Assignment11_20133096_HyunjaeLee

June 6, 2019

0.1 20133096 Hyunjae Lee

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.

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

0.3 2. Read data

```
In [2]: 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 [18]: #
         # 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, row, col, k):
    # This is a prediction function

    w = np.empty((k, 1, 10))
    target = np.random.randn(k, 1+row*col)

    # 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(KrandomFeatures(target, k,image,row,col))
        np.copyto(w[:,:,i], np.matmul(np.linalg.pinv(x_value),real_label))

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

    return predicted_label

def KrandomFeatures(target, k, image, row, col):

    output = np.inner(target, image.transpose())
    output = output.transpose()
    output = np.maximum(output, 0)

    return output
```

0.5 4. Vectorize train and test data

In [4]:
```
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 [19]: Train_predicted_label = Prediction(Train_label, Train_image,
                                            training_data_num, row, col, 1500)
```

```
In [20]: Matrix_calculation(Train_label.reshape((training_data_num,1)),
                            Train_predicted_label, training_data_num)
```

```
=================================================
# True Positive : 57404.00
# False Positive : 2596.00
True Positive rate: 95.673
Error rate: 4.327
=================================================
```

0.6.2 Result : Test data

```
In [21]: Test_predicted_label = Prediction(Test_label, Test_image,
                                           test_data_num, row, col, 1500)
```

```
In [22]: Matrix_calculation(Test_label.reshape((test_data_num,1)),
                            Test_predicted_label, test_data_num)
```

```
=================================================
# True Positive : 9855.00
# False Positive : 145.00
True Positive rate: 98.550
Error rate: 1.450
=================================================
```