Assignment10_20133096_HyunjaeLee

June 6, 2019

0.1 20133096 Hyunjae Lee

Build a binary classifier 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 data (1, x) and the model parameter w for each digit d : f_d(x; w) = 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_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.

0.2 1. Import modules

In [11]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

0.3 2. Read data

In [12]:
```
training_file = "mnist_train.csv"
handle_file = open(training_file, "r")
training_data = handle_file.readlines()
handle_file.close()

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

row = 28
col = 28
```

```
training_data_num = len(training_data)
test_data_num = len(test_data)

Train_image  = np.empty((row * col+1, training_data_num),dtype=float)
Train_label  = np.empty(training_data_num, dtype=int)

Test_image  = np.empty((row * col+1, test_data_num), dtype=float)
Test_label  = np.empty(test_data_num, dtype=int)
```

0.4 3. Define functions

In [50]:
```
#
# normalize the values of the input data to be [0, 1]
#
def normalize(data):

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

    return output

def Matrix_calculation(label, prediction, label_num):

    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(label[i,0] == prediction[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)
    true_positive_rate = TP_num / label_num
    error_rate = FP_num / label_num

    print("==================================================")
    print("# True Positive : %.2f" % TP_num)
    print("# False Positive : %.2f" % FP_num)
    print("True Positive rate: %.3f" % (true_positive_rate*100))
    print("Error rate: %.3f" % (error_rate*100))
    print("==================================================")

def Prediction(label, image, num):
    # This is a prediction function
```

```
    w = np.empty((1+row*col, 1, 10))

    # Calculating predicted labels
    predicted_label = np.empty((num, 1))

    # from 0 to 9
    for i in range(0,10):
        real_label = np.where((label==i), +1, -1).\
        reshape((num,1))
        x_value = np.copy(image).transpose()
        np.copyto(w[:,:,i], np.matmul(np.linalg.pinv(x_value), real_label))

    l = np.empty((10))
    for i in range(0, num):
        for j in range(0,10):
            l[j] = np.matmul(x_value[i,:], w[:,:,j])
        predicted_label[i] = np.argmax(l)

    return predicted_label
```

0.5 4. Vectorize train and test data

```
In [25]: 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)
             Train_label[count]       = label
             Train_image[:, count]    = np.insert(im_vector, 0, 1)

         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)

             Test_label[count]       = label
             Test_image[:, count]    = np.insert(im_vector, 0, 1)
```

0.6 5. Prediction function

0.6.1 Result : Training data

```
In [33]: Train_predicted_label = Prediction(Train_label, Train_image, training_data_num)
```

```
In [52]: Matrix_calculation(Train_label.reshape((training_data_num,1)),
                            Train_predicted_label, training_data_num)
```

```
==================================================
# True Positive : 51463.00
# False Positive : 8537.00
True Positive rate: 85.772
Error rate: 14.228
==================================================
```

0.6.2 Result : Test data

```
In [35]: Test_predicted_label = Prediction(Test_label, Test_image, test_data_num)
```

```
In [51]: Matrix_calculation(Test_label.reshape((test_data_num,1)),
                            Test_predicted_label, test_data_num)
```

```
==================================================
# True Positive : 8876.00
# False Positive : 1124.00
True Positive rate: 88.760
Error rate: 11.240
==================================================
```