Homework 4, Problem 3 Classification on simulated data

ECE C143A/C243A, Spring Quarter 2022, Prof. J.C. Kao, TAs T. Monsoor, W. Yu

Background

We will now apply the results of Problems 1 and 2 to simulated data. The dataset can be found on CCLE as ps4_simdata.mat.

The following describes the data format. The .mat file has a single variable named 'trial', which is a structure of dimensions (20 data points) × (3 classes). The nth data point for the kth class is denoted via:

`data['trial'][n][k][0]` where n = 0,...,19 and k = 0,1,2 are the data points and classes respectively. The `[0]` after `[n][k]` is an artifact of how the `.mat` file is imported into Python. You can get a clearer sense of this below in the plotting scripts.

To make the simulated data as realistic as possible, the data are non-negative integers, so one can think of them as spike counts. With this analogy, there are D = 2 neurons and K = 3 stimulus conditions.

Please follow steps (a)–(e) below for each of the three models. The result of this problem should be three separate plots, one for each model. These plots will be similar in spirit to Figure 4.5 in PRML.

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
import scipy.special
import scipy.io as sio
import math

# Load matplotlib images inline
%matplotlib inline

# Reloading any code written in external .py files.
%load_ext autoreload
%autoreload 2

data = sio.loadmat('ps4_simdata.mat') # load the .mat file.
NumData = data['trial'].shape[0]
NumClass = data['trial'].shape[1]
```

```
The autoreload extension is already loaded. To reload it, use:
  %reload_ext autoreload
```

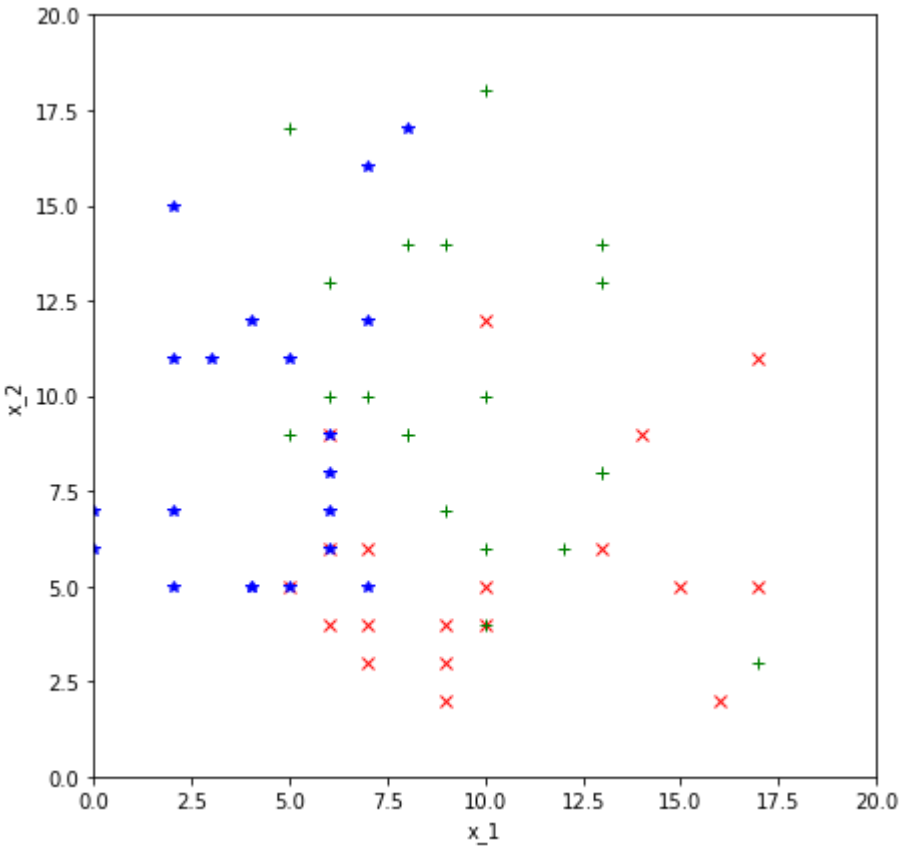
(a) Plot the data points

Here, to get you oriented on the dataset, we'll give you code that plots the data points. You do not have to write any new code here, but you should review this code to understand it, since we'll ask you to make plots in later parts of the notebook.

Here, we plot the data points in a two-dimensional space. For classes k = 1, 2, 3, we use a red ×, green +, and blue $*$ for each data point, respectively. The axis limits of the plot are between 0 and 20. You should use these axes bounds for the rest of the homework.

In [ ]:
```
# a
plt.figure(figsize=(7,7))
#=====================================================#
# PLOTTING CODE BELOW
#=====================================================#
dataArr =  np.zeros((NumClass,NumData ,2)) # dataArr contains the points
for classIX in range(NumClass):
    for dataIX in range(NumData):
        x = data['trial'][dataIX,classIX][0][0][0]
        y = data['trial'][dataIX,classIX][0][1][0]
        dataArr[classIX,dataIX,0]=x
        dataArr[classIX,dataIX,1]=y
MarkerPat=np.array(['rx','g+','b*'])

for classIX in range(NumClass):
    for dataIX in range(NumData):
        plt.plot(dataArr[classIX,dataIX,0],dataArr[classIX,dataIX,1],MarkerPat
[classIX])

#=====================================================#
# END PLOTTING CODE
#=====================================================#
plt.axis([0,20,0,20])
plt.xlabel('x_1')
plt.ylabel('x_2')
```

Out[ ]: Text(0,0.5,'x_2')



(b) (15 points) Find the ML model parameters

Find the ML model parameters, for each model, using results from Problem 1. Report the values of all the ML parameters for each model. (Please print the names and values of all the ML parameters in Jupyter Notebook)

In [ ]:
```
#=====================================================#
# YOUR CODE HERE:
#   Find the parameters for each model you derived in problem 1 using
#   the simulated data, and print out the values of each parameter.
#
#   To facilitate plotting later on, we're going to ask you to
#   format the data in the following way.
#
#   (1) Keep three dictionaries, modParam1, modParam2, and modParam3
#       which contain the model parameters for model 1 (Gaussian, shared cov),
#       model 2 (Gaussian, class specific cov), and model 3 (Poisson).
#
#       The Python dictionary is like a MATLAB struct. e.g., you can declare:
#       modParam1 = {} # declares the dictionary
#       modParam1['pi'] = np.array((0.33, 0.33, 0.34)) # sets the field 'pi' t
o be
#         an np.array of size (3,) containing the class probabilities.
#
#   (2) modParam1 has the following structure
#
#     modParam1['pi'] is an np.array of size (3,) containing the class probabi
lities.
#     modParam1['mean'] is an np.array of size (3,2) containing the class mean
s.
#     modParam1['cov'] is an np.array of size (2,2) containing the shared cov.
#
#   (3) modParam2:
#
#     modParam2['pi'] is an np.array of size (3,) containing the class probabi
lities.
#     modParam2['mean'] is an np.array of size (3,2) containing the class mean
s.
#     modParam2['cov'] is an np.array of size (3,2,2) containing the cov for e
ach of the 3 classes.
#
#   (4) modParam3:
#     modParam2['pi'] is an np.array of size (3,) containing the class probabi
lities.
#     modParam2['mean'] is an np.array of size (3,2) containing the Poisson pa
rameters for each class.
#
#   These should be consistent with the print statement after this code block.
#
#   HINT: the np.mean and np.cov functions ought simplify the code.
#
#=====================================================#
modParam1, modParam2, modParam3 ={},{},{}
modParam1['pi']= np.array([1/NumClass]*NumClass)
modParam1['mean'] = np.mean(dataArr, axis=1)
modParam1['cov'] = np.cov(dataArr.reshape((NumClass*NumData,2)).T)

modParam2['pi']= np.array([1/NumClass]*NumClass)
modParam2['mean'] = np.mean(dataArr, axis=1)
modParam2['cov'] = np.zeros((NumClass, 2,2))
modParam3['pi']= np.array([1/NumClass]*NumClass)
modParam3['mean'] = np.zeros((NumClass, 2))
```

```
for classix in range(NumClass):
    modParam2['cov'][classix] = np.cov(dataArr[classix].T)
    modParam3['mean'][classix] = np.mean(dataArr[classix], axis=0)

#=====================================================#
# END YOUR CODE
#=====================================================#

# Print out the model parameters
print("Model 1:")
print("Class priors:")
print( modParam1['pi'])
print("Means:")
print( modParam1['mean'])
print("Cov:")
print( modParam1['cov'])

print("model 2:")
print("Class priors:")
print( modParam2['pi'])
print("Means:")
print( modParam2['mean'])
print("Cov1:")
print( modParam2['cov'][0])
print("Cov2:")
print( modParam2['cov'][1])
print("Cov3:")
print( modParam2['cov'][2])

print("model 3:")
print("Class priors:")
print( modParam3['pi'])
print("Lambdas:")
print( modParam3['mean'])
```

```
Model 1:
Class priors:
[ 0.33333333  0.33333333  0.33333333]
Means:
[[ 10.75   5.55]
 [  9.6   10.1 ]
 [  4.3    9.  ]]
Cov:
[[ 20.20649718  -2.47146893]
 [ -2.47146893  16.54548023]]
model 2:
Class priors:
[ 0.33333333  0.33333333  0.33333333]
Means:
[[ 10.75   5.55]
 [  9.6   10.1 ]
 [  4.3    9.  ]]
Cov1:
[[ 22.09210526   2.25      ]
 [  2.25         7.62894737]]
Cov2:
[[ 10.04210526  -4.95789474]
 [ -4.95789474  16.62105263]]
Cov3:
[[  5.69473684   2.63157895]
 [  2.63157895  15.26315789]]
model 3:
Class priors:
[ 0.33333333  0.33333333  0.33333333]
Lambdas:
[[ 10.75   5.55]
 [  9.6   10.1 ]
 [  4.3    9.  ]]
```

(c) Plot the ML mean

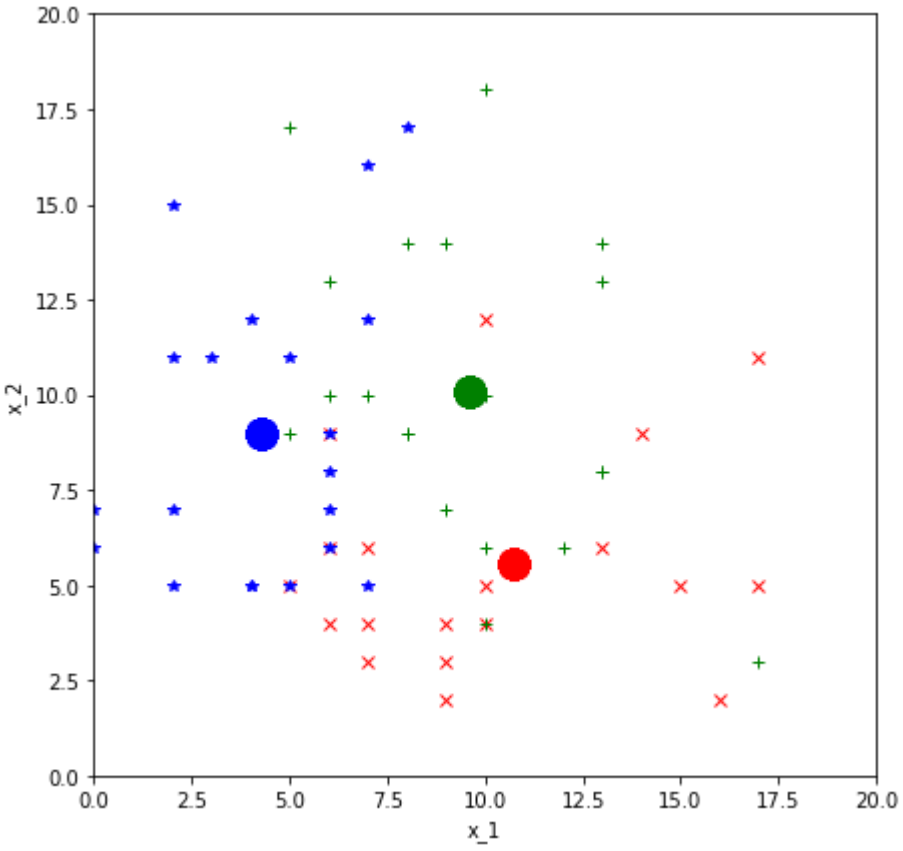
The following code plots the ML mean for each class. You should read the code to understand what is going on. If you followed our instructions on how to format the data, you should not have to modify any code here. This plot needs to be generated for us to check if you implemented the means correctly. You may also use this as a sanity check.

If you made modifications in the way the data is formatted, you need to change this code to visualize the ML means

For each class, we plot the ML mean on top of the data using a solid dot of the appropriate color. We set the marker size of this dot to be much larger than the marker sizes you used in part a, so the dot is easy to see.

In [ ]:
```
# c
plt.figure(figsize=(7,7))
#=====================================================#
# ML MEAN PLOT CODE HERE.
#=====================================================#
colors = ['r.','g.','b.']
for classIX in range(NumClass):
    for dataIX in range(NumData):
        plt.plot(dataArr[classIX,dataIX,0],dataArr[classIX,dataIX,1],MarkerPat
[classIX])
        plt.plot(modParam1['mean'][classIX,0],modParam1['mean'][classIX,1],col
ors[classIX],markersize=30)
#=====================================================#
# END CODE
#=====================================================#
plt.axis([0,20,0,20])
plt.xlabel('x_1')
plt.ylabel('x_2')
```

Out[ ]: Text(0,0.5,'x_2')



(d) Plot the ML covariance ellipsoids.

The following code plots the ML covariance for each class. You should read the code to understand what is going on. If you followed our instructions on how to format the data, you should not have to modify any code here. This plot needs to be generated for us to check if you implemented the means correctly. You may also use this as a sanity check.

If you made modifications in the way the data is formatted, you need to change this code to visualize the ML covariance ellipsoids

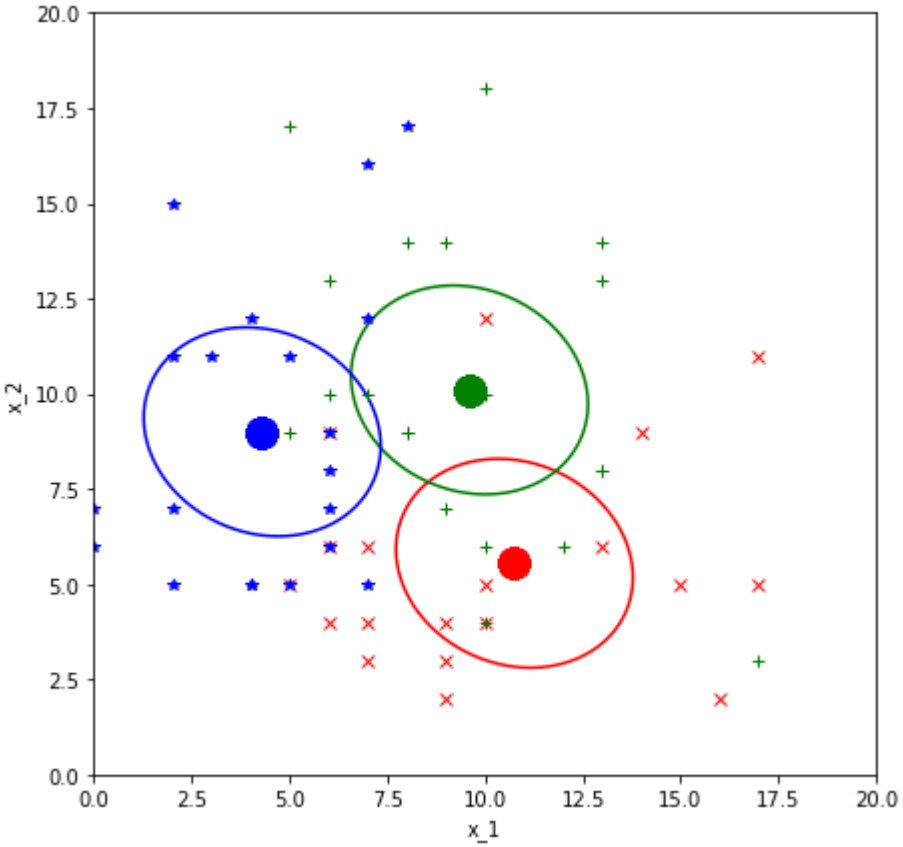
For each class, we plot the ML covariance using an ellipse of the appropriate color. We plot this on top of the data with the means. This part only encapsulates the Gaussian models i) and ii). We generate separate plots for models i) and ii).

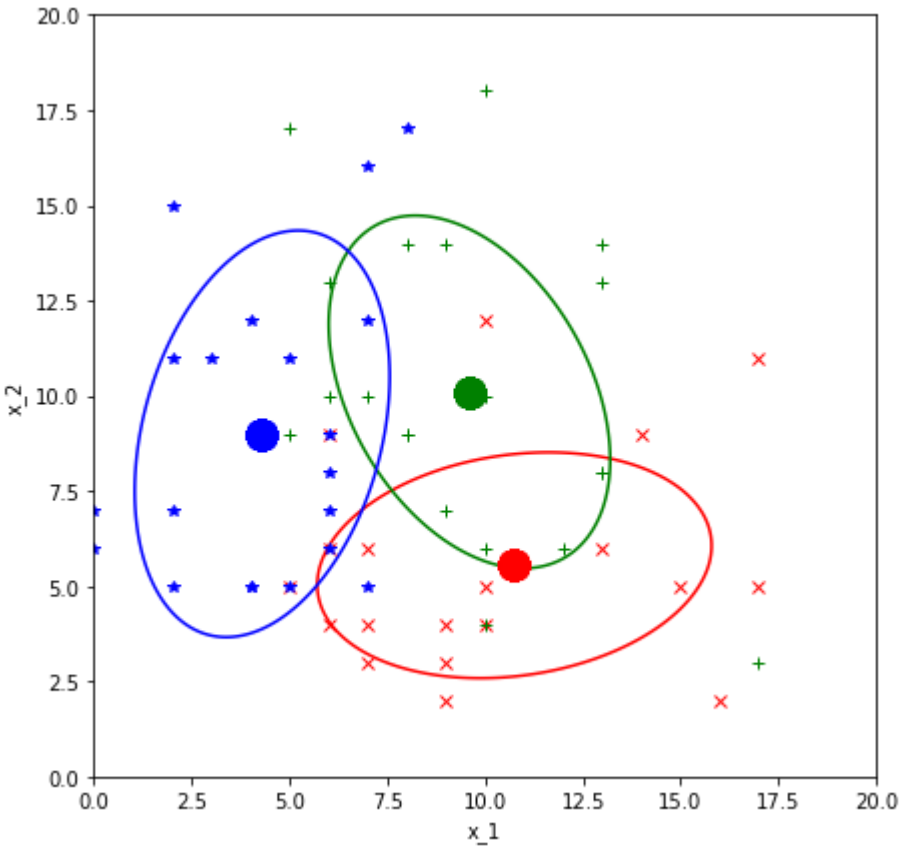
We use of `plt.contour` can be used to draw an iso-probability contour for each class. To aid interpretation, the contour should be drawn at the same probability level for each class. We call `plt.contour(X, Y, Z, levels = level, colors = color)`.

For this specific problem, we choose the contour level so you can see each ellipsoid reasonably, e.g. levels = 0.007, where X and Y are obtained via `[X,Y] = np.meshgrid(np.linspace(0, 20, N), np.linspace(0, 20, N))`, where N is the number of partitions, e.g. N = 20. Z is the function value, Please set the contour color to be the same as data points color.

Please understand this code, as it will facilitate the last part of this notebook where we ask you to generate a plot with classification boundaries. In prior years we asked the students to generate this, but have provided it here to reduce the homework load.

In [ ]:
```
# d
#=====================================================#
# ML COV PLOT CODE HERE.
#=====================================================#
colors2 = ['r','g','b']
modParam = [modParam1 , modParam2]
for modelIX in range(2):
    plt.figure(modelIX,figsize=(7,7))
    for classIX in range(NumClass):
        for dataIX in range(NumData):
            #plot the points and their means, just like before
            plt.plot(dataArr[classIX,dataIX,0],dataArr[classIX,dataIX,1],Marke
rPat[classIX])
            plt.plot(modParam[modelIX]['mean'][classIX,0],modParam[modelIX]['m
ean'][classIX,1],colors[classIX],markersize=30)
        plt.axis([0,20,0,20])
        plt.xlabel('x_1')
        plt.ylabel('x_2')
        MarkerCol=['r','g','b']

    #now begins plotting the elipse
    for classIX in range(NumClass):
        currMean=modParam[modelIX]['mean'][classIX ,:]
        if(modelIX == 0):
            currCov=modParam[modelIX]['cov']
        else:
            currCov=modParam[modelIX]['cov'][classIX]
        xl = np.linspace(0, 20, 201)
        yl = np.linspace(0, 20, 201)
        [X,Y] = np.meshgrid(xl,yl)

        Xlong = np.reshape(X-currMean[0],(np.prod(np.size(X))))
        Ylong = np.reshape(Y-currMean[1],(np.prod(np.size(X))))
        temp = np.row_stack([Xlong,Ylong])
        Zlong = []
        for i in range(np.size(Xlong)):
            Zlong.append(np.matmul(np.matmul(temp[:,i], np.linalg.inv(currCov
)), temp[:,i].T))
        Zlong = np.matrix(Zlong)
        Zlong = np.exp(-Zlong/2)/np.sqrt((2*np.pi)*(2*np.pi)*np.linalg.det(cur
rCov))
        Z = np.reshape(Zlong,X.shape)
        isoThr=0.007
        plt.contour(X,Y,Z,levels = isoThr,colors = colors2[classIX])
#=====================================================#
# END CODE
#=====================================================#
```


20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
x_2
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
x_1


20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
x_2
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
x_1

(e) (15 points) Plot multi-class decision boundaries

Plot multi-class decision boundaries corresponding to the decision rule

$$\hat{k} = argmax_k \, P(C_k|x)$$

and label each decision region with the appropriate class k. This should be plotted on top of your means and the covariance ellipsoids. Thus, you should start by copying and pasting code from the prior Jupyter Notebook cell.

To plot the multi-class decision boundaries, we recommend that you do it by classifying a dense sampling of the two-dimensional data space.

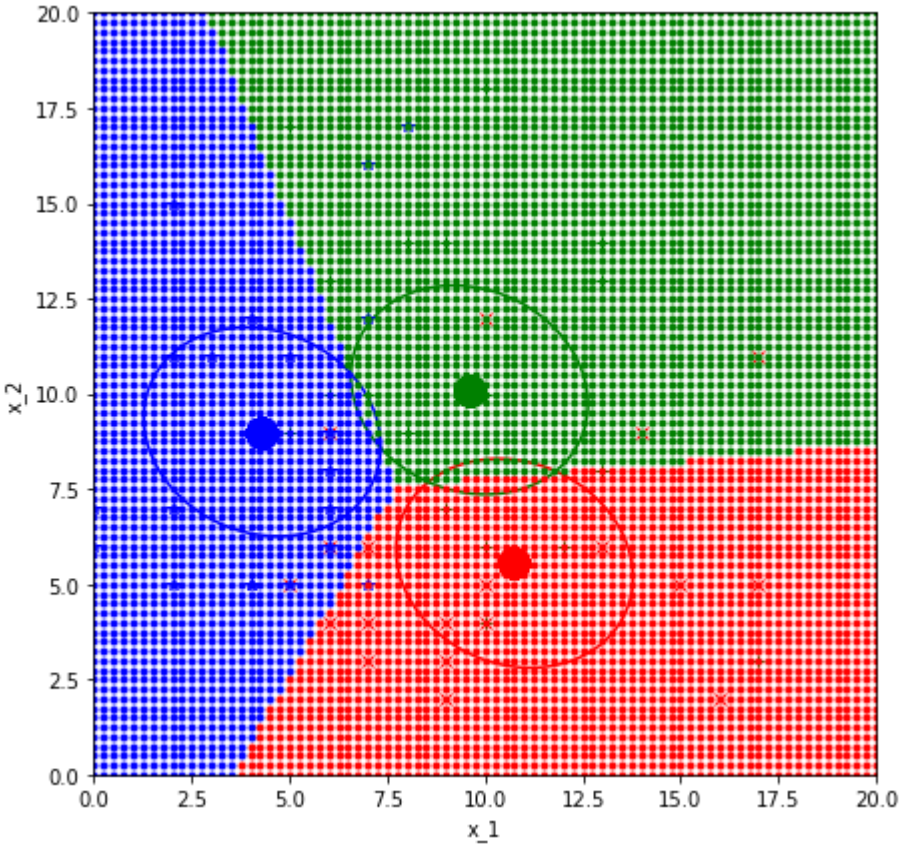
Hint 1: You can do this by calling `[X,Y] = np.meshgrid(np.linspace(0, 20, N), np.linspace(0, 20, N))` to partition the space as done in the previous section, and then classifying each of these points. N should be large; in our solution, we use N = 81. Then at each of these points, draw a dot of the color of the classified class.

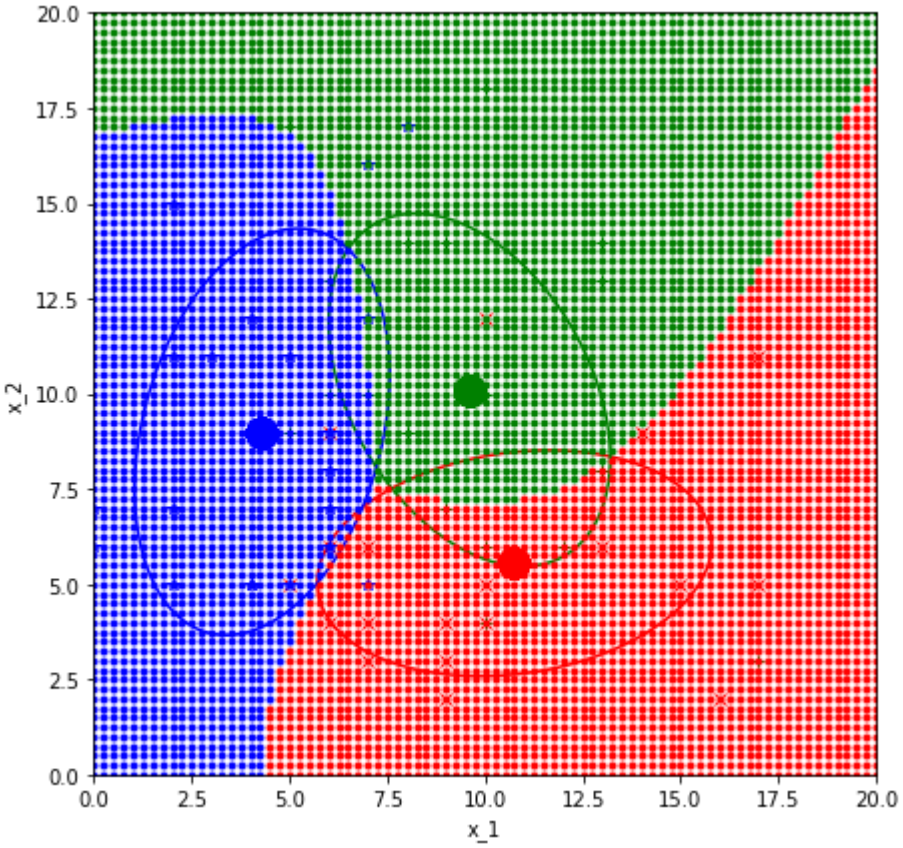
Hint 2: You can check that you've done this properly by verifying that the decision boundaries pass through the intersection points of the contours drawn in part (d).

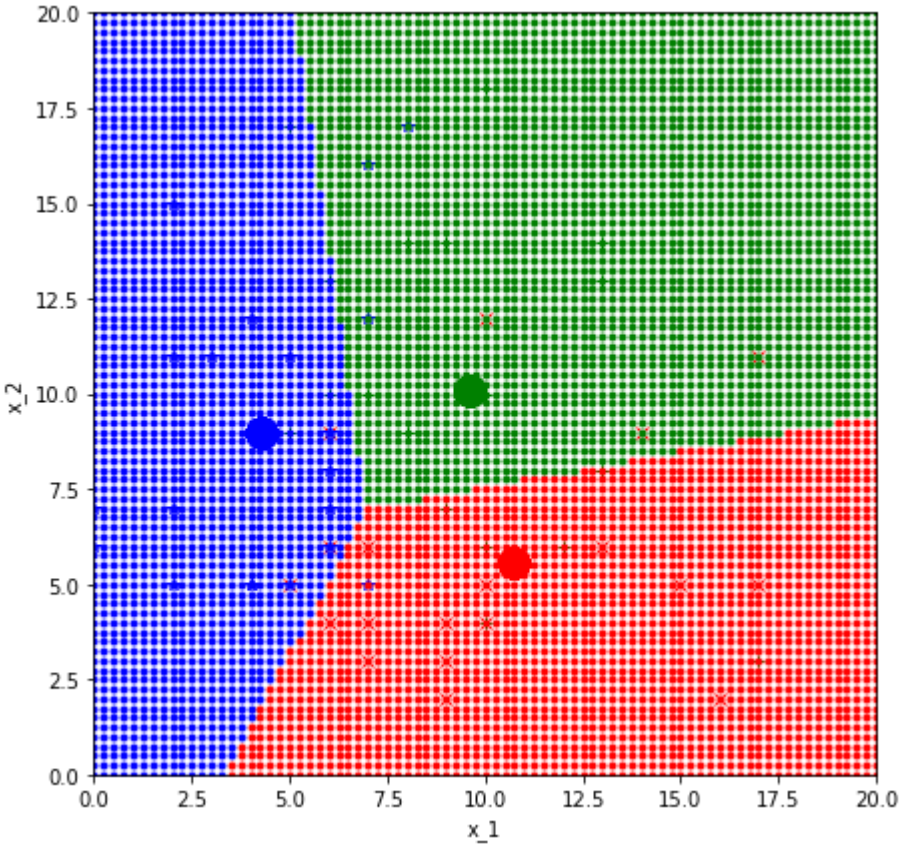
Hint 3: It's a good idea to do this one model at a time. You should get things working for model 1 for a smaller `N` value, so in code development, it doesn't take a long time to test your code. In the final result, your code will probably take some time to run because you're classifying each data point in a dense grid for each model.

In [ ]:
```
#e
#=====================================================#
# YOUR CODE HERE:
#   Plot the data points, their means, covariance ellipsoids,
#     and decision boundaries for each model.
#   Note that the naive Bayes Poisson model does not have an ellipsoid.
#   As in the above description, the decision boundary should be achieved
#     by densely classifying points in a grid.
#=====================================================#
colors2 = ['r','g','b']
modParam = [modParam1 , modParam2]
for modelIX in range(2):
    plt.figure(modelIX,figsize=(7,7))
    for classIX in range(NumClass):
        for dataIX in range(NumData):
            #plot the points and their means, just like before
            plt.plot(dataArr[classIX,dataIX,0],dataArr[classIX,dataIX,1],Marke
rPat[classIX])
            plt.plot(modParam[modelIX]['mean'][classIX,0],modParam[modelIX]['m
ean'][classIX,1],colors[classIX],markersize=30)
        plt.axis([0,20,0,20])
        plt.xlabel('x_1')
        plt.ylabel('x_2')
        MarkerCol=['r','g','b']

    #now begins plotting the elipse
    for classIX in range(NumClass):
        currMean=modParam[modelIX]['mean'][classIX ,:]
        if(modelIX == 0):
            currCov=modParam[modelIX]['cov']
        else:
            currCov=modParam[modelIX]['cov'][classIX]
        xl = np.linspace(0, 20, 201)
        yl = np.linspace(0, 20, 201)
        [X,Y] = np.meshgrid(xl,yl)

        Xlong = np.reshape(X-currMean[0],(np.prod(np.size(X))))
        Ylong = np.reshape(Y-currMean[1],(np.prod(np.size(X))))
        temp = np.row_stack([Xlong,Ylong])
        Zlong = []
        for i in range(np.size(Xlong)):
            Zlong.append(np.matmul(np.matmul(temp[:,i], np.linalg.inv(currCov
)), temp[:,i].T))
        Zlong = np.matrix(Zlong)
        Zlong = np.exp(-Zlong/2)/np.sqrt((2*np.pi)*(2*np.pi)*np.linalg.det(cur
rCov))
        Z = np.reshape(Zlong,X.shape)
        isoThr=0.007
        plt.contour(X,Y,Z,levels = isoThr,colors = colors2[classIX])
    # Begin plotting boundary
    N = 81
    [X,Y] = np.meshgrid(np.linspace(0, 20, N), np.linspace(0, 20, N))
    Z = np.zeros_like(X)
    #Classify Model
    for i in range(N):
        for j in range(N):
```

```
            vals = np.zeros(NumClass)
            x = np.array([X[i,j],Y[i,j]])
            for classix in range(NumClass):
                dem = x-modParam[modelIX]['mean'][classix]
                if modelIX==0:
                    vals[classix] = np.log(modParam[modelIX]['pi'][classix])-
0.5*dem@np.linalg.pinv(modParam[modelIX]['cov'])@dem - 0.5*np.log(np.linalg.de
t(modParam[modelIX]['cov']))
                else:
                    vals[classix] = np.log(modParam[modelIX]['pi'][classix])-
0.5*dem@np.linalg.pinv(modParam[modelIX]['cov'][classix])@dem - 0.5*np.log(np.
linalg.det(modParam[modelIX]['cov'][classix]))

            plt.plot(X[i,j],Y[i,j], marker='.', color = colors2[np.argmax(vals
)],linestyle='none')
plt.figure(2,figsize=(7,7))
for classIX in range(NumClass):
    for dataIX in range(NumData):
        #plot the points and their means, just like before
        plt.plot(dataArr[classIX,dataIX,0],dataArr[classIX,dataIX,1],MarkerPat
[classIX])
        plt.plot(modParam3['mean'][classIX,0],modParam3['mean'][classIX,1],col
ors[classIX],markersize=30)

    plt.axis([0,20,0,20])
    plt.xlabel('x_1')
    plt.ylabel('x_2')
    MarkerCol=['r','g','b']
# Begin plotting boundary
D=2
N = 81
[X,Y] = np.meshgrid(np.linspace(0, 20, N), np.linspace(0, 20, N))
Z = np.zeros_like(X)
#Classify Model
for i in range(N):
    for j in range(N):
        vals = np.zeros(NumClass)
        x = np.array([X[i,j],Y[i,j]])
        for classix in range(NumClass):
            vals[classix] = np.log(modParam3['pi'][classix]) +np.log(modParam3
['mean'][classix]).T@x - np.sum(modParam3['mean'][classix])- np.sum(np.log(sci
py.special.factorial(x)))

        plt.plot(X[i,j],Y[i,j], marker='.', color = colors2[np.argmax(vals)],l
inestyle='none')
#====================================================#
# END YOUR CODE
#====================================================#
```


20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
x_2
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
x_1


20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
x_2
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
x_1


20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
x_2
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
x_1