1)

To apply Discrimination by Regression we are need to some updates on Linear discrimination.

a)

Instead of the softmax function, K sigmoid functions to generate $\hat{y}_{\{ic\}}$values.

$$S(x) = \frac{1}{\{1 + e^{\{-\alpha x\}}\}}$$

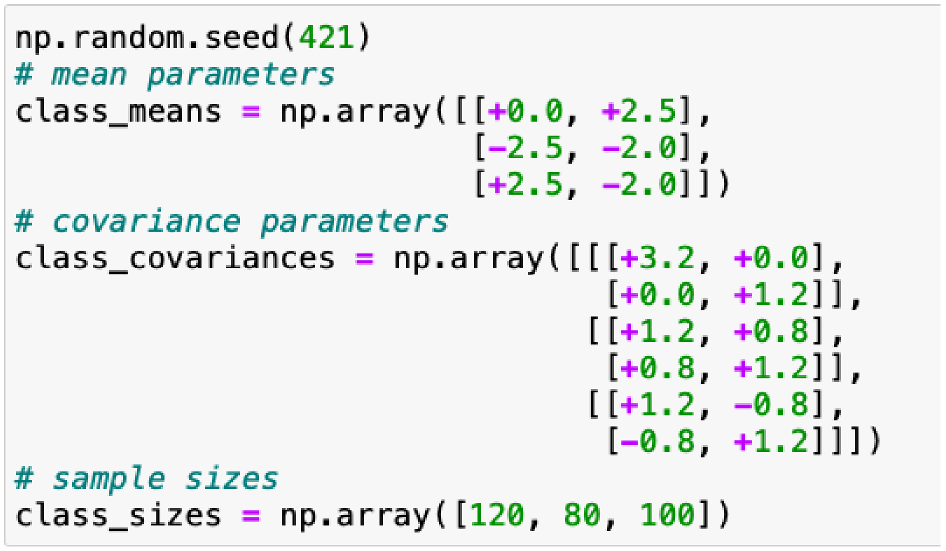
b)

Instead of the negative log-likelihood, use the sum squared errors as the error function to minimize

$$Error = 0.5 \sum_{i=1}^{N} \sum_{c=1}^{K} (y_{ic} - \hat{y}_{ic})^2$$

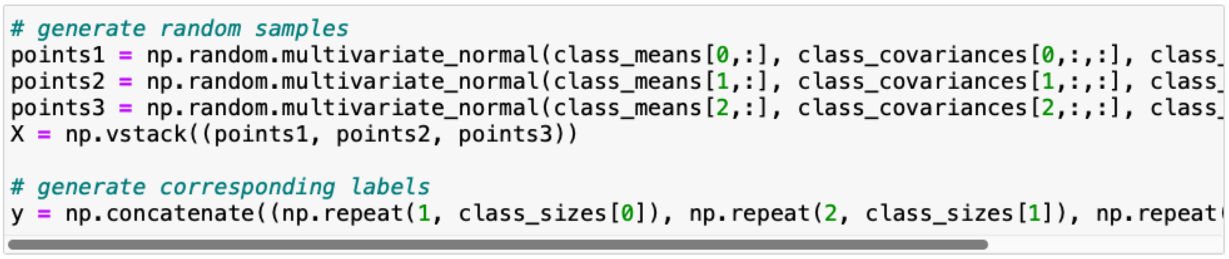
2)

Generate random data points from three bivariate Gaussian densities. Firstly, define class_means and class_covariances and class_sizes with given parameters.



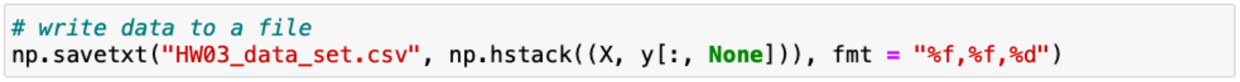
```
np.random.seed(421)
# mean parameters
class_means = np.array([[+0.0, +2.5],
                        [-2.5, -2.0],
                        [+2.5, -2.0]])
# covariance parameters
class_covariances = np.array([[[+3.2, +0.0],
                               [+0.0, +1.2]],
                              [[+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])
```

Then, created random 3 classes based on a multivariate normal distribution. random samples created and its corresponding labels created with given class_sizes.



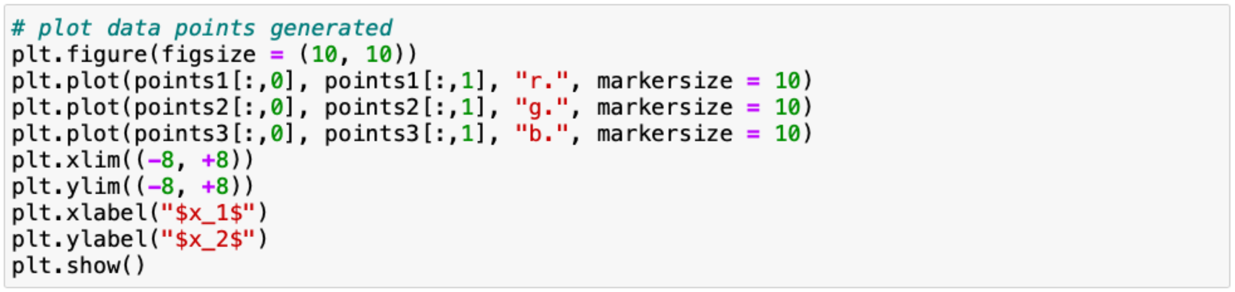
```
# generate random samples
points1 = np.random.multivariate_normal(class_means[0,:], class_covariances[0,:,:], class_
points2 = np.random.multivariate_normal(class_means[1,:], class_covariances[1,:,:], class_
points3 = np.random.multivariate_normal(class_means[2,:], class_covariances[2,:,:], class_
X = np.vstack((points1, points2, points3))

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

I saved these data point to the csv file to use later. This is not necessary in this Homework, I wanted to save them for later.

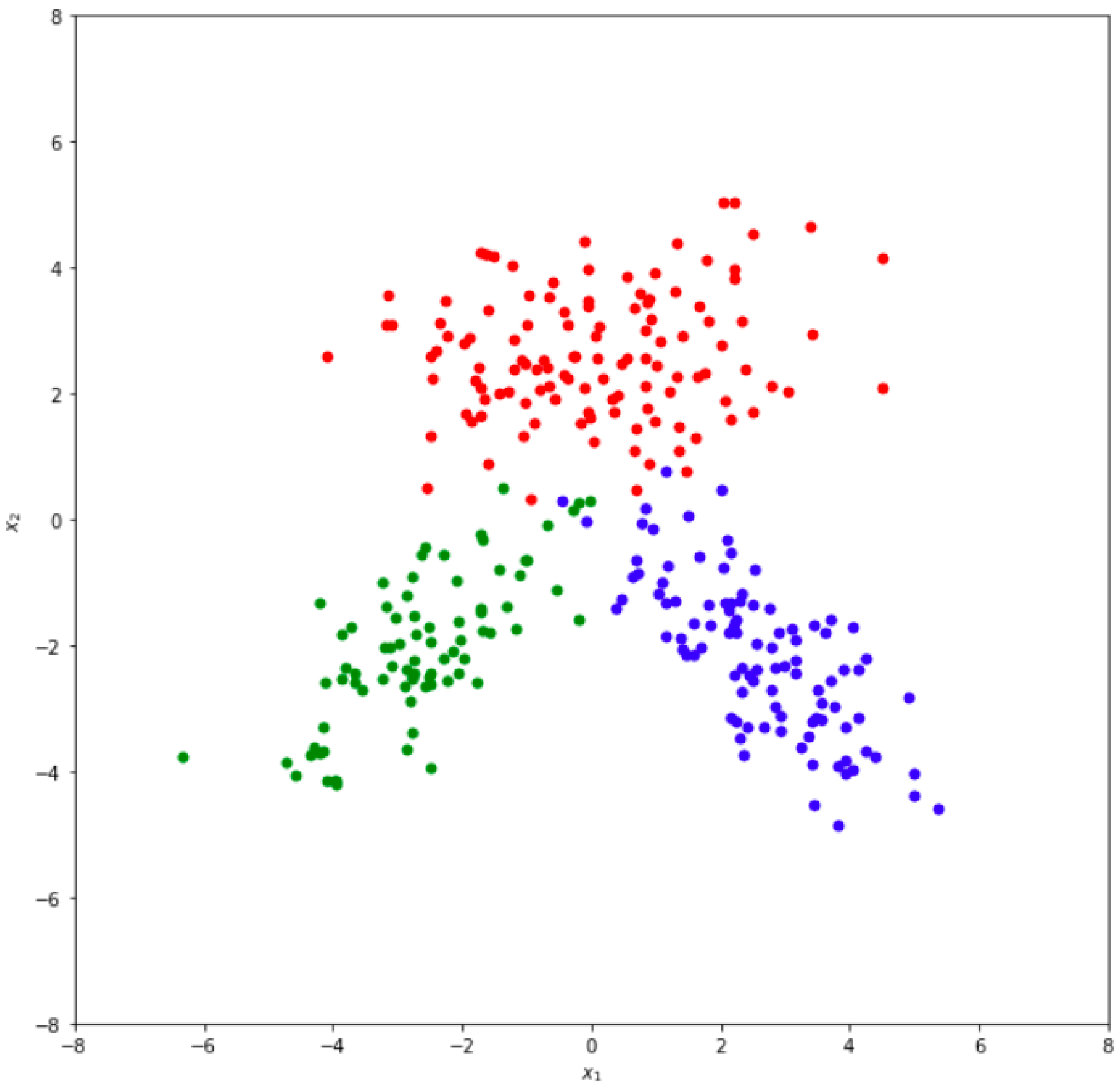


```
# write data to a file
np.savetxt("HW03_data_set.csv", np.hstack((X, y[:, None])), fmt = "%f,%f,%d")
```

Our data points should be checked to see how it looks like like. I used matplotlib.pyplot library top rint data points.

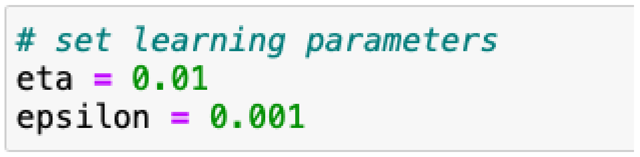


```
# plot data points generated
plt.figure(figsize = (10, 10))
plt.plot(points1[:,0], points1[:,1], "r.", markersize = 10)
plt.plot(points2[:,0], points2[:,1], "g.", markersize = 10)
plt.plot(points3[:,0], points3[:,1], "b.", markersize = 10)
plt.xlim((-8, +8))
plt.ylim((-8, +8))
plt.xlabel("$x_1$")
plt.ylabel("$x_2$")
plt.show()
```



3)

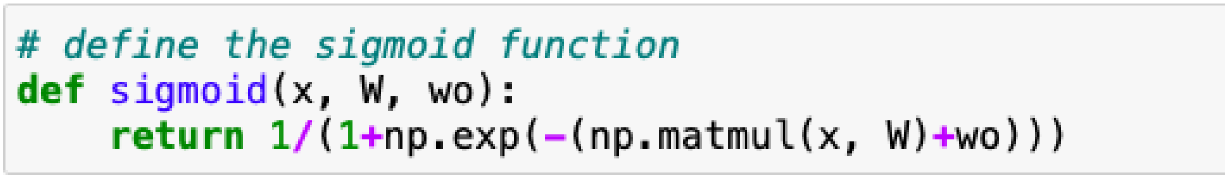
First defined learning parameters



```
# set learning parameters
eta = 0.01
epsilon = 0.001
```

Sigmoid function is defined to use in the parameter estimation

$$S(x) = \frac{1}{1 + e^{-\alpha x}}$$



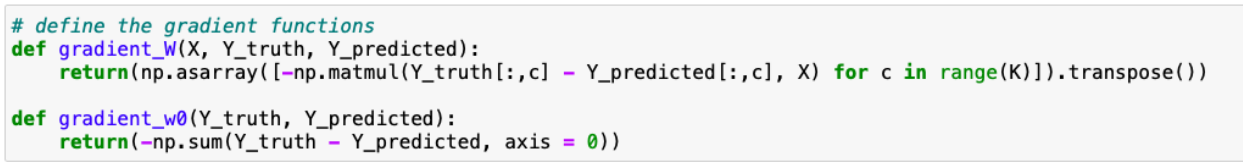
```
# define the sigmoid function
def sigmoid(x, W, wo):
    return 1/(1+np.exp(-(np.matmul(x, W)+wo)))
```

Derivative of Error is derived with respect to w and w0. By multiplying with (-)step size we are going to get gradients(which is steps).

$$\underbrace{\Delta x}_{\substack{\text{step}\\\text{(update)}}} = -\underbrace{\eta}_{\substack{\text{step}\\\text{size}}} \cdot \underbrace{\frac{\partial f(x)}{\partial x}}_{\text{derivative}}$$

$$\frac{\partial \text{Error}}{\partial \boldsymbol{w}_c} = -\sum_{i=1}^{N} (y_{ic} - \widehat{y}_{ic})\boldsymbol{x}_i$$

$$\frac{\partial \text{Error}}{\partial w_{c0}} = -\sum_{i=1}^{N} (y_{ic} - \widehat{y}_{ic})$$

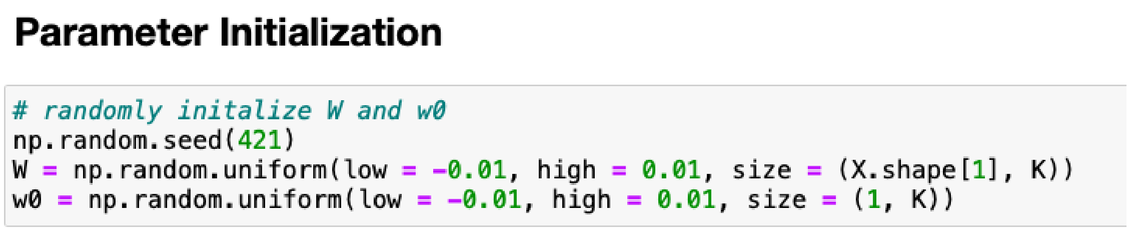


```
# define the gradient functions
def gradient_W(X, Y_truth, Y_predicted):
    return(np.asarray([-np.matmul(Y_truth[:,c] - Y_predicted[:,c], X) for c in range(K)]).transpose())

def gradient_w0(Y_truth, Y_predicted):
    return(-np.sum(Y_truth - Y_predicted, axis = 0))
```

Initialize parameter with small w,w0 (to get rid of memory issues.)

W has shape (N, K)

w0 has shape (1,K)



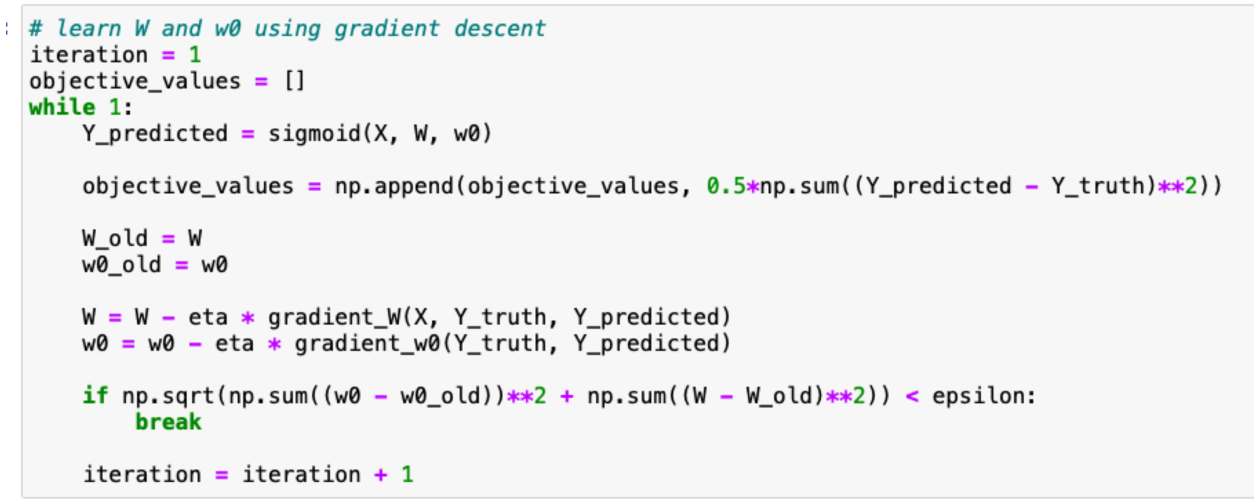
Parameter Initialization

```
# randomly initalize W and w0
np.random.seed(421)
W = np.random.uniform(low = -0.01, high = 0.01, size = (X.shape[1], K))
w0 = np.random.uniform(low = -0.01, high = 0.01, size = (1, K))
```

Now iterate over parameters to converge objective value. If improvement is less than epsilon, we stop. That means we found good enough parameter estimation. Here we could use iteration limit, however epsilon does same thing.

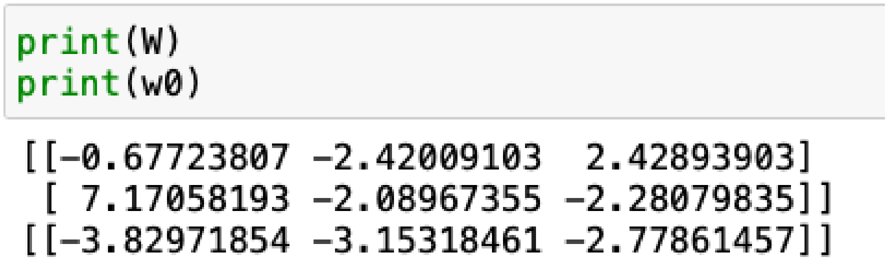
for y_predicted I used sigmoid function instead of softmax for discrimination by regression.

$$\text{Error} = 0.5 \sum_{i=1}^{N} \sum_{c=1}^{K} (y_{ic} - \hat{y}_{ic})^2$$



```
# learn W and w0 using gradient descent
iteration = 1
objective_values = []
while 1:
    Y_predicted = sigmoid(X, W, w0)

    objective_values = np.append(objective_values, 0.5*np.sum((Y_predicted - Y_truth)**2))

    W_old = W
    w0_old = w0

    W = W - eta * gradient_W(X, Y_truth, Y_predicted)
    w0 = w0 - eta * gradient_w0(Y_truth, Y_predicted)

    if np.sqrt(np.sum((w0 - w0_old))**2 + np.sum((W - W_old)**2)) < epsilon:
        break

    iteration = iteration + 1
```

When we look at the parameter estimates, we finally get such a result.



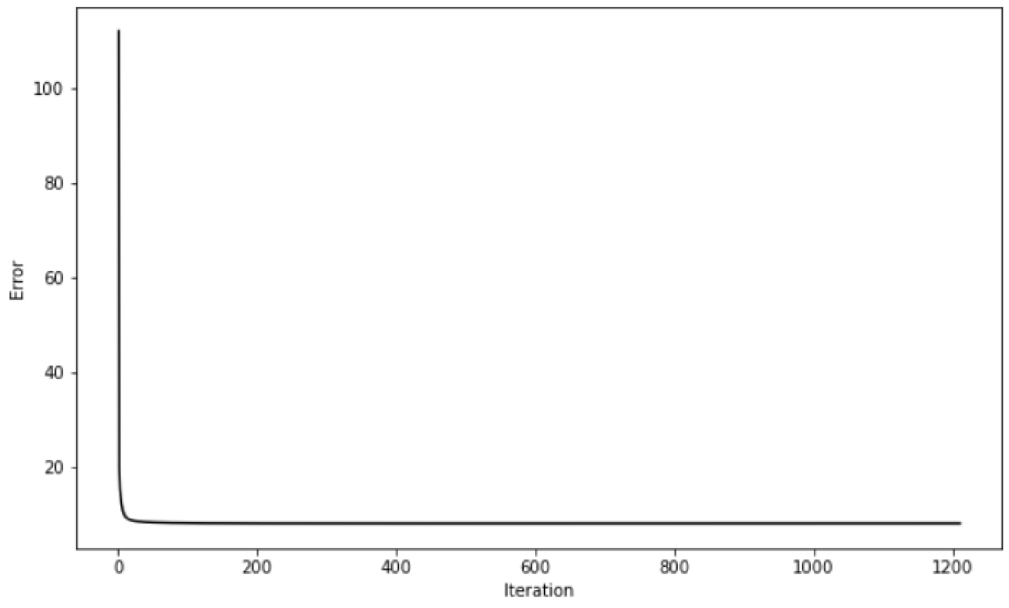
```
print(W)
print(w0)
```

```
[[-0.67723807 -2.42009103  2.42893903]
 [ 7.17058193 -2.08967355 -2.28079835]]
[[-3.82971854 -3.15318461 -2.77861457]]
```

4)

Here objective function values throughout iterations. As we can see after some iterations improvement is very limited. We could limit our iterations, if particular error is enough for us.

```
# plot objective function during iterations
plt.figure(figsize = (10, 6))
plt.plot(range(1, iteration + 1), objective_values, "k-")
plt.xlabel("Iteration")
plt.ylabel("Error")
plt.show()
```



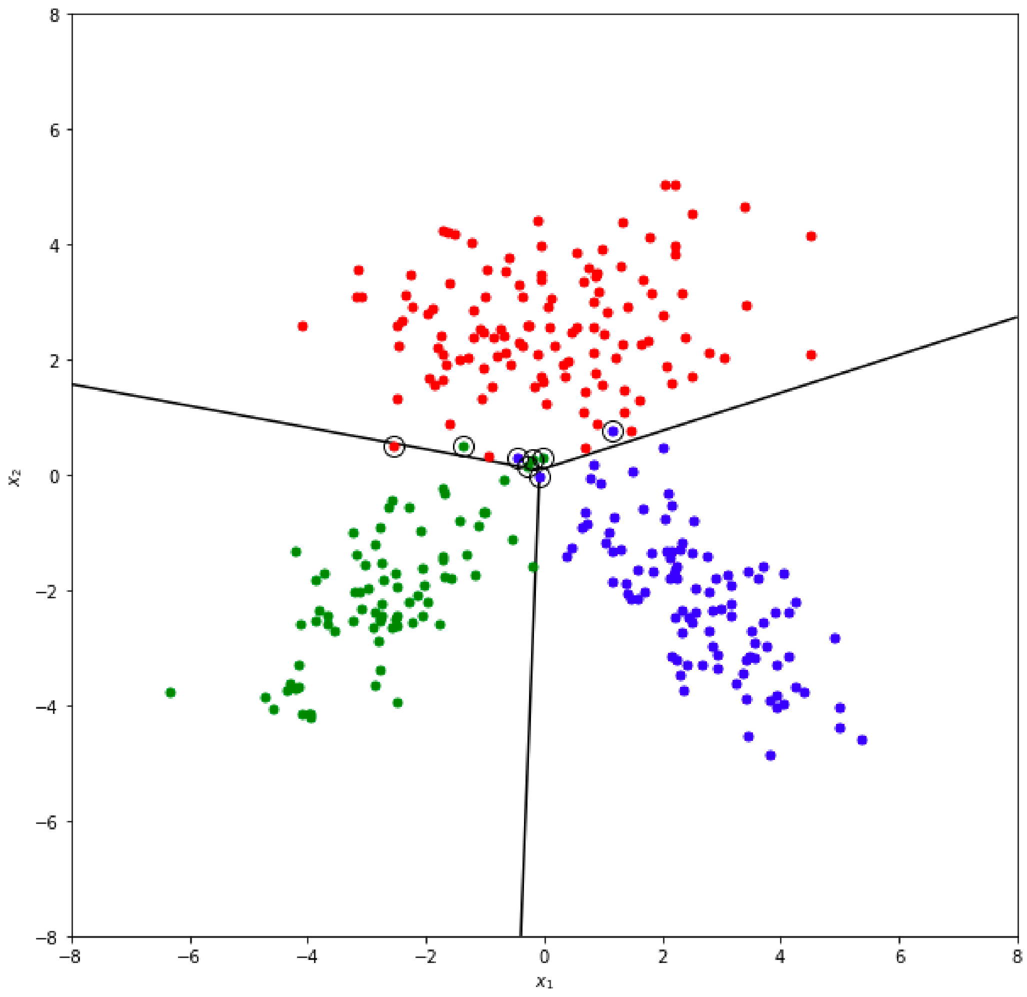
5)

Here confusion matrix for the data points which is developed using parameters from step 3

```
# calculate confusion matrix
y_predicted = np.argmax(Y_predicted, axis = 1) + 1
confusion_matrix = pd.crosstab(y_predicted, y_truth, rownames = ['y_pred'], colnames = ['y_truth'])
print(confusion_matrix)
```

```
y_truth    1   2   3
y_pred
1        119   4   2
2          1  76   1
3          0   0  97
```

Draw decision boundaries which is calculated from estimated parameters. Then marked misclassified data points. When I don’t assign nan values to out pf area, it gets unwanted lines. By using discriminant values, plotted contour graph lines


x_1
x_2
−8
−6
−4
−2
0
2
4
6
8