ENGR 421 / DASC 521: Introduction to Machine Learning
Homework 02: Discrimination by Regression

First of all I read the section 10.8 (3rd edition) from the textbook for Discrimination by Regression . As the error function, I used the sum of squares error function.

$$\boldsymbol{r}^t = \boldsymbol{y}^t + \boldsymbol{\epsilon}$$

where $\boldsymbol{\epsilon} \sim \mathcal{N}_K(0, \sigma^2 \mathbf{I}_K)$. Assuming a linear model for each class, we have

$$y_i^t = \text{sigmoid}(\boldsymbol{w}_i^T \boldsymbol{x}^t + w_{i0}) = \frac{1}{1 + \exp[-(\boldsymbol{w}_i^T \boldsymbol{x}^t + w_{i0})]}$$

Then the sample likelihood is

$$l(\{\boldsymbol{w}_i, w_{i0}\}_i | \mathcal{X}) = \prod_t \frac{1}{(2\pi)^{K/2} |\Sigma|^{1/2}} \exp\left[-\frac{\|\boldsymbol{r}^t - \boldsymbol{y}^t\|^2}{2\sigma^2}\right]$$

and the error function is

$$E(\{\boldsymbol{w}_i, w_{i0}\}_i | \mathcal{X}) = \frac{1}{2} \sum_t \|\boldsymbol{r}^t - \boldsymbol{y}^t\|^2 = \frac{1}{2} \sum_t \sum_i (r_i^t - y_i^t)^2$$

Our dataset consists of 1000 rows and 784 columns, each column represents a pixel.

Then to train the data, I divided the first 500 rows into the train set and the last 500 rows as the test set.

```
#Train set
X_train=X[0:500,:]
Y_train=Y_truth[0:500,:]

#Test set
X_test=X[0:-500,:]
Y_test=Y_truth[0:-500,:]
```

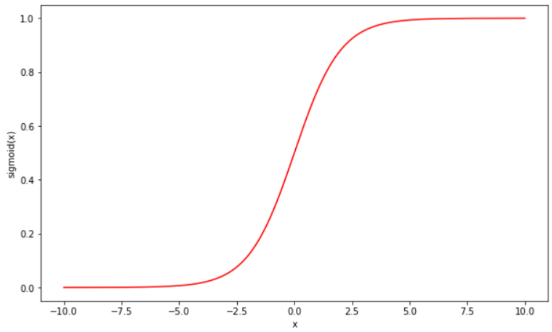
There were 2 different classes in the book, but there were 5 different classes in our homework, so I converted the target variable (y) to 0-1 format with one hot encoding.

I learned that I should use sigmoid function in multi class classification problems instead of softmax.

Learn a discrimination by regression algorithm using the sigmoid function for this multiclass classification problem. You can use the following learning parameters. ¶

$$\text{sigmoid}(\boldsymbol{w}^\top \boldsymbol{x} + w_0) = \frac{1}{1 + \exp\left[-(\boldsymbol{w}^\top \boldsymbol{x} + w_0)\right]}$$

```
# define the sigmoid function
def sigmoid(X, w, w0):
    return(1 / (1 + np.exp(-(np.matmul(X, w) + w0))))

draw_seq = np.linspace(-10, +10, 2001)
plt.figure(figsize = (10, 6))
plt.plot(draw_seq, 1 / (1 + np.exp(-draw_seq)), "r-")
plt.xlabel("x")
plt.ylabel("sigmoid(x)")
plt.show()
```



Gradient Function

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

$$\frac{\partial \text{Error}}{\partial w_0} = -\sum_{i=1}^{N}(y_i - \widehat{y}_i)$$

```
# define the gradient functions
def gradient_W(X, Y_truth, Y_predicted):
    return(np.asarray([-np.sum(np.repeat((Y_truth[:,c] - Y_predicted[:,c])[:, None], X.shape[1], axis = 1) * X, axis = 0) for c i

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

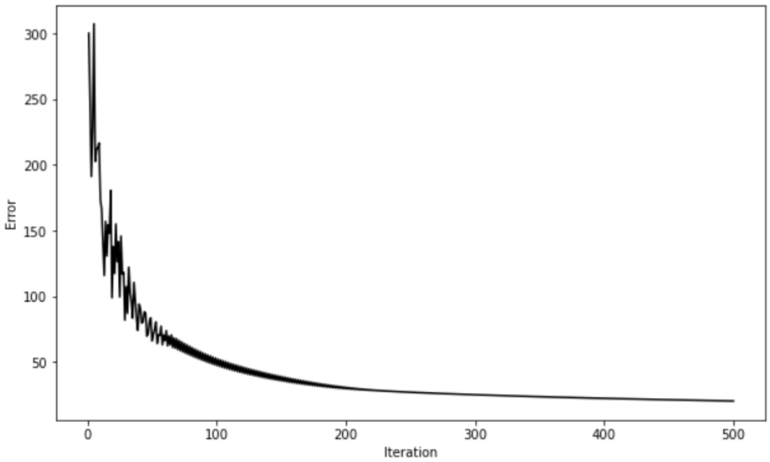
After defined gradient functions I set the the parametes of algortihm as:

- eta = 0.0001
- epsilon = 1e-3
- max iteration = 500

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

    objective_values = np.append(objective_values,np.sum((Y_train-Y_predicted)**2)/2)
    #-np.sqrt(((Y_train - Y_predicted) ** 2).mean()))
   #-np.sum(((Y_train)**2-(Y_predicted))**2))
    W_old = W
    w0_old = w0

    W = W - eta * gradient_W(X_train, Y_train, Y_predicted)
    w0 = w0 - eta * gradient_w0(Y_train, Y_predicted)

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

    iteration = iteration + 1
print(W)
print(w0)
```

```
[[-0.02489191 -0.04240257 -0.14855656 -0.14236667  0.05037148]
 [-0.03031641 -0.02866678 -0.13963906 -0.15083246  0.04803366]
 [-0.04381186 -0.03482827 -0.13850118 -0.14562813  0.05454492]
 ...
 [-0.1007952  -0.03446409 -0.08644267 -0.13277171  0.05842846]
 [-0.03624593 -0.04564407 -0.12643439 -0.13703142  0.04298337]
 [-0.03391512 -0.04229446 -0.1444833  -0.14224122  0.05587851]]
[-0.0369684  -0.04199999 -0.1355629  -0.1542288   0.05443006]
```

Objective Function During Iterations:



Train Confusion Matrix:

```
# calculate confusion matrix
y_train=y_truth[0:500]
y_predicted = np.argmax(Y_predicted, axis = 1) + 1
confusion_matrix_train = pd.crosstab(y_predicted, y_train, rownames = ['y_pred'], colnames = ['y_train_truth'])
print(confusion_matrix_train)
```

```
y_train_truth    1   2    3    4   5
y_pred
1              106   0    0    0   0
2                0  85    0    0   0
3                1   1  112    0   0
4                0   0    0  100   0
5                0   0    0    0  95
```

Test Confusion Matrix:

```
# calculate confusion matrix
y_test=y_truth[500:1000]
y_predicted = np.argmax(Y_predicted, axis = 1) + 1
confusion_matrix_test = pd.crosstab(y_predicted, y_test, rownames = ['y_pred'], colnames = ['y_test_truth'])
print(confusion_matrix_test)
```

```
y_test_truth   1   2   3    4   5
y_pred
1             80   5   5    0   3
2              0  82   0    0   0
3             12   4  97    0   3
4              0   0   0  103   1
5              7   0   3    0  95
```