Homework 03 – Discrimination by Regression

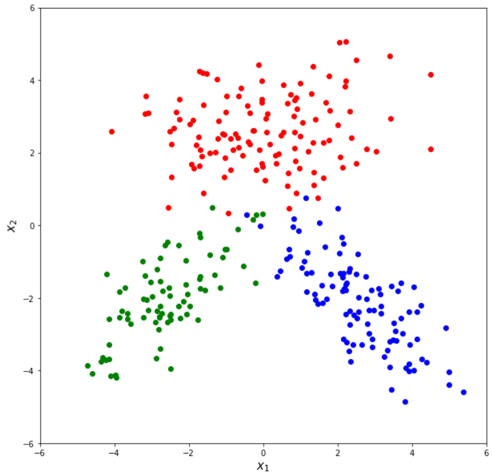
Generating Data

- After importing NumPy and Pandas libraries, I've generated random data points with the parameters provided in the guide.

```
np.random.seed(421)

data_1 = np.random.multivariate_normal(mean=np.array([+0.0, +2.5]),
                                       cov=np.array([[+3.2, +0.0],
                                                     [+0.0, +1.2]]), size=120)
data_2 = np.random.multivariate_normal(mean=np.array([-2.5, -2.0]),
                                       cov=np.array([[+1.2, +0.8],
                                                     [+0.8, +1.2]]), size=80)
data_3 = np.random.multivariate_normal(mean=np.array([+2.5, -2.0]),
                                       cov=np.array([[+1.2, -0.8],
                                                     [-0.8, +1.2]]), size=100)

X = np.vstack([data_1, data_2, data_3]) # data matrix
y_truth = np.concatenate([np.repeat(1, data_1.shape[0]),
                          np.repeat(2, data_2.shape[0]),
                          np.repeat(3, data_3.shape[0])])

data_set = np.hstack((X, y_truth[:, None]))
```

Plotting the Data

- I have plotted the data to make sure that it looks similar to the figure in the guide.



Number of classes, number of samples, and one-of-K encoding

- To use later, I have created a "K" variable for the number of classes, a "N" variable for the number of samples, and a "Y_truth" array by using one-of-K encoding.

```
# K: number of classes
K = np.max(y_truth)

# N: number of samples
N = data_set.shape[0]

# one-of-K encoding
Y_truth = np.zeros((N, K)).astype(int
Y_truth[range(N), y_truth - 1] = 1

print(f"K, N: {(K, N)}\n")

print("Y_truth (one-of-K encoding):")
print(Y_truth)
```

```
K, N: (3, 300)

Y_truth (one-of-K encoding):
[[1 0 0]
 [1 0 0]
 [1 0 0]
 [1 0 0]
 [1 0 0]
 [1 0 0]
```

Sigmoid Function

- Then, I defined the sigmoid function using the following formula:

$$y = \hat{P}(C_1|\mathbf{x}) = \text{sigmoid}(\boldsymbol{w}^T\boldsymbol{x} + w_0) = \frac{1}{1+\exp[-(\mathbf{w}^T\mathbf{x}+w_0)]}$$

```python
def sigmoid(X, W, w0):
    return 1 / (1 + np.exp(-(X@W) + np.repeat(w0, X.shape[0], axis=0)))
```

Gradient Functions

- I have used the following gradient functions from Lab 04.

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

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

```python
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))
```

Eta (step size/learning factor) and Epsilon

- I have defined eta and epsilon parameters as given in the guide.

```python
eta = 0.01
epsilon = 0.001
```

Parameter Initialization

- I have initialized W and w_0 by using NumPy's random.uniform function.

```python
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))
```

Iterative Algorithm and Update Equations

- In the iteration, I have used the sum of squared errors, as indicated in the guide, and the following update equations from "10.7.1 Two Classes" section of the book.

The sum of squared errors:

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

$$\Delta w_j = \eta \sum_t (r^t - y^t)x_j^t, j = 1, \ldots, d$$

$$\Delta w_0 = -\eta \sum_t (r^t - y^t)$$

```python
iteration = 1
objective_values = []

while 1:
    print(f"iteration #{iteration}")
    Y_predicted = sigmoid(X, W, w0)

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

- After 1210 iterations, it has stopped by breaking out of the while loop.

Parameter Estimations

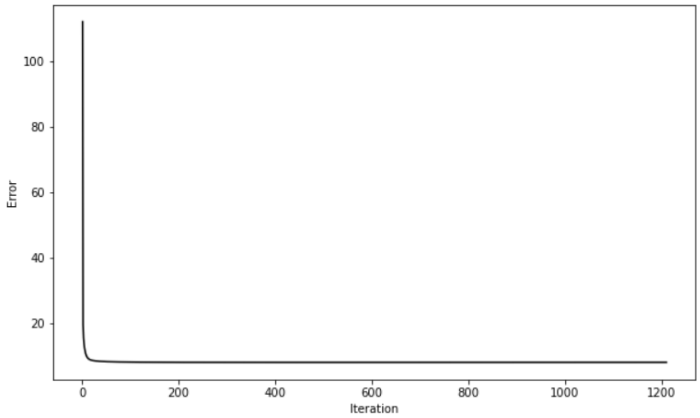
- Here are the parameter estimations after the iteration stopped:

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

```
[[-0.67723968 -2.42009122  2.42893886]
 [ 7.17059648 -2.08967379 -2.28079817]]
[[3.82972553 3.15318481 2.77861437]]
```

Convergence

- The plot of objective values throughout iterations is shown in the following figure:



Confusion Matrix

- Then, I have calculated the confusion matrix by using Pandas's crosstab function.

```
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
```

Drawing Decision Boundaries

- Lastly, I have plotted the data with the decision boundaries calculated by the discrimination by regression algorithm.