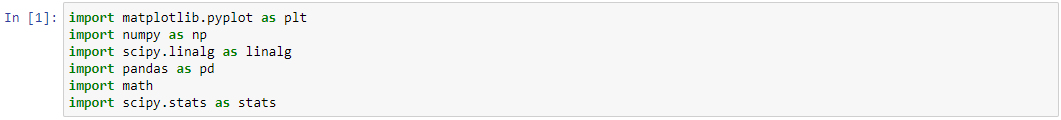
HW-1 Report

Berke Can Rizai, 69282

Generating Data

After doing the imports with the following cell,



We initialize random seed and start assigning the values that are required such as means, deviations and number of elements in each class.

Text, letter

Description automatically generated

Here, for example 0 and 2.5 is the means of the first class because each class have two features. In the deviations, first class has no covariance while second has positive and second has negative covariance. Class sizes are stored on an numpy array.

With the np.random.multivariate\_normal, we create normal functions that have some randomness, normal distribution have the properties we have given to it such as mean and deviation and points1, 2 and 3 are counted as class\_sizes[i]. 120, 80, 100 respectively.

Graphical user interface, text, application

Description automatically generated

Y contains the tags of each point such that [1,1,1...2,2,2,2,2,2...3,3,3,3,3,3,...3]. np.repeat just repeats the first parameter with second arguement times.

We plot the points with the matplotlib.pyplot’s plot function and we pass each points into it as well as style, color and size. We add labels to plot with xlabel and ylabel and show it.

Code and output is as follows; (in next page)

Text, scatter chart

Description automatically generatedChart, scatter chart

Description automatically generated

We calculate the parameters for the score function.

Mean is calculated as;

Graphical user interface, text, application

Description automatically generated

We pass axis=0 so that it calculates them “column-wise” and we don’t get mean of x and y for each points.

K is the number of classes.

I calculated covariance with the np.cov for each class.

Graphical user interface, text

Description automatically generated

Our priors are just number of each element in the each class divided by the total number of points. We simply do the calculation as follows;

Graphical user interface, text, application

Description automatically generated

ML Model

I used the following equation from the multivariate methods lecture,

Text

Description automatically generated with medium confidence

We have dimension of 2. We know the covariance matrix, pi, given vector x which consists of the features of a single point, mean vector and posteriors. I implemented the score function directly.

def score\_func(x,means,cova,count):

result = np.sum((-1\*np.log(2\*math.pi)) - (0.5\*np.log(abs(cova))) - 0.5\*(np.dot(np.array(x-means).T,2\*np.linalg.inv(cova))\*(x-means))) + np.log(count)

return result

First covariance is in absolute value because negative covarianced class-3 were causing issue as we can’t take log of a negative value. We have dot product of transpose of x-means vector with inverse of covariance matrix multiplied with the x-means vector and add the prior log.

In the getResult(x), we pass vector to this function and it returns the class that has the maximum likelihood this vector may be in. Note that vector is just point with two features.

Graphical user interface, text, application

Description automatically generated

Implementation is easy, we iterate over all the maximum values and points and we settle for the max.

Graphical user interface, text, application

Description automatically generated

Here, we first stack the points so that I can iterate over them easily later, I cast it to df so that i can use iterrows function. I append all the results to a list and I create “actual” that stores all the actual labels of the points. I cast it to numpy array to use in the confusion\_matrix.

Graphical user interface, application, Word

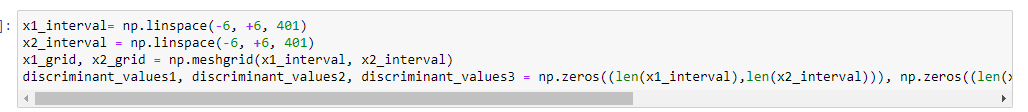
Description automatically generated

I pass the results I have found with result[:,0], this 0 is passed because the 1st col holds the points x, y values. They are compared with actual and we see the score. Here model found 119 reds out of 120, 77 greens out of 80 and 96 blues out of 100.

Chart

Description automatically generated

This just returns score values for given point and class.



Intervals are linearly spaced points that means between -6 and 6 there are 401 points. With meshgrid, we basically combine two intervals for each combination. In the discriminant values, we fill them with 0 as 2d matrix.

Text

Description automatically generated

Here, we iterate over each combination in the grid and fill the discriminant values for each corresponding entry. We will use this to draw discriminant lines.

Text

Description automatically generated

After making the figure a bit bigger, I calculated likelihood of each of the classes and choosed most likely one by comparing them with the other classes, both of them After a class is chosen as most likely, it is drawn with the color we predicted and with the if statement, I looked at the actual result in actual[k[2]] if they are different, I circle them with ko marker with size 12.

A picture containing diagram

Description automatically generated

Here, I remove the line parts that are not directly competing with each other as you will see in the last figure than with the contour as we used in class, I drawed the lines with grids with levels=0 and color black.

Chart, scatter chart

Description automatically generated

Finally, I wanted to show an area of doubt so that maybe we could see annotations that could be done wrongly due to randomness factor, here scores that are similar (s1-s2<4) are painted with dots. This does this,

Text

Description automatically generated

Again, points are labeled and drawn.

We finally get cool looking figure;

Chart, scatter chart

Description automatically generated