Eren Barış Bostancı
68770

ENGR 421 – HW1

Data Generation:

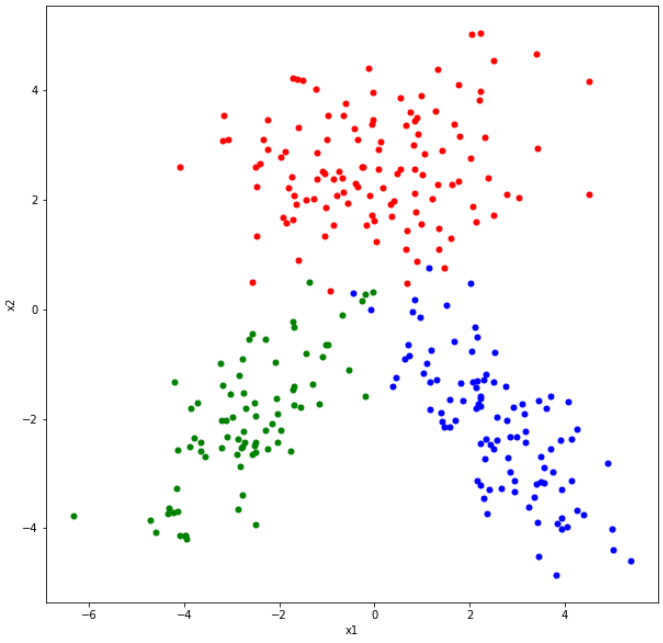
First, I used the given parameters and create 3 classes, which are points1, points2, points3. I followed the Lab examples for this part and generate the x, y data arrays.

```
[2]: np.random.seed(421)

     # mean parameters
     class_means = np.array([[+0.0, +2.5],
                             [-2.5, -2.0],
                             [+2.5, -2.0]])
     # covariance parameters
     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]]])
     # sample sizes
     class_sizes = np.array([120, 80, 100])
```

```
[3]: 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.concatenate((points1, points2, points3))

     # generate corresponding labels
     y = np.concatenate((np.repeat(1, class_sizes[0]), np.repeat(2, class_sizes[1]), np.repeat(3, class_sizes[2])))
```

Then, I plot the generated data to see if it is correct according to the given example plot.

```
[4]: plt.figure(figsize = (10, 10))
     plt.plot(points1[:,0], points1[:,1], "r.", markersize = 10)
     plt.plot(points2[:,0], points2[:,1], "g.", markersize = 10)
     plt.plot(points3[:,0], points3[:,1], "b.", markersize = 10)
     plt.xlabel("x1")
     plt.ylabel("x2")
     plt.show()
```



Eren Barış Bostancı
68770

Parameter Estimation:

I estimated the parameters, which are means, covariance and priors for all the three classes.

[8]:
```
K=3
sample_means = []
sample_covariances = []
class_priors = []

for i in range(K):
    sample_means.append(np.mean(x[y == (i + 1)], axis=0))
    sample_covariances.append(np.cov(np.transpose(x[y == i + 1])))
    class_priors.append(np.mean(y == (i + 1)))



print("sample_means\n")
print(sample_means)
print("\n")
print("sample_covariances\n")
print(sample_covariances)
print("\n")
print("class_priors\n")
print(class_priors)
```

```
sample_means

[array([0.04453807, 2.61225128]), array([-2.65871583, -2.04611631]), array([ 2.56054453, -2.12492713])]


sample_covariances

[array([[2.83985866, 0.22625045],
       [0.22625045, 1.01248436]]), array([[1.43826462, 1.02345485],
       [1.02345485, 1.37825958]]), array([[ 1.42107708, -1.08927412],
       [-1.08927412,  1.52276646]])]


class_priors

[0.4, 0.26666666666666666, 0.3333333333333333]
```

Confusion Matrix

After calculating sample means, covariances and priors, we can use them to find the score values.

$$W_c = -\frac{1}{2}\hat{\Sigma}_c^{-1}$$

$$w_c = \hat{\Sigma}_c^{-1} \cdot \hat{\mu}_c$$

$$w_{c0} = -\frac{1}{2}\hat{\mu}_c^T \cdot \hat{\Sigma}_c^{-1} \cdot \hat{\mu}_c - \frac{D}{2}\log(2\pi) - \frac{1}{2}\log(|\hat{\Sigma}_c|) + \log[\hat{P}(y=c)]$$

First, I calculate the Wc, wc, wc0 values as we did in the lecture.

[10]:
```
Wc = []
wc = []
wc0 = []
D = 2 #dimension

for i in range(K):
    cov_inv = np.linalg.inv(sample_covariances[i])
    Wc.append(np.array(cov_inv / -2))
    wc.append(np.matmul(cov_inv, sample_means[i]))
    wc0.append(np.array(-(np.matmul(np.matmul(np.transpose(sample_means[i]), cov_inv), sample_means[i])) * 0.5 - (D*np.log(2*math.pi)/2) - np.log(np.linalg.det(sample_covariances[i])) * 0.5 + np.log(class_priors[i])))

print("Wc")
print(Wc)
print("\n")
print("wc")
print(wc)
print("\n")
print("wc0")
print(wc0)
print("\n")

y_pred = []
for i in range(len(x)):
    scores = np.array([0, 0, 0])
    for j in range(K):
        score = np.matmul(np.matmul(np.transpose(x[i]), Wc[j]), x[i]) + np.matmul(np.transpose(wc[j]), x[i]) + wc0[j]
        scores[j] = score
    y_pred.append(scores)

y_pred = np.argmax(y_pred, axis=1) + 1
confusion_matrix = pd.crosstab(y_pred, y, rownames=['y_pred'], colnames=['y_truth'])
print(confusion_matrix)
```

```
Wc
[array([[-0.17925641,  0.04005676],
       [ 0.04005676, -0.5027859 ]]), array([[-0.73716165,  0.54739447],
       [ 0.54739447, -0.76925533]]), array([[-0.7789482 , -0.55720173],
       [-0.55720173, -0.72693053]])]


wc
[array([-0.19330918,  2.62323812]), array([-1.6797412 , -0.23723907]), array([ 1.62103693, -0.2358691 ])]


wc0
[array(-6.69524052), array(-5.60163075), array(-5.25105644)]


y_truth    1   2   3
y_pred
1        120   3   3
2          0  77   1
3          0   0  96
```

After finding scores, I put them in y_pred and then take their maximums and set y_pred accordingly and used it to print confusion matrix. For this run There is 7 mistakes.