5][0]
avg_1[1][1]=class_Feature_Average_TRAIN[5][1]


avg_2[0][0]=class_Feature_Average_TRAIN[1][0]
avg_2[0][1]=class_Feature_Average_TRAIN[1][1]
avg_2[1][0]=class_Feature_Average_TRAIN[4][0]
avg_2[1][1]=class_Feature_Average_TRAIN[4][1]

avg_3[0][0]=class_Feature_Average_TRAIN[2][0]
```

```
avg_3[0][1]=class_Feature_Average_TRAIN[2][1]
avg_3[1][0]=class_Feature_Average_TRAIN[3][0]
avg_3[1][1]=class_Feature_Average_TRAIN[3][1]


#pB.plotDecBoundary(given_Data_TRAIN[:,0:2],given_Data_TRAIN[:,13],avg[:,:]) #FOR Q2 part C
#Pb.plotDecBoundary(given_Data_TRAIN[:,0:2],label_1,avg_1[:,:]) # 1 vs 23
#Pb.plotDecBoundary(given_Data_TRAIN[:,0:2],label_2,avg_2[:,:]) # 2 vs 13
Pb.plotDecBoundary(given_Data_TRAIN[:,0:2],label_3,avg_3[:,:]) # 3 vs 12
```

**#PLOT_ in Q 2 B**

```
################################################
## EE559 Harikrishna Prabhu
################################################

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

def plotDecBoundary(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[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.
    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]], 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')
    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,
markerfacecolor='r', markeredgecolor='w')
    m2, = plt.plot(sample_mean[1,0], sample_mean[1,1], 'gd', markersize=12,
markerfacecolor='g', markeredgecolor='w')
    if nclass == 3:
```

```
        m3, = plt.plot(sample_mean[2,0], sample_mean[2,1], 'bd', markersize=12,
markerfacecolor='b', markeredgecolor='w')

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

    plt.gca().add_artist(l1)

    plt.show()
```

**#PLOT in Q C**

```
#################################################
## EE559 HW Harikrishna
#################################################

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

def plotDecBoundary(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[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.
    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]], 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')
    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,
markerfacecolor='r', markeredgecolor='w')
    m2, = plt.plot(sample_mean[1,0], sample_mean[1,1], 'gd', markersize=12,
markerfacecolor='g', markeredgecolor='w')
    if nclass == 3:
```

```
        m3, = plt.plot(sample_mean[2,0], sample_mean[2,1], 'bd', markersize=12,
markerfacecolor='b', markeredgecolor='w')

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

    plt.gca().add_artist(l1)

    plt.show()
```