Assignment09

May 30, 2019

20132915 Nam, Geun Woo

1 Assignment09

Build a binary classifier to classify digit 0 against all the other digits at MNIST dataset.

Let $x = (x_1, x_2, ..., x_m)$ 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 : $f_w(x) = w_0 * 1 + w_1 * x_1 + w_2 * x_2 + ... + w_m * x_m$ where $w = (w_0, w_1, ..., w_m)$

The prediction function $f_w(x)$ should have the following values: $f_w(x) = +1$ if $label(x) = 0$ $f_w(x) = -1$ if $label(x)$ is not 0

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

1. Compute an optimal model parameter using the training dataset
2. Compute (1) True Positive, (2) False Positive, (3) True Negative, (4) False Negative based on the computed optimal model parameter using (1) training dataset and (2) testing dataset.

```
In [1]: import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
```

```
In [2]: #
        # normalize the values of the input data to be [0, 1]
        #
        def normalize(data):

            data_normalized = (data - min(data)) / (max(data) - min(data))

            return(data_normalized)
```

```
In [121]: def distance_L2(a, b):
              distance = (a-b)**2
              return distance
```

```
In [3]: # Import MNIST dataset
        # mnist_train.csv, mnist_test.csv
```

```
training_file_data = "mnist_train.csv"
handle_file = open(training_file_data, "r")
training_data = handle_file.readlines()
handle_file.close()

test_file_data = "mnist_test.csv"
handle_file = open(test_file_data, "r")
test_data = handle_file.readlines()
handle_file.close()

size_row = 28
size_col = 28
training_data_num = len(training_data)
test_data_num = len(test_data)
```

In [50]:
```
#
# make a matrix each column of which represents an images in a vector form
#
list_image_training  = np.empty((size_row * size_col+1, training_data_num),\
                                dtype=float)
list_label_training  = np.empty(training_data_num, dtype=int)

for count, line in enumerate(training_data):

    line_data   = line.split(',')
    label       = line_data[0]
    im_vector   = np.asfarray(line_data[1:])
    im_vector   = normalize(im_vector)

    # list_label[] : label
    list_label_training[count]       = label
    # list_image, append 1 to the front of the array
    list_image_training[:, count]    = np.insert(im_vector, 0, 1)


list_image_test  = np.empty((size_row * size_col+1, test_data_num), dtype=float)
list_label_test  = np.empty(test_data_num, dtype=int)

for count, line in enumerate(test_data):

    line_data   = line.split(',')
    label       = line_data[0]
    im_vector   = np.asfarray(line_data[1:])
    im_vector   = normalize(im_vector)

    # list_label[] : label
    list_label_test[count]       = label
    # list_image, append 1 to the front of the array
```

```
            list_image_test[:, count]    = np.insert(im_vector, 0, 1)
```

$xw = y$
x = image
w = model parameter
y = predicted label
$x^{(i)}$ = actual label
$y^{(i)}$ = predicted label 1. Compute the above prediction fuction. 2. Label by the smallest distance.

In [225]:
```
# Make a prediction function
# xw = y
actual_label_training = np.where((list_label_training==0), +1, -1).\
    reshape((training_data_num,1))
x_training = np.copy(list_image_training).transpose()
w = np.matmul(np.linalg.pinv(x_training), actual_label_training)
```

In [189]:
```
# Calculating predicted labels

predicted_label_training = np.matmul(x_training, w)
for i in range(0, training_data_num):
    distance_zero = distance_L2(predicted_label_training[i,0], +1)
    distance_non_zero = distance_L2(predicted_label_training[i,0], -1)
    if distance_zero >= distance_non_zero:
        predicted_label_training[i,0] = -1
    else:
        predicted_label_training[i,0] = +1
```

In [200]:
```
# Calculate and show TP, FP, TN, FN
# TP: actual: positive && predicted: positive
# FN: actual: positive && predicted: negative
# FP: actual: negative && predicted: positive
# TN: actual: negative && predicted: negative
def calculate_confusion_matrix(actual_label, predicted_label, label_num):
    TP = np.zeros((label_num))
    FP = np.zeros((label_num))
    TN = np.zeros((label_num))
    FN = np.zeros((label_num))
    for i in range(0, label_num):
        if(actual_label[i,0] == +1 and predicted_label[i,0] == +1):
            TP[i] = 1
        elif(actual_label[i,0] == +1 and predicted_label[i,0] == -1):
            FN[i] = 1
        elif(actual_label[i,0] == -1 and predicted_label[i,0] == +1):
            FP[i] = 1
        elif(actual_label[i,0] == -1 and predicted_label[i,0] == -1):
            TN[i] = 1
    # precision
    # recall
    # F1 score
```

```
            # ...
            print("TP: %d" % np.count_nonzero(TP==1))
            print("FN: %d" % np.count_nonzero(FN==1))
            print("FP: %d" % np.count_nonzero(FP==1))
            print("TN: %d" % np.count_nonzero(TN==1))
            print("Sum: %d " % np.sum(TP+FN+FP+TN))
```

In [223]:
```
print("Training data")
calculate_confusion_matrix(actual_label_training, predicted_label_training, training
```

```
Training data
TP: 5167
FN: 756
FP: 179
TN: 53898
Sum: 60000
```

In [218]:
```
# Make a prediction function
# xw = y
actual_label_test = np.where((list_label_test==0), +1, -1).reshape((test_data_num,1))
x_test = np.copy(list_image_test).transpose()
#w = np.matmul(np.linalg.pinv(x_test), actual_label_test)
```

In [219]:
```
# Calculating predicted labels

predicted_label_test = np.matmul(x_test, w)
for i in range(0, test_data_num):
    distance_zero = distance_L2(predicted_label_test[i,0], +1)
    distance_non_zero = distance_L2(predicted_label_test[i,0], -1)
    if distance_zero >= distance_non_zero:
        predicted_label_test[i,0] = -1
    else:
        predicted_label_test[i,0] = +1
```

In [224]:
```
print("Test data")
calculate_confusion_matrix(actual_label_test, predicted_label_test, test_data_num)
```

```
Test data
TP: 866
FN: 114
FP: 43
TN: 8977
Sum: 10000
```