Assignment09_20133096_HyunjaeLee

May 28, 2019

0.0.1 20133096 Hyunjae Lee

0.0.2 Assignment 09

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.

0.1 1. Read mnist data

```
In [71]: import numpy as np
         import matplotlib.pyplot as plt
         import pandas as pd
         import os
         from pandas import Series, DataFrame


         # Open mnist files

         train_data = "mnist_train.csv"
         test_data = "mnist_test.csv"

         train_data_open = open(train_data, "r")
         test_data_open = open(test_data, "r")

         trainData = train_data_open.readlines()
         testData = test_data_open.readlines()
```

```
train_data_open.close()
test_data_open.close()


# Some variables

row = 28
col = 28

len_train = len(trainData)
len_test = len(testData)
```

0.2 2. Define functions

```
In [124]: def Initiate(data, image, label):

              count = 0

              for line in data:
                  line_data = line.split(',')

                  # 1st column is label of the image
                  eachlabel = line_data[0]

                  # the rest of columns represent image intensity
                  vec = np.asfarray(line_data[1:])
                  vec = normalize(vec)

                  label[count] = eachlabel
                  image[:, count] = vec

                  count += 1

              return image, label

          def whitening(inputdata, mode=0):
              mean = np.mean(inputdata)
              std = np.std(inputdata)

              if mode == 1:
                  return mean, std

              return (inputdata - mean) / std

          def backwhitening(inputdata):
              mean , std = whitening(inputdata, mode=1)
```

```
    return (inputdata * std) + mean

def normalize(data):

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

    return normalized

def binaryClassifier(data, target):

    length = len(data)
    res = np.zeros((length))

    for k in range(length):
        if(data[k] == target):
            res[k] = 1
        else:
            res[k] = -1

    return res

def checkAnswer(data):
    if data >= 0 :
        return 1
    else:
        return -1

def makeTable(label, image, theta, length):
    table = np.zeros((2,2))
    binary_hat = binaryClassifier(label, 0)
    avg = np.zeros((row * col, 4))

    true_count = 0
    false_count = 0

    for i in range(length):
        if checkAnswer(theta.dot(image[:, i])) == 1:
            if(binary_hat[i] == 1):
                # True Positive
                table[0][0] += 1
                avg[:, 0] += image[:, i]
                true_count += 1
            else:
                # False Positive
                table[1][0] += 1
                avg[:, 1] += image[:, i]
                false_count += 1
        else:
```

```
            if(binary_hat[i] != 1):
                # False Negative
                table[1][1] += 1
                avg[:, 3] += image[: ,i]
                false_count += 1

            else:
                # True Negative
                table[0][1] += 1
                avg[:, 2] += image[:, i]
                true_count += 1

    data = {
        'TRUE' : [table[0][0], table[0][1]],
        'FALSE' : [table[1][0], table[1][1]],
    }

    ratiodata = {
        'TRUE' : [ table[0][0]/true_count, table[0][1]/true_count],
        'FALSE' : [ table[1][0]/false_count, table[1][1]/false_count],
    }
    print('# zero : ', true_count)
    print('# Non-zero : ', false_count)

    frame = DataFrame(data, columns = ['TRUE', 'FALSE'],
                      index = ['POSITIVE', 'NEGATIVE'])
    display(frame)

    ratioframe = DataFrame(ratiodata, columns = ['TRUE', 'FALSE'],
                      index = ['POSITIVE', 'NEGATIVE'])
    display(ratioframe)

    showAvgImage(avg[:, 0] / table[0][0] , avg[:, 1] / table[1][0],
                avg[:, 2] / table[0][1], avg[:, 3] / table[1][1], )


def showAvgImage(truePos, falsePos, trueNeg, falseNeg):

    plt.subplots_adjust(hspace=0.5)

    fig1 = plt.subplot(2,2,1)
    fig1.imshow(truePos.reshape(row, col))
    fig1.set_title("True Positive")

    fig2 = plt.subplot(2,2,2)
    fig2.imshow(falsePos.reshape(row, col))
    fig2.set_title("False Positive")
```

```
        fig3 = plt.subplot(2,2,3)
        fig3.imshow(trueNeg.reshape(row, col))
        fig3.set_title("True Negative")

        fig4 = plt.subplot(2,2,4)
        fig4.imshow(falseNeg.reshape(row, col))
        fig4.set_title("False Negative")
```

```
In [73]: # Initiate vectorizing each data set
         train_image = np.empty((row * col , len_train) , dtype = float)
         test_image = np.empty((row * col , len_test) , dtype = float)
         train_label = np.empty(len_train, dtype = int)
         test_label = np.empty(len_test, dtype = int)

         train_image, train_label = Initiate(trainData, train_image, train_label)
         test_image, test_label = Initiate(testData, test_image, test_label)
```

0.3 3. Compute an optimal model parameter using the training dataset

```
In [74]: index = np.where(~train_image.any(axis=1))[0]
         matrixA = train_image[~np.all(train_image == 0, axis = 1)]
         A = np.matrix(np.transpose(matrixA))
         B = np.matrix(np.transpose(binaryClassifier(train_label, 0)))

         temp_theta = (A.T * A).I*A.T*B.T
         theta = np.zeros((row * col))
         count = 0

         for i in range(row * col):
             if i not in index:
                 theta[i] = temp_theta[count]
                 count += 1
```

0.4 4. 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.

0.4.1 4.1 : Training set

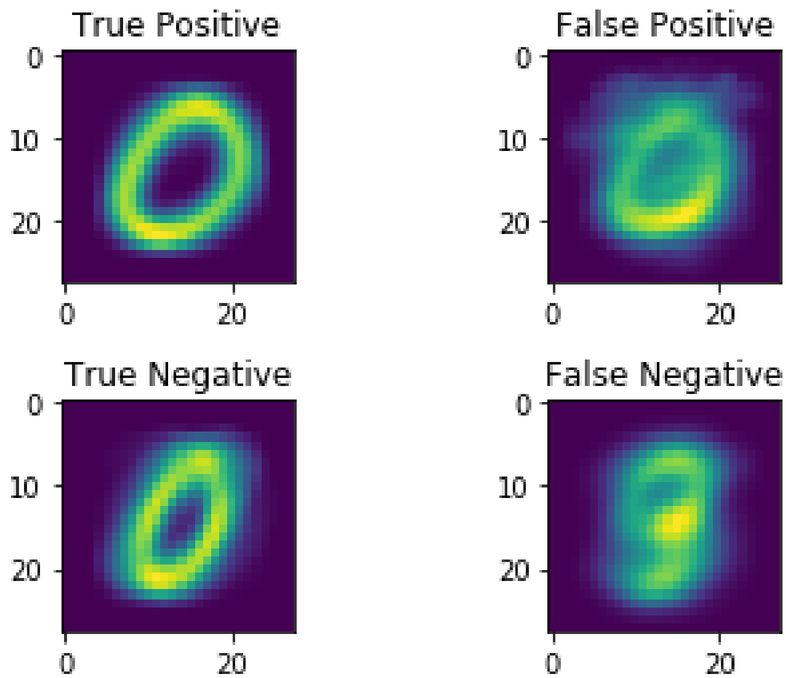
```
In [125]: makeTable(train_label, train_image, theta, len_train)
          print(" Total # train set : ", len_train)
```

```
# zero :  5923
# Non-zero :  54077


            TRUE    FALSE
POSITIVE  5353.0   1289.0
NEGATIVE   570.0  52788.0
```

```
             TRUE     FALSE
POSITIVE  0.903765  0.023836
NEGATIVE  0.096235  0.976164
```

```
 Total # train set :  60000
```



0.4.2 4.2 : Test set

```
In [126]: makeTable(test_label, test_image, theta, len_test)
          print(" Total # test set : ", len_test)
```

```
# zero :  980
# Non-zero :  9020
```

```
          TRUE   FALSE
POSITIVE  916.0   214.0
NEGATIVE   64.0  8806.0
```

```
             TRUE     FALSE
POSITIVE  0.934694  0.023725
NEGATIVE  0.065306  0.976275
```

```
Total # test set :  10000
```