```python
"""
Wine Dataset (for question 2)
Name: Haolun Cheng
USCID: 1882563827
EE559 HW2
"""

import csv
import numpy as np
import matplotlib.pyplot as plt
from plotDecBoundaries import plotDecBoundaries

clas1xtotal = []
clas1ytotal = []
clas2xtotal = []
clas2ytotal = []
clas3xtotal = []
clas3ytotal = []
allclasstwofeatures = []
allclasslabels = []
allclassmeans = []
feature1 = []
feature2 = []
feature3 = []
label1 = []
label2 = []
label3 = []

# Open train csv file for training the classifier
with open('wine_train.csv', 'r') as train:
    training_set = csv.reader(train)

    # Train
    for line in training_set:
        x, y, label = line[0], line[1], line[-1]
        allclasstwofeatures.append((float(x), float(y)))
        allclasslabels.append(label)
        if label == '1':
            clas1xtotal.append(float(x))
            clas1ytotal.append(float(y))
            feature1.append((float(x), float(y)))
            label1.append(label)
        elif label == '2':
            clas2xtotal.append(float(x))
            clas2ytotal.append(float(y))
            feature2.append((float(x), float(y)))
            label2.append(label)
        else:
            clas3xtotal.append(float(x))
            clas3ytotal.append(float(y))
            feature3.append((float(x), float(y)))
            label3.append(label)

train.close()

# Compute mean for each class
clas1xmean = np.mean(clas1xtotal)
clas1ymean = np.mean(clas1ytotal)

```

```python
clas2xmean = np.mean(clas2xtotal)
clas2ymean = np.mean(clas2ytotal)

clas3xmean = np.mean(clas3xtotal)
clas3ymean = np.mean(clas3ytotal)

clas1_mean_point = np.array((clas1xmean, clas1ymean))
clas2_mean_point = np.array((clas2xmean, clas2ymean))
clas3_mean_point = np.array((clas3xmean, clas3ymean))

# Calculate the linear equations for each pair of points
# class1 and class2 equation
slope1 = (clas2ymean - clas1ymean) / (clas2xmean - clas1xmean) # get the slope
slope12 = -1 / slope1
midPoint1x = (clas2xmean + clas1xmean) / 2
midPoint1y = (clas2ymean + clas1ymean) / 2
intercept12 = midPoint1y - (slope12 * midPoint1x)
clas1sign = (slope12 * clas1xmean + intercept12) - clas1ymean
g12_1, g12_2 = 0, 0
if clas1sign > 0:
    g12_1 = 1
    g12_2 = -1
elif clas1sign < 0:
    g12_1 = -1
    g12_2 = 1

# class1 and class3 equation
slope2 = (clas3ymean - clas1ymean) / (clas3xmean - clas1xmean) # get the slope
slope13 = -1 / slope2
midPoint2x = (clas3xmean + clas1xmean) / 2
midPoint2y = (clas3ymean + clas1ymean) / 2
intercept13 = midPoint2y - (slope13 * midPoint2x)
clas3sign = (slope13 * clas3xmean + intercept13) - clas3ymean
g13_1, g13_3 = 0, 0
if clas3sign > 0:
    g13_3 = 1
    g13_1 = -1
elif clas3sign < 0:
    g13_3 = -1
    g13_1 = 1

# class2 and class3 equation
slope3 = (clas3ymean - clas2ymean) / (clas3xmean - clas2xmean) # get the slope
slope23 = -1 / slope3
midPoint3x = (clas2xmean + clas3xmean) / 2
midPoint3y = (clas2ymean + clas3ymean) / 2
intercept23 = midPoint3y - (slope23 * midPoint3x)
clas2sign = (slope23 * clas2xmean + intercept23) - clas2ymean
g23_2, g23_3 = 0, 0
if clas2sign > 0:
    g23_2 = 1
    g23_3 = -1
elif clas2sign < 0:
    g23_2 = -1
    g23_3 = 1

# Classify data points (training set)
```

```python
countTrainingError = 0
totalTrainingPoints = 0
with open('wine_train.csv', 'r') as training:
    train_set = csv.reader(training)

    for line in train_set:
        count0, count1, count2, count3 = 0, 0, 0, 0
        totalTrainingPoints += 1
        x, y, label = line[0], line[1], line[-1]
        result12 = (slope12 * float(x) + intercept12) - float(y)
        result13 = (slope13 * float(x) + intercept13) - float(y)
        result23 = (slope23 * float(x) + intercept23) - float(y)

        # Use the OvO rule
        if result12 > 0:
            if g12_1 == 1:
                count1 += 1
            else:
                count2 += 1
        elif result12 < 0:
            if g12_1 == -1:
                count1 += 1
            else:
                count2 += 1
        else:
            count0 += 1

        if result13 > 0:
            if g13_1 == 1:
                count1 += 1
            else:
                count3 += 1
        elif result13 < 0:
            if g13_1 == -1:
                count1 += 1
            else:
                count3 += 1
        else:
            count0 += 1

        if result23 > 0:
            if g23_2 == 1:
                count2 += 1
            else:
                count3 += 1
        elif result23 < 0:
            if g23_2 == -1:
                count2 += 1
            else:
                count3 += 1
        else:
            count0 += 1

        # Classify to class
        if count0 != 0:
            countTrainingError += count0
        else:
            if count1 > count2 and count1 > count3:
                if label != '1':
                    countTrainingError += 1
            if count2 > count1 and count2 > count3:
                if label != '2':
                    countTrainingError += 1
            if count3 > count2 and count3 > count1:
                if label != '3':
                    countTrainingError += 1
training.close()

# Fine error rate for training set
error_rate = float(countTrainingError) / float(totalTrainingPoints)
print("Error rate for the training set: " + str(error_rate))

# Classify data points (test set)
countTestError = 0
totalTestPoints = 0
with open('wine_test.csv', 'r') as test:
    test_set = csv.reader(test)

    for line in test_set:
        count0, count1, count2, count3 = 0, 0, 0, 0
        totalTestPoints += 1
        x, y, label = line[0], line[1], line[-1]
        testPoint = np.array((x, y))
        result12 = (slope12 * float(x) + intercept12) - float(y)
        result13 = (slope13 * float(x) + intercept13) - float(y)
        result23 = (slope23 * float(x) + intercept23) - float(y)

        # Use the OvO rule
        if result12 > 0:
            if g12_1 == 1:
                count1 += 1
            else:
                count2 += 1
        elif result12 < 0:
            if g12_1 == -1:
                count1 += 1
            else:
                count2 += 1
        else:
            count0 += 1

        if result13 > 0:
            if g13_1 == 1:
                count1 += 1
            else:
                count3 += 1
        elif result13 < 0:
            if g13_1 == -1:
                count1 += 1
            else:
                count3 += 1
        else:
            count0 += 1

        if result23 > 0:
            if g23_2 == 1:
                count2 += 1
            else:
                count3 += 1
        elif result23 < 0:
            if g23_2 == -1:
                count2 += 1
            else:
                count3 += 1
        else:
            count0 += 1

        # Classify to class
        if count0 != 0:
            countTestError += count0
        else:
            if count1 > count2 and count1 > count3:
                if label != '1':
                    countTestError += 1
            if count2 > count1 and count2 > count3:
                if label != '2':
                    countTestError += 1
            if count3 > count2 and count3 > count1:
                if label != '3':
                    countTestError += 1

test.close()

# Fine error rate for test set
error_rate = float(countTestError) / float(totalTestPoints)
print("Error rate for the test set: " + str(error_rate))

# Plot the data points
xAxis1 = [i[0] for i in feature1]
yAxis1 = [i[1] for i in feature1]
xAxis2 = [i[0] for i in feature2]
yAxis2 = [i[1] for i in feature2]
xAxis3 = [i[0] for i in feature3]
yAxis3 = [i[1] for i in feature3]
plt.plot(xAxis1, yAxis1, 'r.', xAxis2, yAxis2, 'b^', xAxis3, yAxis3, 'g.')
plt.xlabel('Feature1')
plt.ylabel('Feature2')
plt.title('Feature Plot of all Elements')
plt.show()

# Class 1 & 2 decision boundaries and regions variables
classmeans12 = [[clas1xmean, clas1ymean], [clas2xmean, clas2ymean]]
features12 = feature1 + feature2
labels12 = label1 + label2
twofeatures12 = np.array(features12).astype(float)
claslabels12 = np.array(labels12).astype(float)
samplemeans12 = np.array(classmeans12).astype(float)

# Class 1 & 3 decision boundaries and regions variables
classmeans13 = [[clas1xmean, clas1ymean], [clas3xmean, clas3ymean]]
features13 = feature1 + feature3
labels13 = label1 + label3
twofeatures13 = np.array(features13).astype(float)
claslabels13 = np.array(labels13).astype(float)
samplemeans13 = np.array(classmeans13).astype(float)

# Class 2 & 3 decision boundaries and regions variables
classmeans23 = [[clas2xmean, clas2ymean], [clas3xmean, clas3ymean]]
features23 = feature2 + feature3
labels23 = label2 + label3
```

```python
twofeatures23 = np.array(features23).astype(float)
claslabels23 = np.array(labels23).astype(float)
samplemeans23 = np.array(classmeans23).astype(float)

#Final decision boundaries and regions variables
allclassmeans = [[clas1xmean, clas1ymean], [clas2xmean, clas2ymean],
[clas3xmean, clas3ymean]]
twofeatures = np.array(allclasstwofeatures).astype(float)
claslabels = np.array(allclasslabels).astype(float)
samplemeans = np.array(allclassmeans).astype(float)

# Plot the decision boundaries
plotDecBoundaries(twofeatures, claslabels, samplemeans12, class1=1, class2=2)
plotDecBoundaries(twofeatures, claslabels, samplemeans13, class1=1, class2=3)
plotDecBoundaries(twofeatures, claslabels, samplemeans23, class1=2, class2=3)
plotDecBoundaries(twofeatures, claslabels, samplemeans)
```

```python
#################################################
## 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, sample_mean, class1 = 0, class2 = 1):

    #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))

    # Total number of means
    nmean = np.unique(sample_mean).shape[0]

    # 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
    l = plt.legend(('Class 1', 'Class 2', 'Class 3'), 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 nmean == 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 nmean == 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 ' + str(class1) + ' Mean', 'Class ' + str(class2) + ' Mean'], loc=4)

    plt.gca().add_artist(l1)

    plt.show()
```