ENGR-421 HW-3 REPORT

Name-Surname: Barış KAPLAN

Initially, I have imported the necessary libraries. These libraries are as follows:

import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import math

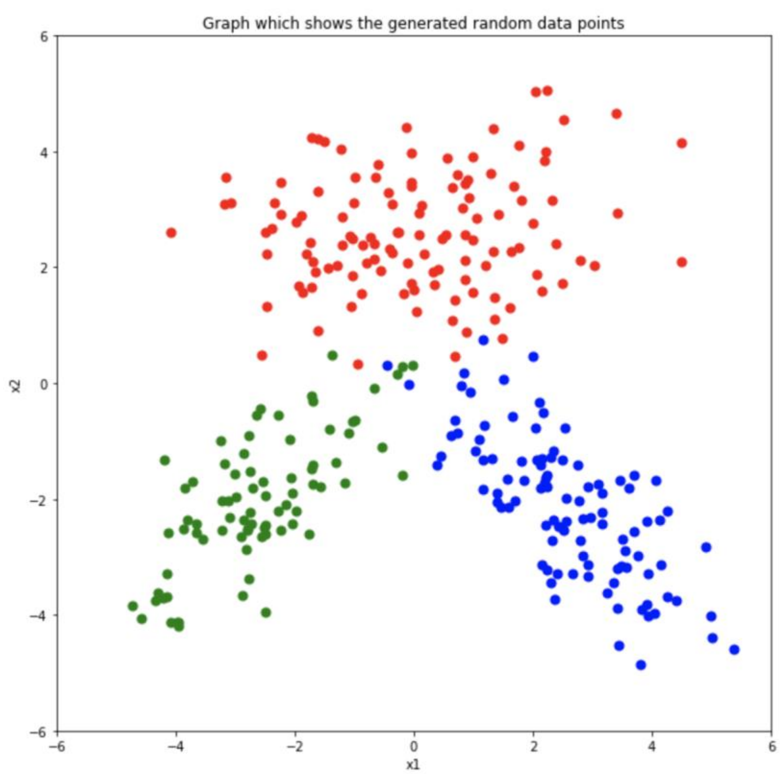
Then, I have generated the mean, covariance matrix, and size parameters.

After that, by using the **np.random.multivariate_normal** function of the numpy library, I have generated the random data points.

By using the **np.vstack** function of numpy library, I have vertically stacked the created random data points.

After that, I have generated the class labels by utilizing the **np.concatenate** function of the numpy library.

Next, by using the **plt.plot** function, I have generated the plot of the created random data points. By utilizing **plt.xlim** and **plt.ylim** functions, I have determined the limits of the x values and the limits of the y values in the plot. You can see the generated plot in Figure 1.



le writing the sigmoid

Figure 1: The plot containing the created random data points

$$\frac{1}{1+\exp\left[-\left[w^T . x + w_0\right]\right]}$$

sigmoid function

Figure 2: The sigmoid function formula I have used

Subsequently, I have defined the gradient functions for W and w0. While defining the gradient functions, I have utilized the formula in Figure 3 for W and the formula in Figure 4 for w0.

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

Figure 3: The gradient function for W

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

Figure 4: The gradient function for w0

By using Figure 5 and Figure 6, I have found the gradients in Figure 3 & Figure 4.

$$\hat{y} = \begin{bmatrix} w_1 & w_2 & \cdots & w_D \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_D \end{bmatrix} + w_0$$

$$= \begin{bmatrix} w_0 & w_1 & w_2 & \cdots & w_D \end{bmatrix} \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ \vdots \\ x_D \end{bmatrix} = \underset{1 \times (D+1)}{W^T} . \underset{(D+1) \times 1}{X}$$

Figure 5: Predicted y (y hat)

$\text{Error}_i(W|x_i, y_i) = \ldots$ squared error

$$= \frac{1}{2}\left[y_i - s(w^T . x_i)\right]^2$$

$$= \frac{1}{2}\left[y_i - w^T . x_i\right]^2$$

$$\frac{\partial \text{Error}_i}{\partial w} = \cancel{\frac{1}{2}} . \cancel{2} . \left[y_i - w^T . x_i\right] \frac{\partial \left[y_i - w^T . x_i\right]}{\partial w}$$

$$= \underbrace{\left[y_i - w^T . x_i\right]}_{1 \times 1} \underbrace{(-x_i)}_{(D+1) \times 1} = -(y_i - \hat{y}_i) . x_i$$

Figure 6: The derivation of gradients

In this step, by using the np.random.uniform() function of numpy library, I have randomly initialized W and w0.

Then, I have applied the gradient descent algorithm to W and w0. In this algorithm, I have generated the predicted values by using the sigmoid function (sigmoid_calculation_function). Furthermore, to update W and w0 values, I have subtracted the update amount delta W from W and delta w0 from w0 (you can see the update amounts for W and w0 in Figure 7).

$$\Delta w = -\eta \cdot \frac{\partial Error}{\partial w} = -\eta \cdot \left[-\sum_{i=1}^{N} (y_i - \hat{y}_i) \cdot x_i \right]$$

$$= \eta \cdot \sum_{i=1}^{N} (y_i - \hat{y}_i) \cdot x_i$$

$$\Delta w_0 = -\eta \cdot \frac{\partial Error}{\partial w} = \eta \cdot \sum_{i=1}^{N} (y_i - \hat{y}_i)$$

Figure 7: The update amount formulas for W and w0

In the gradient descent algorithm, I have updated W and w0 until the condition in Figure 8 is met. When the condition in Figure 8 is met, I have terminated the algorithm.

As the error function, I have utilized the sum squared errors in the gradient descent algorithm. I have utilized the formula in Figure 9 for obtaining the sum squared errors. You can see the learned values of W and w0 (values after applying the gradient descent algorithm) in Figure 10.

```
if np.sqrt(np.sum((w0Val - w0_old_Val))**2 + np.sum((WVal - W_old_Val)**2)) < epsVal:
```

Figure 8: The break condition of the gradient descent algorithm

After that, I have plotted the "Iteration vs Error" fu ... ration vs Error" function (see Figure 11) that the gra ... ed in ... ab function of the pa ... fusion matrix. We can see from the confusion matrix (see Figure 12) that there is 1 misclassified data

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

Figure 9: The formula for the sum squared errors

```
W:

[[-0.67723844 -2.42009122  2.42893886]
 [ 7.17058383 -2.08967379 -2.28079817]]

w0:

[[-3.82971889 -3.15318481 -2.77861437]]
```

Figure 10: The values of W and w0 after the gradient descent algorithm finishes

point for class 1 (having the red color) , there are 4 misclassified data points for class 2 (having the green color), and there are 3 misclassified data points for class 3 (having the blue color). In total, there are 8 (1+4+3) misclassified data points. You can see the decision boundaries between the classes and the misclassified data points from the Figure 13. While drawing the decision boundaries, I have used plt.contour function of the matplotlib.pyplot library.

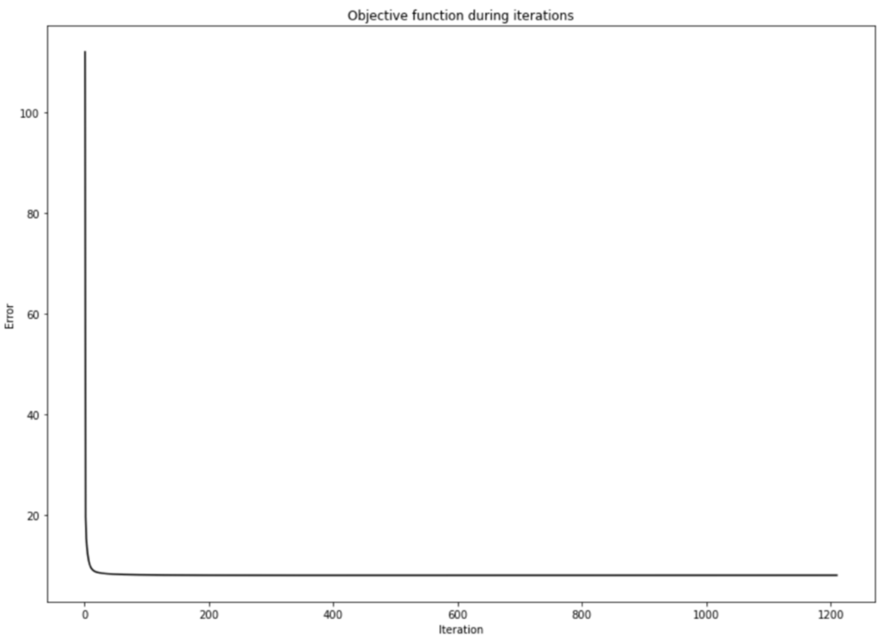


Figure 11: The "Iteration vs Error" graph

```
The Confusion Matrix:

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

Figure 12: The confusion matrix for the data points in my training set

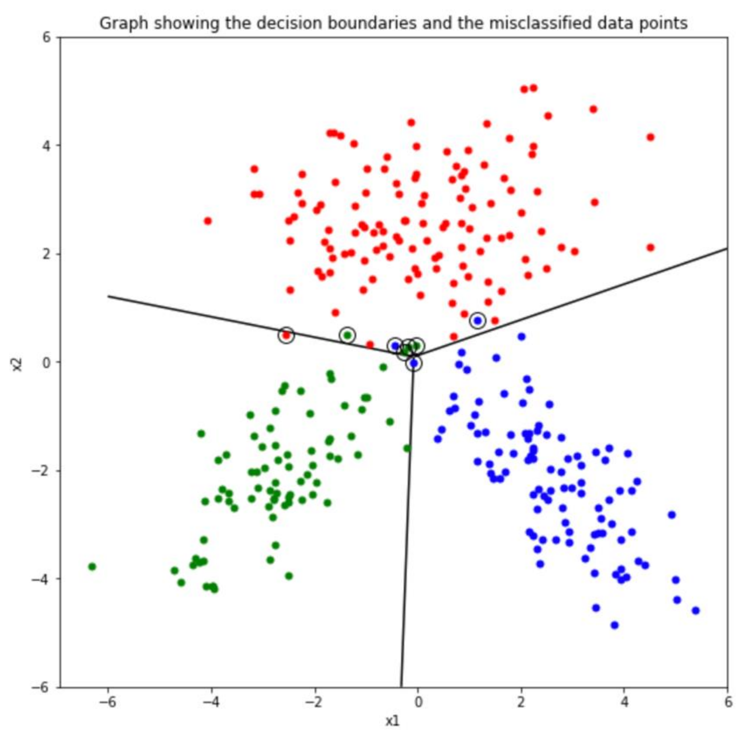


Figure 13: The plot showing the decision boundaries and the misclassified data points