Build a binary classifier based on k random features for each digit against all the other digits at MNIST dataset.

June 6, 2019

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 $g_k = max(innerproduction(r_k, x), 0)$ 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.

In [24]:
```
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 Define all of Function which is needed.

Whitening Function.

In [25]:
```
def normalize_whitening(data):
```

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

        return(data_normalized)
```

Label List and Mnist Image Vector List.

```
In [26]: 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_bf   = np.asfarray(line_data[1:])

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

                 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 [28]: 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
```

Define random vector r_k

```
In [29]: def r_x(list_image):
             r_k=np.asarray([[0.0]*784]*784)
             for i in range(784):
                 index=np.random.randint(len(list_image))
                 r_k[i]=list_image[index]
             return r_k
```

Define r_k, x_k inner product.

```
In [27]: def g_x(list_image,list_label):
             g_k=np.asarray([[0.0]*784]*len(list_label))
             for i in range(len(list_label)):
                 g_k[i]=r_k@list_image[i].T
                 g_k[i]=normalize_whitening(g_k[i])
             return g_k
```

Method for making Pseudo Inverse

```
In [30]: def pseudo_inverse(A,f_x):
             Apinv = (inv(A.T @ A)@ A.T)@f_x
             return Apinv
```

Seta = w and A = x Matrix $g = (g_1, g_2, ..., g_m)$ image in the dataset. $w = (w_0, w_1, ..., w_m)$

```
In [31]: def seta_and_A(g_k,f_x):
             one=np.array([[1.0]]*len(g_k))

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

             return seta,A
```

Method for Classification_matrix.

```
In [32]: 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
```

Compute true positive rate and error rate.

```
In [33]: 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
```

1 For Train Set.

```
In [34]: file_data                = "mnist_train.csv"
         handle_file        = open(file_data, "r")
         data                        = handle_file.readlines()
         handle_file.close()

In [35]: list_label,list_image=label_and_img(data)
         list_image=np.asarray(list_image.T)

In [36]: r_k=r_x(list_image)

In [37]: g_k= g_x(list_image,list_label)

In [38]: 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 [39]: seta_0,A_0 = seta_and_A(g_k,f_x_0)
         seta_1,A_1 = seta_and_A(g_k,f_x_1)
         seta_2,A_2 = seta_and_A(g_k,f_x_2)
         seta_3,A_3 = seta_and_A(g_k,f_x_3)
         seta_4,A_4 = seta_and_A(g_k,f_x_4)
         seta_5,A_5 = seta_and_A(g_k,f_x_5)
         seta_6,A_6 = seta_and_A(g_k,f_x_6)
         seta_7,A_7 = seta_and_A(g_k,f_x_7)
         seta_8,A_8 = seta_and_A(g_k,f_x_8)
         seta_9,A_9 = seta_and_A(g_k,f_x_9)

In [40]: 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;
```

```
In [41]: print(final_f_x)
```

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

Classification of Matrix.

```
In [42]: classification_train = Classification_matrix(final_f_x,list_label)
         print(classification_train)
```

```
[[5620    1   89   44    7  207  122   64   69   74]
 [   3 6548  136   73   39   49   29  104  308   39]
 [  26   45 4915  195   42   39   76   44   92   21]
 [  23   14  153 5230    6  519    5   70  208  114]
 [  16   28   88   22 5280   67   43  128   81  363]
 [  55   26   11   97   53 3956   97   11  249   16]
 [  68   19  226   55   57  183 5473    7   45    6]
 [   3    9   79  126   21   41    0 5517   23  441]
 [ 105   44  238  179   56  223   71   23 4633   57]
 [   4    8   23  110  281  137    2  297  143 4818]]
```

1.1 Compute (1) true positive rate, (2) error rate using the train set

```
In [43]: tp_rate,error_rate=truepositive_and_error_rate(list_label,classification_train)
         print(tp_rate)
         print(error_rate)
```

```
86.65
13.350000000000001
```

2 For Test set.

```
In [44]: file_data_test                = "mnist_test.csv"
         handle_file_test        = open(file_data_test, "r")
         data_test                        = handle_file_test.readlines()
         handle_file_test.close()
```

In [45]:
```
list_label_test,list_image_test=label_and_img(data_test)
list_image_test=np.array(list_image_test.T)
```

In [46]:
```
r_k_test=r_x(list_image_test)
```

In [47]:
```
g_k_test= g_x(list_image_test,list_label_test)
```

In [48]:
```
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 [49]:
```
seta_0_test,A_0_test = seta_and_A(g_k_test,f_x_0_test)
seta_1_test,A_1_test = seta_and_A(g_k_test,f_x_1_test)
seta_2_test,A_2_test = seta_and_A(g_k_test,f_x_2_test)
seta_3_test,A_3_test = seta_and_A(g_k_test,f_x_3_test)
seta_4_test,A_4_test = seta_and_A(g_k_test,f_x_4_test)
seta_5_test,A_5_test = seta_and_A(g_k_test,f_x_5_test)
seta_6_test,A_6_test = seta_and_A(g_k_test,f_x_6_test)
seta_7_test,A_7_test = seta_and_A(g_k_test,f_x_7_test)
seta_8_test,A_8_test = seta_and_A(g_k_test,f_x_8_test)
seta_9_test,A_9_test = seta_and_A(g_k_test,f_x_9_test)
```

In [50]:
```
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;
```

Classification of Matrix

```
In [51]: classification_train_test = Classification_matrix(final_f_x_test,list_label_test)
         print(classification_train_test)
```

```
[[ 949    0   12    5    1   18   22    5   10   13]
 [    0 1119    8    4    3    4    3   15   31   11]
 [    2    4  894   24    4    7    4   12    7    1]
 [    4    1   24  895    0   59    0    8   29   16]
 [    0    1   11    4  912   18    6   15    8   59]
 [    7    0    1   16    2  703   15    1   36    1]
 [    6    5   26    4    9   21  896    3    9    1]
 [    1    0   16   19    1   10    0  922    8   45]
 [   10    5   35   25    7   40   12    3  818   14]
 [    1    0    5   14   43   12    0   44   18  848]]
```

2.1 Compute (1) true positive rate, (2) error rate using the test set.

```
In [52]: tp_rate_test,error_rate_test=truepositive_and_error_rate(list_label_test,classificati
         print(tp_rate_test)
         print(error_rate_test)
```

```
89.56
10.440000000000001
```

3 Show as a Matrix.

```
In [53]: 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  |        86.650 %       |      89.560 %      |
+-----------------------+-----------------------+--------------------+
|  Error rate           |        13.350 %       |      10.440 %      |
+-----------------------+-----------------------+--------------------+
```

4 As a result, True positive rate has increased more than previous assignment.