Engr421 Homework1 Report

Doğukan Soyuyüce

69331

1-We are asked to implement an algorithm which classifies three class of randomly produced data set.

2- As first step I defined the random data points using random.multivariate_normal from numpy library.

Defining means and covariances in order to produce random data points

In [3]:
```
np.random.seed(69331)
# I will produce my random data through using np.random.multivariate_normal function.
# I produced class_means and class_covariances in order to use in that function. I will have random data points
# in that given mean and covariances intervals

class_means = np.array([[+0.0, +2.5], [-2.5, -2.0], [+2.5, -2.0]])
class_covariances = np.array([[[+3.2, +0.0], [+0.0, +1.2]],
                              [[+1.2, -0.8], [-0.8, +1.2]],
                              [[+1.2, +0.8], [+0.8, +1.2]]])

# I assigned class sized 120 for class1 , 90 for class2 and 90 for class3 .
class_sizes = np.array([120, 90, 90])
```

In [17]:
```
# I generated my random data sets using class means and covariances with help pf random.multivariate_normal function.
points1 = np.random.multivariate_normal(class_means[0,:], class_covariances[0,:,:], class_sizes[0])
points2 = np.random.multivariate_normal(class_means[1,:], class_covariances[1,:,:], class_sizes[1])
points3 = np.random.multivariate_normal(class_means[2,:], class_covariances[2,:,:], class_sizes[2])
X = np.vstack((points1, points2, points3))

# We have corresponding labels for each data we have. I created my label set. 1 for class1 , 2 for class2, 3 for class3.
y = np.concatenate((np.repeat(1, class_sizes[0]), np.repeat(2, class_sizes[1]), np.repeat(3, class_sizes[2])))
```

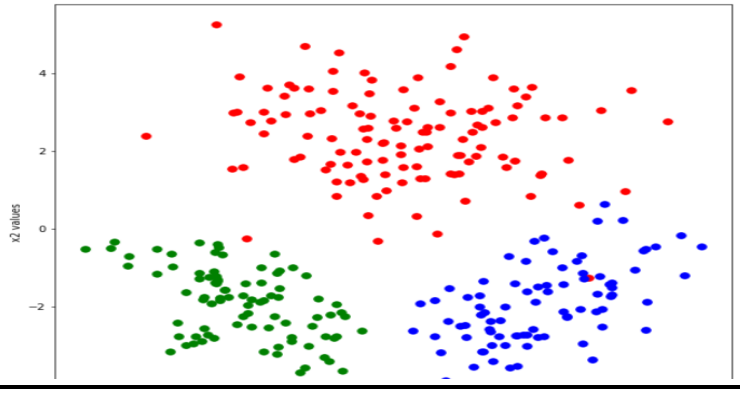
In [18]:
```
# I stacked my data and labels together and I had a dataset ready to use for training.
df = np.hstack((X, y[:, None]))
```

3- I visualized my data points using plt.plot.

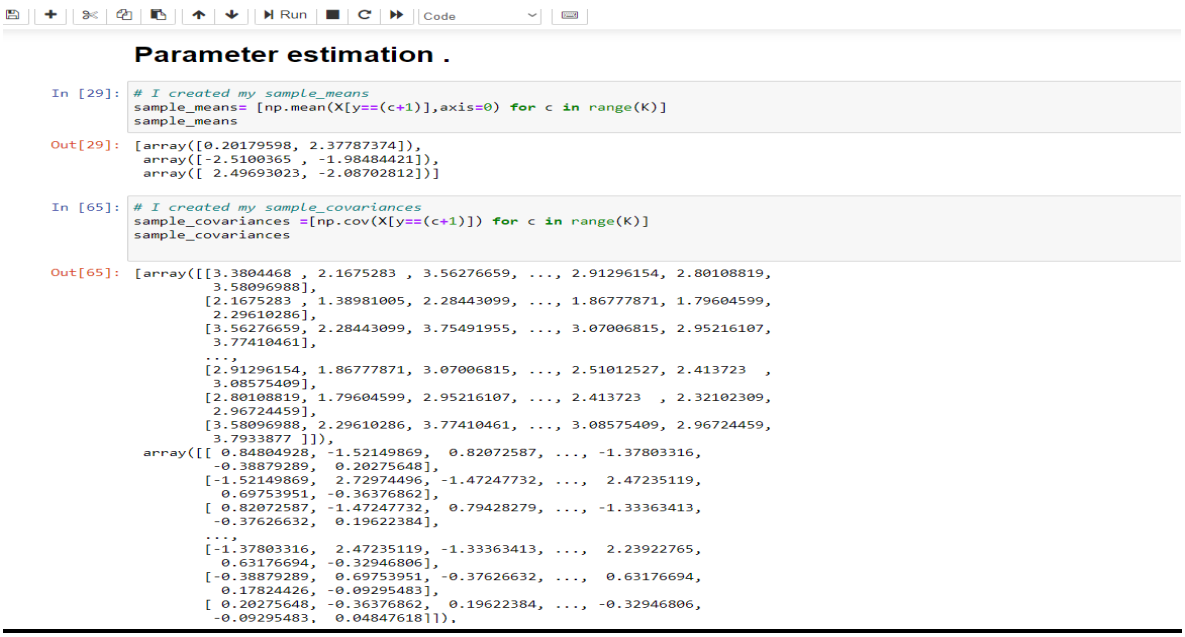
In [19]:
```
# I visualized my data in 3 different colors.
plt.figure(figsize = (10, 10))
plt.plot(points1[:,0], points1[:,1], "r.", markersize = 15)
plt.plot(points2[:,0], points2[:,1], "g.", markersize = 15)
plt.plot(points3[:,0], points3[:,1], "b.", markersize = 15)

# I labeled my x and y coordinates.
plt.xlabel("x1 values")
plt.ylabel("x2 values")
```

Out[19]: Text(0, 0.5, 'x2 values')



4- In parameter estimation part I defined my sample_mean , sample_covariances and class_priors.



Parameter estimation .

In [29]:
```
# I created my sample_means
sample_means= [np.mean(X[y==(c+1)],axis=0) for c in range(K)]
sample_means
```

Out[29]:
```
[array([0.20179598, 2.37787374]),
 array([-2.5100365 , -1.98484421]),
 array([ 2.49693023, -2.08702812])]
```

In [65]:
```
# I created my sample_covariances
sample_covariances =[np.cov(X[y==(c+1)]) for c in range(K)]
sample_covariances
```

Out[65]:
```
[array([[3.3804468 , 2.1675283 , 3.56276659, ..., 2.91296154, 2.80108819,
         3.58096988],
        [2.1675283 , 1.38981005, 2.28443099, ..., 1.86777871, 1.79604599,
         2.29610286],
        [3.56276659, 2.28443099, 3.75491955, ..., 3.07006815, 2.95216107,
         3.77410461],
        ...,
        [2.91296154, 1.86777871, 3.07006815, ..., 2.51012527, 2.413723  ,
         3.08575409],
        [2.80108819, 1.79604599, 2.95216107, ..., 2.413723  , 2.32102309,
         2.96724459],
        [3.58096988, 2.29610286, 3.77410461, ..., 3.08575409, 2.96724459,
         3.7933877 ]]),
 array([[ 0.84804928, -1.52149869,  0.82072587, ..., -1.37803316,
         -0.38879289,  0.20275648],
        [-1.52149869,  2.72974496, -1.47247732, ...,  2.47235119,
          0.69753951, -0.36376862],
        [ 0.82072587, -1.47247732,  0.79428279, ..., -1.33363413,
         -0.37626632,  0.19622384],
        ...,
        [-1.37803316,  2.47235119, -1.33363413, ...,  2.23922765,
          0.63176694, -0.32946806],
        [-0.38879289,  0.69753951, -0.37626632, ...,  0.63176694,
          0.17824426, -0.09295483],
        [ 0.20275648, -0.36376862,  0.19622384, ..., -0.32946806,
         -0.09295483,  0.04847618]]),
```

5-Through using softmax function I defined scores function , and I declared gradient functions in order to learn real values of weights (namely w and w0)

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

6- Objective function is plotted and observed that error is decreasing while iteration increases.

7- I calculated confusion matrix which demonstrates the accuracy of our algorithm.

8- I visualized the final result.

Visualize final result. using contourf function

```
[141]: # Visualizing the final result.
x1_interval = np.linspace(-8, +8, 1201)
x2_interval = np.linspace(-8, +8, 1201)
x1_grid, x2_grid = np.meshgrid(x1_interval, x2_interval)
discriminant_values = np.zeros((len(x1_interval), len(x2_interval), K))
for c in range(K):
    discriminant_values[:,:,c] = W[0, c] * x1_grid + W[1, c] * x2_grid + w0[0, c]

A = discriminant_values[:,:,0]
B = discriminant_values[:,:,1]
C = discriminant_values[:,:,2]
A[(A < B) & (A < C)] = np.nan
B[(B < A) & (B < C)] = np.nan
C[(C < A) & (C < B)] = np.nan
discriminant_values[:,:,0] = A
discriminant_values[:,:,1] = B
discriminant_values[:,:,2] = C

plt.figure(figsize = (10, 10))
plt.plot(X[y_truth == 1, 0], X[y_truth == 1, 1], "r.", markersize = 10)
plt.plot(X[y_truth == 2, 0], X[y_truth == 2, 1], "g.", markersize = 10)
plt.plot(X[y_truth == 3, 0], X[y_truth == 3, 1], "b.", markersize = 10)
plt.plot(X[y_predicted != y_truth, 0], X[y_predicted != y_truth, 1], "ko", markersize = 20, fillstyle = "none")

plt.contourf(x1_grid, x2_grid, A-B, levels = 0,colors="red",alpha=0.5)
plt.contourf(x1_grid, x2_grid, A - C, levels = 0, colors = "green",alpha=0.5)
plt.contourf(x1_grid, x2_grid, B - C, levels = 0, colors = "blue",alpha=0.5)


plt.xlabel("x1 values")
plt.ylabel("x2 values")
```

However , although I define the contour lines I encountered a problem like that. I could not colorize the inside of contours in a effective way.