20145271 Geum Kang Hyeon

Digit Classifying Using MNIST Dateset

Build a binary classifier based on k random features for each digit against all the other digits at MNIST dataset

Let $x = (x_1, x_2, \dots, x_m)$be a vector representing an image in the dataset.

The prediction function $f_w(x)$ is defined by the linear combination of data $(1, x)$ and the model parameter w: $f_d(x; w) = w_0 * 1 + w_1 * g_1 + w_2 * g_2 + \dots + w_k * g_k$ where $w = (w_0, w_1, \dots, w_m)$

where w = (w_0, w_1, ... , w_k) and the basis function g_k is defined by the inner product of random vector r_k and input vector x.

You may want to try to use g_k = max(inner production(r_k, x), 0) to see if it improves the performance.

The prediction function $f_d(x; w)$ should have the following values:

$f_d(x; w)$ = +1 if label(x) is d,

$f_d(x; w)$ = -1 if label(x) is not d

The optimal model parameter w is obtained by minimizing the following objective function: $\sum_i (f_w(x^i - y^i)^2)$

and the label of input x is given by : $argmax_d f_d(x; w)$

1. Declare required variables

References to Assignment 03 code

In [24]:

```
import matplotlib.pyplot as plt
import numpy as np
import random

train_file_data = "mnist_train.csv"
test_file_data = "mnist_test.csv"

handle_file = open(train_file_data, "r")
data = handle_file.readlines()
handle_file.close()

handle_file = open(test_file_data, "r")
data_Test = handle_file.readlines()
handle_file.close()

size_row = 28
size_col = 28

num_image = len(data)
num_image_Test = len(data_Test)

count = 0
count_Test = 0

list_image  = np.empty((num_image, size_row * size_col), dtype=float)
list_label  = np.empty(num_image, dtype=int)

list_image_Test  = np.empty((num_image_Test, size_row * size_col), dtype=float)
list_label_Test  = np.empty(num_image_Test, dtype=int)
```

2. Read Data

In [25]:

```
# Read Train Data
for line in data:
    line_data   = line.split(',')
    label       = line_data[0]
    im_vector   = np.asfarray(line_data[1:])

    list_label[count] = label
    list_image[count] = im_vector
    count += 1

# Read Test Data
for line in data_Test:
    line_data   = line.split(',')
    label       = line_data[0]
    im_vector   = np.asfarray(line_data[1:])

    list_label_Test[count_Test] = label
    list_image_Test[count_Test] = im_vector
    count_Test += 1
```

3. Multiply Random Vectoy

g_k is defined by the inner product of random vector r_k and input vector x.

g_k = max(inner production(r_k, x), 0)

In [26]:

```
random_vector = np.zeros((size_row*size_col, size_row*size_col), dtype=float)

for i in range(size_row*size_col) :
    for j in range(size_row*size_col) :
        random_vector[i][j] = random.gauss(0, 1)

list_image = np.matmul(list_image, random_vector)
list_image_Test = np.matmul(list_image_Test, random_vector)

for i in range(count) :
    for j in range(size_row*size_col) :
        if(list_image[i][j] < 0):
            list_image[i][j] = 0

for i in range(count_Test) :
    for j in range(size_row*size_col):
        if(list_image_Test[i][j] < 0):
            list_image_Test[i][j] = 0
```

4. Align Data A anb b in formula $Ax = b$ for each digit

In [27]:

```
A = np.zeros((count, size_row*size_col), dtype=float)
b = np.zeros((10, count), dtype=float)
b_Test = np.zeros((10, count), dtype=float)

A = list_image
for idx in range(0, 10):
    for i in range(count):
        if list_label[i] == idx:
            b[idx][i] = 1
        else:
            b[idx][i] = -1

    for i in range(count_Test):
        if list_label_Test[i] == idx:
            b_Test[idx][i] = 1
        else:
            b_Test[idx][i] = -1
```

5. Calculate x for each digit

$$x = (A^T A)^{-1} A^T b$$

$$x = A^+ b$$

In [29]:

```
transpose_A = np.transpose(A)

# Calculate Pseudo Inverse
step1 = np.matmul(transpose_A, A)
step2 = np.linalg.pinv(step1)
step3 = np.matmul(step2, transpose_A)

# x = (Pseudo Inverse of A) * b
result = np.zeros((10,len(step3)), dtype=float)

for i in range(0,10):
    result[i] = np.matmul(step3, b[i])
```

6. Compute TP, FP, TN, FN using Train Dataset

TP, TN are Answer, FP, FN are Wrong Answer

Put the actual data in the expression Ax = b and compare it with the answer in the train dataset

In [31]:

```
TP, TN, FP, FN = (0, 0, 0, 0)
for i in range(count):
    value = np.zeros(10, dtype=float)
    for idx in range(0,10):
        value[idx] = np.matmul(list_image[i], result[idx])
    idx = np.argmax(value)
    if value[idx] >= 0 and b[idx][i] == 1:
        TP += 1
    elif value[idx] < 0 and b[idx][i] == -1:
        TN += 1
    elif value[idx] >= 0 and b[idx][i] == -1:
        FP += 1
    elif value[idx] < 0 and b[idx][i] == 1:
        FN += 1
```

7. Compute TP, FP, TN, FN using Test Dataset

x uses x obtained through the train dataset

Model from train dataset can be verified

In [32]:

```
TP_Test, TN_Test, FP_Test, FN_Test = (0, 0, 0, 0)

for i in range(count_Test):
    value = np.zeros(10, dtype=float)
    for idx in range(0,10):
        value[idx] = np.matmul(list_image_Test[i], result[idx])
    idx = np.argmax(value)
    if value[idx] >= 0 and b_Test[idx][i] == 1:
        TP_Test += 1
    elif value[idx] < 0 and b_Test[idx][i] == -1:
        TN_Test += 1
    elif value[idx] >= 0 and b_Test[idx][i] == -1:
        FP_Test += 1
    elif value[idx] < 0 and b_Test[idx][i] == 1:
        FN_Test += 1
```

8. Result

Error Rate = (True Negative + False Negative) / Total Count * 100

In [38]:

```
print('[Train Data Set]')
print('Total Data Count : ' + str(count) + '₩n')
print('TRUE POSITIVE RATE: ' + str(TP) + ' (' + str("%0.1f" % (TP / count * 100)) + '%)')
print('ERROR RATE: ' + str(TN + FN) + ' (' + str("%0.1f" % ((TN+FN) / count * 100)) + '%)')

print('₩n₩n[Test Data Set]')
print('Total Data Count : ' + str(count_Test) + '₩n')
print('TRUE POSITIVE RATE : ' + str(TP_Test) + ' (' + str("%0.1f" % (TP_Test / count_Test * 100
)) + '%)')
print('ERROR RATE : ' + str(TN_Test+FN_Test) + ' (' + str("%0.1f" % ((TN_Test+FN_Test) / count_T
est * 100)) + '%)')
```

```
[Train Data Set]
Total Data Count : 60000

TRUE POSITIVE RATE: 51370 (85.6%)
ERROR RATE: 7478 (12.5%)


[Test Data Set]
Total Data Count : 10000

TRUE POSITIVE RATE : 8635 (86.4%)
ERROR RATE : 1174 (11.7%)
```

Train Data	True	False
Positive	51370 (85.6%)	1152 (1.9%)
Negative	2549 (4.2%)	4929 (8.2%)

Test Data	True	False
Positive	8635 (86.4%)	191 (0.3%)
Negative	475 (0.8%)	699 (1.2%)