Build a binary classifier to classify digit 0 against all the other digits at MNIST dataset

May 30, 2019

Professor: **Hong Hyung-Woo**

Student NB: **20166450**

Major: **Software Engineering**

Name: **Kim Young Min** 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.

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

```
In [359]: 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
```

Whitening Function.

```
In [360]: def normalize_whitening(data):

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

              return(data_normalized)
```

Label List and Mnist Image Vector List.

```
In [361]: 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
```

Count how many Zero image in Mnist data sets and Classify Zero Image Vector List.

```
In [362]: def zero_img(list_label,list_image):
              cnt_zero=0
              for i in range(len(list_label)):
                  if list_label[i]==0:
                      cnt_zero+=1

              zero_img=np.array([[0.0]*784]*cnt_zero)

              k=0
              for j in range(len(list_label)):
                  if list_label[j]==0:
                      zero_img[k]=list_image[j]
                      k=k+1

              return cnt_zero,zero_img
```

Define Prediction Function $F_w(x)$. $f_w(x) = +1$ if label(x) = 0 $f_w(x) = -1$ if label(x) is not 0

```
In [363]: def f_x(list_label):
              f_x=np.array([0.0]*len(list_label))

              for i in range(len(list_label)):
                  if list_label[i]==0:
                      f_x[i]=1
                  else:
                      f_x[i]=-1

              return f_x
```

Method for making Pseudo Inverse

```
In [364]: 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 [365]: 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
```

Approximate $F_w(x)$ **for minimizing the following objective function:** $\sum_i (f_w(x^i - y^i))^2$

```
In [366]: def apx_f_x(seta,A,list_label):
              final_f_x=np.array([0]*len(list_label))
              for i in range(len(list_label)):
                  if (np.sum(seta * A[i])) > 0:
                      final_f_x[i]=0
                  else:
                      final_f_x[i]=100
              return final_f_x
```

Confusion_matrix for true positive, false positive, true negative, false negative

```
In [367]: def Confusion_matrix(final_f_x,list_label):
              real_and_say_zero=0
              for i in range(len(list_label)):
                  if(final_f_x[i]==0 and list_label[i]==0):
                      real_and_say_zero=real_and_say_zero+1
```

```
noreal_but_say_zero=0
for i in range(len(list_label)):
    if(final_f_x[i]==0 and list_label[i]!=0):
        noreal_but_say_zero=noreal_but_say_zero+1


real_but_say_nonzero=0
for i in range(len(list_label)):
    if(final_f_x[i]!=0 and list_label[i]==0):
        real_but_say_nonzero=real_but_say_nonzero+1

real_and_say_nonzero=0
for i in range(len(list_label)):
    if(final_f_x[i]!=0 and list_label[i]!=0):
        real_and_say_nonzero=real_and_say_nonzero+1

return real_and_say_zero,noreal_but_say_zero,real_but_say_nonzero,real_and_say_no
```

0.0.2 Secondly, Performing about Train Set and Test Set.

1 For Train Set.

Read file.csv.

```
In [339]: 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

Build a binary classifier to classify digit 0 against all the other digits at MNIST dataset.

```
In [340]: list_label,list_image=label_and_img(data)
          list_image=np.array(list_image.T)

          cnt_zero,zero_img=zero_img(list_label,list_image)
```

$f_w(x)$ **= 1 or -1**

```
In [341]: f_x=f_x(list_label)
```

Let $x = (x_1, x_2, ..., x_m)$ **be a vector representing an image in the dataset.**

```
In [342]: seta,A=seta_and_A(list_image,f_x)
```

Prediction Function

```
In [343]: final_f_x=apx_f_x(seta,A,list_label)

          print(final_f_x)
```

```
[100   0 100 ... 100 100 100]
```

1.2 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.

```
In [451]: real_and_say_zero,noreal_but_say_zero,real_but_say_nonzero,real_and_say_nonzero= Con
```

```
In [456]: true_positive_rate=(real_and_say_zero/cnt_zero)
          print(true_positive*100,"%")
```

```
87.23619787269965 %
```

```
In [460]: false_positive=noreal_but_say_zero/(60000-cnt_zero)
          print(false_positive*100,"%")
```

```
0.3310094864729922 %
```

```
In [472]: false_negative=real_but_say_nonzero/cnt_zero
          print(false_negative*100,"%")
```

```
12.763802127300355 %
```

```
In [462]: true_negative=real_and_say_nonzero/(60000-cnt_zero)
          print(true_negative*100,"%")
```

```
99.668990513527 %
```

```
In [349]: print(cnt_zero)
          print(noreal_but_say_zero)
```

```
5923
179
```

2 For Test set.

```
In [368]: 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 training dataset

For prediction Function: $w : f_w(x) = w_0 * 1 + w_1 * x_1 + w_2 * x_2 + ... + w_m * x_m$ where $w = (w_0, w_1, ..., w_m)$

In [369]:
```
list_label_test,list_image_test=label_and_img(data_test)
list_image_test=np.array(list_image_test.T)


cnt_zero_test,zero_img_test = zero_img(list_label_test,list_image_test)
```

In [370]:
```
f_x_test=f_x(list_label_test)
seta_test,A_test=seta_and_A(list_image_test,f_x_test)
final_f_x_test=apx_f_x(seta_test,A_test,list_label_test)

print(final_f_x_test)
```

```
[100 100 100 ... 100 100 100]
```

2.2 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.

In [445]:
```
real_and_say_zero_t,noreal_but_say_zero_t,real_but_say_nonzero_t,real_and_say_nonzer
```

In [373]:
```
true_positive_rate_test=(real_and_say_zero_t/cnt_zero_test)
print(true_positive_rate_test*100,"%")
```

```
91.93877551020408 %
```

In [374]:
```
false_positive_rate_test=noreal_but_say_zero_t/(10000-cnt_zero_test)
print(false_positive_rate_test*100,"%")
```

```
0.3547671840354767 %
```

In [474]:
```
false_negative_rate_test=real_but_say_nonzero_t/cnt_zero_test
print(false_negative_rate_test*100,"%")
```

```
8.061224489795919 %
```

In [475]:
```
true_negative_rate_test=real_and_say_nonzero_t/(10000-cnt_zero_test)
print(true_negative_rate_test*100,"%")
```

```
99.64523281596452 %
```

3 Show The Confusion Matrix

```
In [477]: print("""
          +------------------+----------------------+--------------------+
          |                  |      Train set       |     Test set       |
          +------------------+----------------------+--------------------+
          |  True  positive  |       %6.3f %%       |     %6.3f %%       |
          +------------------+----------------------+--------------------+
          |  False positive  |       %6.3f %%       |     %6.3f %%       |
          +------------------+----------------------+--------------------+
          |  True  negative  |       %6.3f %%       |     %6.3f %%       |
          +------------------+----------------------+--------------------+
          |  False negative  |       %6.3f %%       |     %6.3f %%       |
          +------------------+----------------------+--------------------+"""
          %(true_positive_rate*100,true_positive_rate_test*100,false_positive*100,false_positi
```

```
+------------------+----------------------+--------------------+
|                  |      Train set       |     Test set       |
+------------------+----------------------+--------------------+
|  True  positive  |       87.236 %       |     91.939 %       |
+------------------+----------------------+--------------------+
|  False positive  |        0.331 %       |      0.355 %       |
+------------------+----------------------+--------------------+
|  True  negative  |       99.669 %       |     99.645 %       |
+------------------+----------------------+--------------------+
|  False negative  |       12.764 %       |      8.061 %       |
+------------------+----------------------+--------------------+
```