Assignment11

June 1, 2019

20132915 Nam, Geun Woo

1 Assignment11

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

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

The prediction function $f_d(x; w)$ is defined by the linear combination of input vector x and the model parameter w for each digit d :

$f_d(x; w) = w_0 * 1 + w_1 * g_1 + w_2 * g_2 + ... + w_k * g_k$

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.

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

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

The optimal model parameter w is obtained by minimizing the following objective function for each digit d : $\sum_i (f_d(x^{(i)}; w) - y^{(i)})^2$

and the label of input x is given by:

$argmax_d f_d(x; w)$

1. Compute an optimal model parameter using the training dataset for each classifier $f_d(x, w)$
2. Compute (1) true positive rate, (2) error rate 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 [3]:
```
def distance_L2(a, b):
    distance = (a-b)**2
    return distance
```

In [4]:
```
# 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 [16]:
```
#
# 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)
```

Feature expansion
$xw = y$
x = image
w_i = model parameter for classifying where an image is label i
y = predicted label
$x^{(i)}$ = actual label
$y^{(i)}$ = predicted label 1. Compute the above prediction fuction. 2. Label by the smallest distance.

In [17]:
```
k = 1500
r = np.random.randn(k, 1+size_row*size_col)
```

In [44]:
```
list_image_training_expanded = np.inner(r, list_image_training.transpose())
list_image_training_expanded=list_image_training_expanded.transpose()
list_image_training_expanded = np.maximum(list_image_training_expanded, 0)
```

In [33]:
```
# Make a prediction function
# xw = y
w = np.empty((k, 1, 10))
for i in range(0,10):
    actual_label_training = np.where((list_label_training==i), +1, -1).\
    reshape((training_data_num,1))
    x_training = np.copy(list_image_training_expanded)
    np.copyto(w[:,:,i], np.matmul(np.linalg.pinv(x_training.transpose()), actual_label
```

In [38]:
```
# Calculating predicted labels

predicted_label_training = np.empty((training_data_num, 1))
l = np.empty((10))
for i in range(0, training_data_num):
    for j in range(0,10):
        l[j] = np.matmul(x_training.transpose()[i,:], w[:,:,j])
    predicted_label_training[i] = np.argmax(l)
```

In [39]:
```
# Calculate and show true positive rate, error rate
def calculate_confusion_matrix(actual_label, predicted_label, label_num):
    # confusion_matrix = np.zeros((10,10))

    TP_list = np.zeros((label_num, 1))
    FP_list = np.zeros((label_num, 1))
    TN_list = np.zeros((label_num, 1))
    FN_list = np.zeros((label_num, 1))

    for i in range(0, label_num):
        if(actual_label[i,0] == predicted_label[i,0]):
```

```
                TP_list[i] = 1
            else:
                FP_list[i] = 1

        TP_num = np.count_nonzero(TP_list)
        FP_num = np.count_nonzero(FP_list)

        assert( (TP_num + FP_num) == label_num )

        true_positive_rate = TP_num / label_num
        error_rate = FP_num / label_num

        print("TP: %f" % TP_num)
        print("FP: %f" % FP_num)
        print("True Positive rate: %f" % true_positive_rate)
        print("Error rate: %f" % error_rate)
```

In [40]:
```
print("Training data")
calculate_confusion_matrix\
(list_label_training.reshape((training_data_num,1)),\
 predicted_label_training, training_data_num)
```

```
Training data
TP: 57451.000000
FP: 2549.000000
True Positive rate: 0.957517
Error rate: 0.042483
```

In [45]:
```
list_image_test_expanded = np.inner(r, list_image_test.transpose())
list_image_test_expanded = list_image_test_expanded.transpose()
list_image_test_expanded = np.maximum(list_image_test_expanded, 0)
```

In [48]:
```
# Make a prediction function
# xw = y
w = np.empty((k, 1, 10))
for i in range(0,10):
    actual_label_test = np.where((list_label_test==i), +1, -1).\
    reshape((test_data_num,1))
    x_test = np.copy(list_image_test_expanded).transpose()
    np.copyto(w[:,:,i], np.matmul(np.linalg.pinv(x_test.transpose()), actual_label_tes
```

In [50]:
```
# Calculating predicted labels

predicted_label_test = np.empty((test_data_num, 1))
l = np.empty((10))
for i in range(0, test_data_num):
    for j in range(0, 10):
```

```
            l[j] = np.matmul(x_test.transpose()[i,:], w[:,:,j])
        predicted_label_test[i] = np.argmax(l)
```

```
In [51]: print("Test data")
         calculate_confusion_matrix\
         (list_label_test.reshape((test_data_num,1)),\
          predicted_label_test, test_data_num)
```

```
Test data
TP: 9841.000000
FP: 159.000000
True Positive rate: 0.984100
Error rate: 0.015900
```