Assignment09

May 30, 2019

20164245 Hong Jin

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.

1 Set up

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

file_data = "mnist_train.csv"
handle_file = open(file_data, "r")
data = handle_file.readlines()
handle_file.close()

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

size_row = 28
size_col = 28
dim = size_col * size_row
```

```
num_image = len(data)
test_num_image = len(test_data)
```

2 Normalization

```
In [2]: def normalize(data):
            data_normalized = (data-min(data)) / (max(data) - min(data))
            return (data_normalized)
```

3 Functions

```
In [3]: def distance(x,y):
            d = x - y
            s = d ** 2
            return s

        def check(M,val):
            length = len(M)
            res = np.zeros((length))
            for i in range(length):
                if(M[i] == val):
                    res[i] = 1
                else:
                    res[i] = -1
            return res

        def sign(x):
            if(x>=0):
                return 1
            else:
                return -1
```

4 Make label, image array with train, test data

```
In [4]: list_image = np.empty((size_row * size_col, num_image), dtype=float)
        list_label = np.empty(num_image, dtype=int)
        test_list_image = np.empty((size_row * size_col, test_num_image), dtype=float)
        test_list_label = np.empty(test_num_image, dtype=int)
        count = 0
        test_count = 0

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

```
            list_label[count] = label
            list_image[:,count] = im_vector

            count += 1

        for test_line in test_data:
            test_line_data = test_line.split(',')
            test_label = test_line_data[0]
            test_im_vector = np.asfarray(test_line_data[1:])
            test_im_vector = normalize(test_im_vector)

            test_list_label[test_count] = test_label
            test_list_image[:,test_count] = test_im_vector

            test_count += 1
```

5 Define Matrix A

$f_i(x) = x^{i-1}, i = 1, \ldots, p$

$\hat{f}(x) = \theta_1 + \theta_2 x + \cdots + \theta_p x^{p-1}$

$$A = \begin{bmatrix} 1 & x^{(1)} & \cdots & (x^{(1)})^{p-1} \\ 1 & x^{(2)} & \cdots & (x^{(2)})^{p-1} \\ \vdots & \vdots & & \vdots \\ 1 & x^{(N)} & \cdots & (x^{(N)})^{p-1} \end{bmatrix}$$

(x^i means scalar x to ith power; $x^{(i)}$ is ith data point)

$\theta = (A^T A)^{-1} A^T b$

In [5]:
```
R = np.zeros((dim,dim))
for i in range(dim):
    R[i] = np.random.normal(0,1,size=dim)

def defMatrix(x, p):
    model = np.zeros((dim,dim))
    for i in range(p):
        model[i] = R[i]
    return np.dot(model,x)
```

6 Compute an optimal model parameter using the training dataset

In [6]:
```
scores = np.zeros(10)
B = np.matrix(np.transpose(check(list_label,0)))   # (60000, 1)
B_hat = check(list_label,0)
B_hat_test = check(test_list_label,0)

train_set = list_image[:54000]
```

```
test_set = list_image[54000:]

for j in range(10):
    p = 2**j
    feature = defMatrix(list_image, p)

    index = np.where(~feature.any(axis=1))[0]
    A = feature[~np.all(feature == 0, axis=1)]
    A = np.matrix(np.transpose(A))

    temp_theta = (A.T * A).I*A.T*B.T

    theta = np.zeros((size_col*size_row))
    count_num = 0

    for i in range(dim):
        if i not in index:
            theta[i]=temp_theta[count_num]
            count_num +=1

    nums = np.zeros((2,2))
    dist = 0
    min_num = 100000000
    feature = defMatrix(list_image, p)
    for i in range(count):
        dist += distance(theta, feature[:,i])

    for i in range(len(dist)):
        if(dist[i]!=0):
            if(dist[i] < min_num):
                min_num = dist[i]
    scores[j] = min_num
m = 2**np.argmin(scores)
print("best = " + str(m))
```

```
best = 512
```

7 Plot Everage Image

```
In [7]: def plotImage(im_avg):
            p1 = plt.subplot(2,2,1)
            p1.imshow(im_avg[:,0].reshape((size_row, size_col)),cmap='gray')
            p1.set_title("True Positive")
            p2 = plt.subplot(2,2,2)
            p2.imshow(im_avg[:,1].reshape((size_row, size_col)),cmap='gray')
            p2.set_title("False Positive")
```

```
p3 = plt.subplot(2,2,3)
p3.imshow(im_avg[:,2].reshape((size_row, size_col)),cmap='gray')
p3.set_title("True Negative")
p4 = plt.subplot(2,2,4)
p4.imshow(im_avg[:,3].reshape((size_row, size_col)),cmap='gray')
p4.set_title("False Negative")
plt.subplots_adjust(hspace=1)
```

8 Compute Accuracy

In [8]:
```
def computeAcc(image, counts, hat):
    nums = np.zeros((2,2))
    acc_num = 0
    nacc_num = 0
    im_avg = np.zeros((dim,4))
    feature = defMatrix(image, m)
    for i in range(counts):
        if sign(theta.dot(feature[:,i])) == 1:
            acc_num += 1
            if(hat[i] == 1):
                # True Positive
                nums[0][0] += 1
                im_avg[:,0] += image[:,i]
            else:
                # False Positive
                nums[1][0] += 1
                im_avg[:,1] += image[:,i]
        else:
            nacc_num += 1
            if(hat[i] == 1):
                # False Negative
                nums[0][1] += 1
                im_avg[:,3] += image[:,i]
            else:
                # True Negative
                nums[1][1] += 1
                im_avg[:,2] += image[:,i]


    im_avg[:,0] /= nums[0,0]
    im_avg[:,1] /= nums[1,0]
    im_avg[:,2] /= nums[1,1]
    im_avg[:,3] /= nums[0,1]


    plotImage(im_avg)
```

```
        tp = nums[0,0] / acc_num
        fp = nums[1,0] / acc_num
        tn = nums[1,1] / nacc_num
        fn = nums[0,1] / nacc_num

        print("Tp = " + str(tp))
        print("Fp = " + str(fp))
        print("Tn = " + str(tn))
        print("Fn = " + str(fn))
        print("Positive = " + str(tp+fp))
        print("Negative = " + str(tn+fn))
```

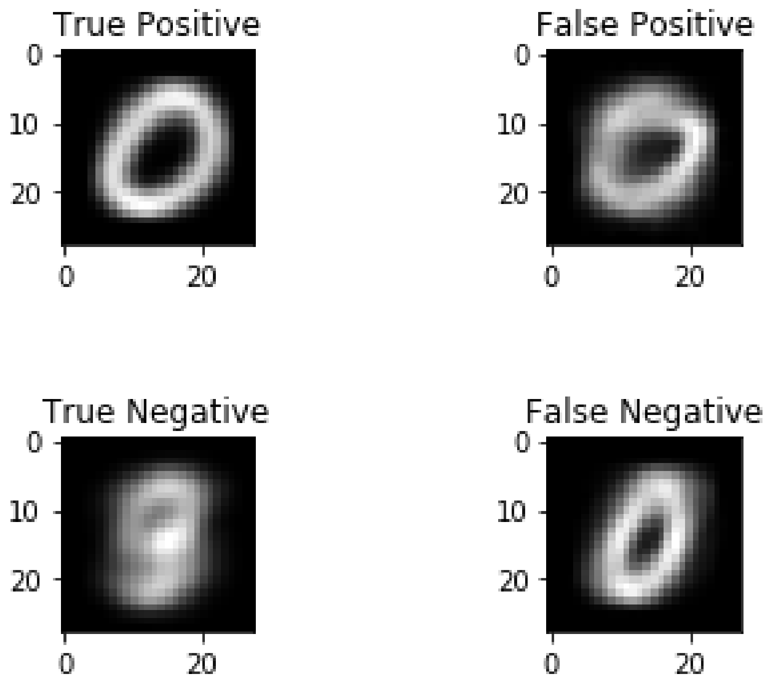
9 Compute (1) True Positive, (2) False Positive, (3) True Negative, (4) False Negative

```
In [9]: feature = defMatrix(list_image, m)
        index = np.where(~feature.any(axis=1))[0]
        A = feature[~np.all(feature == 0, axis=1)]
        A = np.matrix(np.transpose(A))
        B = np.matrix(np.transpose(check(list_label,0)))   # (60000, 1)

        temp_theta = (A.T * A).I*A.T*B.T
        theta = np.zeros((dim))
        count_num = 0

        for i in range(dim):
            if i not in index:
                theta[i]=temp_theta[count_num]
                count_num +=1
```

9.1 Training set

```
In [10]: computeAcc(list_image, count, B_hat)
```

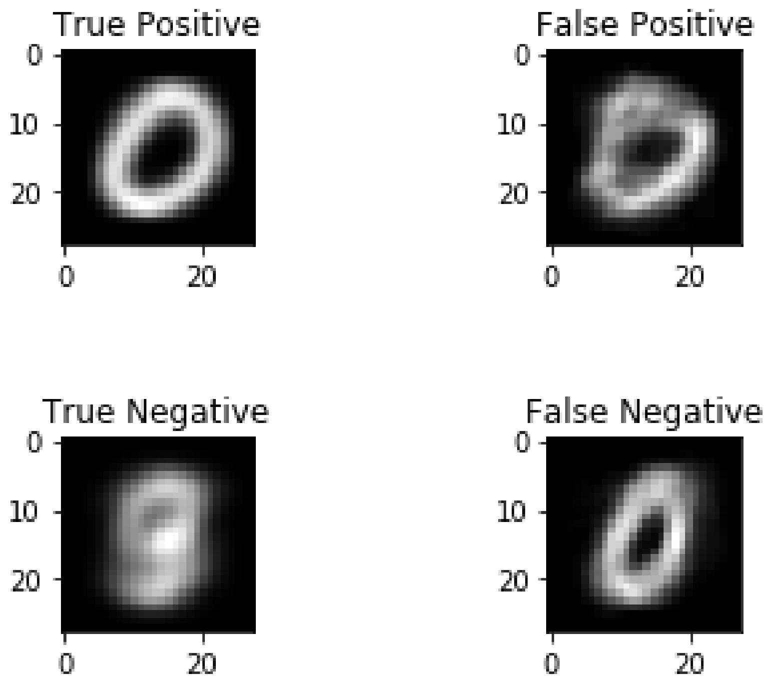
```
Tp = 0.9406690140845071
Fp = 0.059330985915492955
Tn = 0.9893225331369662
Fn = 0.010677466863033874
Positive = 1.0
Negative = 1.0
```



9.2 Test set

```
In [11]: computeAcc(test_list_image, test_count, B_hat_test)
```

```
Tp = 0.9375639713408394
Fp = 0.062436028659160696
Tn = 0.9929070154050759
Fn = 0.007092984594924083
Positive = 1.0
Negative = 1.0
```


True Positive
False Positive
True Negative
False Negative
0
10
20
0
20