Build a binary classifier for each digit against all the other digits at MNIST dataset.

June 5, 2019

Professor: **Hong Hyung-Woo**

Student NB: **20166450**

Major: **Software Engineering**

Name: **Kim Young Min** 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.

In [1]:
```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from numpy.linalg import inv
from astropy.table import Table

%matplotlib inline

import pylab as pl
```

0.0.1 At First, Define all of Function which is needed.

Whitening Function.

In [2]:
```
def normalize_whitening(data):

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

    return(data_normalized)
```

Label List and Mnist Image Vector List.

In [3]:
```
def label_and_img(data):
    size_row        = 28    # height of the image
    size_col          = 28    # width of the image
    num_image         = len(data)
    count                = 0  # count for the number of images
    list_image  = np.empty((size_row * size_col, num_image), dtype=float)
    list_label  = np.empty(num_image, dtype=int)
    for line in data:
        line_data   = line.split(',')
        label       = line_data[0]
        im_vector   = np.asfarray(line_data[1:])
        im_vector   = normalize_whitening(im_vector)

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

        count += 1

    return list_label,list_image
```

Define Prediction Function $F_w(x)$**.** $f_w(x) = +1$ if label(x) = d $f_w(x) = -1$ if label(x) is not d

In [16]:
```
def f_x(list_label,num):
    f_x=np.asarray([0.0]*len(list_label))

    for i in range(len(list_label)):
        if list_label[i]== num:
            f_x[i]=1
        else:
            f_x[i]=-1

    return f_x
```

Method for making Pseudo Inverse

In [5]:
```
def pseudo_inverse(A,f_x):
    Apinv = (inv(A.T @ A)@ A.T)@f_x
    return Apinv
```

Seta = w and A = x Matrix $x = (x_1, x_2, ..., x_m)$ image in the dataset. $w = (w_0, w_1, ..., w_m)$

```
In [6]: def seta_and_A(list_image,f_x):
            one=np.array([[1.0]]*len(list_image))

            A= np.hstack([one,list_image])
            Apinv=np.linalg.pinv(A)
            seta=Apinv @ f_x

            return seta,A
```

Method for Classification_matrix.

```
In [40]: def Classification_matrix(final_f_x,list_label):
             cnt=0
             classification=np.asarray([[0]*10]*10)
             for k in range(10):
                 for j in range(10):
                     cnt=0
                     for i in range(len(list_label)):
                         if(final_f_x[i]==k and list_label[i]==j):
                             cnt=cnt+1
                     classification[k][j]=cnt

             return classification
```

```
In [69]: def truepositive_and_error_rate(list_label,classification_train):
             tp_cnt=0
             for i in range(10):
                 tp_cnt+=classification_train[i][i]

             tp_rate=tp_cnt/len(list_label)*100
             error_rate=(len(list_label)-tp_cnt)/len(list_label)*100

             return tp_rate,error_rate
```

0.0.2 Secondly, Performing about Train Set and Test Set.

1 For Train Set.

Read file.csv.

```
In [73]: file_data                = "mnist_train.csv"
         handle_file        = open(file_data, "r")
         data                        = handle_file.readlines()
         handle_file.close()
```

1.1 Compute an optimal model parameter using the training dataset

```
In [74]: list_label,list_image=label_and_img(data)
         list_image=np.array(list_image.T)
```

Compute A, f(x), w about each classifier.

```
In [75]: f_x_0=f_x(list_label,0)
         f_x_1=f_x(list_label,1)
         f_x_2=f_x(list_label,2)
         f_x_3=f_x(list_label,3)
         f_x_4=f_x(list_label,4)
         f_x_5=f_x(list_label,5)
         f_x_6=f_x(list_label,6)
         f_x_7=f_x(list_label,7)
         f_x_8=f_x(list_label,8)
         f_x_9=f_x(list_label,9)
```

```
In [76]: seta_0,A_0 = seta_and_A(list_image,f_x_0)
         seta_1,A_1 = seta_and_A(list_image,f_x_1)
         seta_2,A_2 = seta_and_A(list_image,f_x_2)
         seta_3,A_3 = seta_and_A(list_image,f_x_3)
         seta_4,A_4 = seta_and_A(list_image,f_x_4)
         seta_5,A_5 = seta_and_A(list_image,f_x_5)
         seta_6,A_6 = seta_and_A(list_image,f_x_6)
         seta_7,A_7 = seta_and_A(list_image,f_x_7)
         seta_8,A_8 = seta_and_A(list_image,f_x_8)
         seta_9,A_9 = seta_and_A(list_image,f_x_9)
```

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

```
In [77]: final_f_x=np.asarray([0]*len(list_label))
         argmax_list=np.asarray([0.0]*10)
         for i in range(len(list_label)):
             argmax_list[0]=np.sum(seta_0*A_0[i])
             argmax_list[1]=np.sum(seta_1*A_1[i])
             argmax_list[2]=np.sum(seta_2*A_2[i])
             argmax_list[3]=np.sum(seta_3*A_3[i])
             argmax_list[4]=np.sum(seta_4*A_4[i])
             argmax_list[5]=np.sum(seta_5*A_5[i])
             argmax_list[6]=np.sum(seta_6*A_6[i])
             argmax_list[7]=np.sum(seta_7*A_7[i])
             argmax_list[8]=np.sum(seta_8*A_8[i])
             argmax_list[9]=np.sum(seta_9*A_9[i])
             for j in range(10):
                 if argmax_list[j]==np.max(argmax_list):
                     final_f_x[i]=j
                     break;
```

Prediction Function

```
In [78]: print(final_f_x)
```

```
[5 0 4 ... 5 6 8]
```

Classification of Matrix

```
In [79]: classification_train = Classification_matrix(final_f_x,list_label)
         print(classification_train)
```

```
[[5682    2   99   42   10  164  108   55   75   68]
 [   7 6548  264  167   99   95   74  189  493   60]
 [  18   40 4792  176   42   28   61   37   63   20]
 [  14   15  149 5159    6  433    1   47  226  117]
 [  24   19  108   32 5211  105   70  170  105  371]
 [  43   31   11  124   50 3990   90    9  222   12]
 [  64   14  234   56   39  192 5476    2   56    4]
 [   4   12   91  115   23   36    0 5426   20  491]
 [  61   55  192  135   59  235   35   10 4411   38]
 [   6    6   18  125  303  143    3  320  180 4768]]
```

1.2 Compute (1) true positive rate, (2) error rate using (1) training dataset and (2) testing dataset.

```
In [80]: tp_rate,error_rate=truepositive_and_error_rate(list_label,classification_train)
         print(tp_rate)
         print(error_rate)
```

```
85.77166666666668
14.228333333333335
```

2 For Test set.

```
In [81]: file_data_test                  = "mnist_test.csv"
         handle_file_test        = open(file_data_test, "r")
         data_test                          = handle_file_test.readlines()
         handle_file_test.close()
```

2.1 Compute an optimal model parameter using the test dataset

```
In [82]: list_label_test,list_image_test=label_and_img(data_test)
         list_image_test=np.array(list_image_test.T)
```

Compute A, f(x), w about each classifier.

In [83]:
```
f_x_0_test=f_x(list_label_test,0)
f_x_1_test=f_x(list_label_test,1)
f_x_2_test=f_x(list_label_test,2)
f_x_3_test=f_x(list_label_test,3)
f_x_4_test=f_x(list_label_test,4)
f_x_5_test=f_x(list_label_test,5)
f_x_6_test=f_x(list_label_test,6)
f_x_7_test=f_x(list_label_test,7)
f_x_8_test=f_x(list_label_test,8)
f_x_9_test=f_x(list_label_test,9)
```

In [84]:
```
seta_0_test,A_0_test = seta_and_A(list_image_test,f_x_0_test)
seta_1_test,A_1_test = seta_and_A(list_image_test,f_x_1_test)
seta_2_test,A_2_test = seta_and_A(list_image_test,f_x_2_test)
seta_3_test,A_3_test = seta_and_A(list_image_test,f_x_3_test)
seta_4_test,A_4_test = seta_and_A(list_image_test,f_x_4_test)
seta_5_test,A_5_test = seta_and_A(list_image_test,f_x_5_test)
seta_6_test,A_6_test = seta_and_A(list_image_test,f_x_6_test)
seta_7_test,A_7_test = seta_and_A(list_image_test,f_x_7_test)
seta_8_test,A_8_test = seta_and_A(list_image_test,f_x_8_test)
seta_9_test,A_9_test = seta_and_A(list_image_test,f_x_9_test)
```

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

In [85]:
```
final_f_x_test=np.asarray([0]*len(list_label_test))
argmax_list_test=np.asarray([0.0]*10)
for i in range(len(list_label_test)):
    argmax_list_test[0]=np.sum(seta_0_test*A_0_test[i])
    argmax_list_test[1]=np.sum(seta_1_test*A_1_test[i])
    argmax_list_test[2]=np.sum(seta_2_test*A_2_test[i])
    argmax_list_test[3]=np.sum(seta_3_test*A_3_test[i])
    argmax_list_test[4]=np.sum(seta_4_test*A_4_test[i])
    argmax_list_test[5]=np.sum(seta_5_test*A_5_test[i])
    argmax_list_test[6]=np.sum(seta_6_test*A_6_test[i])
    argmax_list_test[7]=np.sum(seta_7_test*A_7_test[i])
    argmax_list_test[8]=np.sum(seta_8_test*A_8_test[i])
    argmax_list_test[9]=np.sum(seta_9_test*A_9_test[i])
    for j in range(10):
        if argmax_list_test[j]==np.max(argmax_list_test):
            final_f_x_test[i]=j
            break;
```

Prediction Function

In [86]:
```
print(final_f_x_test)
```

```
[7 2 1 ... 4 5 6]
```

Classification of Matrix

```
In [87]: classification_train_test = Classification_matrix(final_f_x_test,list_label_test)
         print(classification_train_test)
```

```
[[ 949    0   11    4    0   17   15    5   13   14]
 [   0 1115   36   16   19   23    9   34   44   13]
 [   1    5  884   18    5    3    4   13    8    3]
 [   6    1   17  893    0   55    0    3   26   14]
 [   1    2   13    5  898   18    9   21   13   58]
 [   8    1    0   14    1  692   16    1   29    3]
 [   7    4   23    3    6   21  896    1    9    1]
 [   1    1   20   17    1   16    0  908    8   53]
 [   6    6   23   24    7   37    9    1  801   10]
 [   1    0    5   16   45   10    0   41   23  840]]
```

2.2 Compute (1) true positive rate, (2) error rate using (1) training dataset and (2) testing dataset.

```
In [88]: tp_rate_test,error_rate_test=truepositive_and_error_rate(list_label_test,classificati
         print(tp_rate_test)
         print(error_rate_test)
```

```
88.75999999999999
11.24
```

3 Show as a Matrix.

```
In [89]: print("""
         +-----------------------+-----------------------+--------------------+
         |                       |       Train set       |      Test set      |
         +-----------------------+-----------------------+--------------------+
         |  True  positive rate  |        %6.3f %%       |      %6.3f %%      |
         +-----------------------+-----------------------+--------------------+
         |  Error rate           |        %6.3f %%       |      %6.3f %%      |
         +-----------------------+-----------------------+--------------------+"""
         %(tp_rate,tp_rate_test,error_rate,error_rate_test))
```

```
+-----------------------+-----------------------+--------------------+
|                       |       Train set       |      Test set      |
+-----------------------+-----------------------+--------------------+
|  True  positive rate  |        85.772 %       |      88.760 %      |
```

Error rate	14.228 %	11.240 %