Assignment08

November 22, 2018

1 Assignment08

1.1 Name : Choi Bowon

1.2 Student ID : 20155212

1.3 GitHub : https://github.com/ChoiBowon/Assignment

In [108]:
```
import matplotlib.pyplot as plt
import numpy as np
from scipy import signal
```

In [109]:
```
file_data_train = "mnist_train.csv"
file_data_test  = "mnist_test.csv"

h_data_train    = open(file_data_train, "r")
h_data_test     = open(file_data_test, "r")

data_train      = h_data_train.readlines() #train data
data_test       = h_data_test.readlines() #test data

h_data_train.close()
h_data_test.close()
```

In [110]:
```
size_row    = 28    # height of the image
size_col    = 28    # width of the image

num_train   = len(data_train)   # number of training images
num_test    = len(data_test)    # number of testing images
```

In [111]:
```
#
# normalize the values of the input data to be [0, 1]
#
def normalize(data):

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

    return(data_normalized)
```

In [112]:
```
#
# example of distance function between two vectors x and y
#
def distance(x, y):

    d = (x - y) ** 2
    s = np.sum(d)
    # r = np.sqrt(s)

    return(s)
```

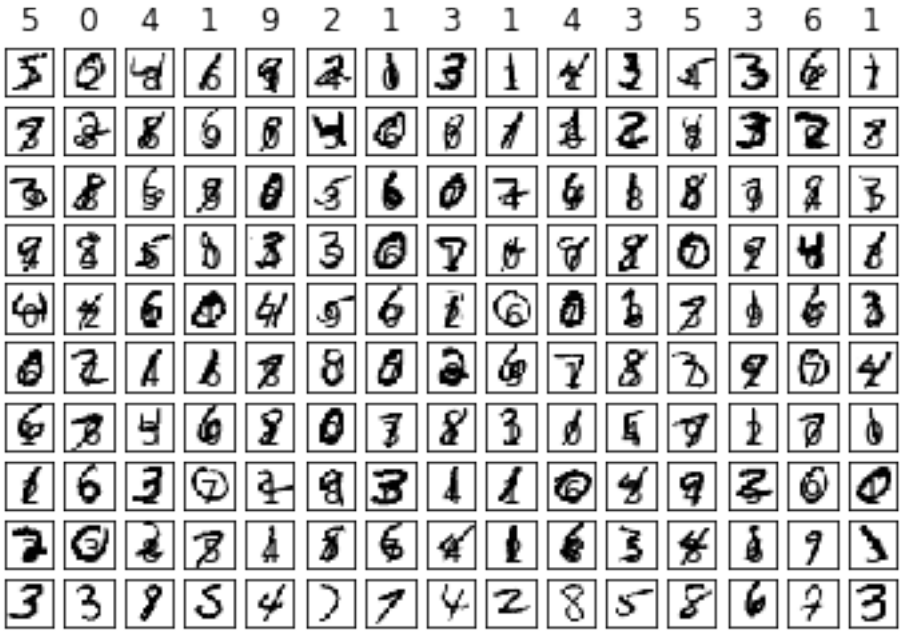
In [113]:
```
#
# make a matrix each column of which represents an images in a vector form
#
list_image_train    = np.empty((size_row * size_col, num_train), dtype=float) # trai
list_label_train    = np.empty(num_train, dtype=int) # train data label

list_image_test     = np.empty((size_row * size_col, num_test), dtype=float) #  test
list_label_test     = np.empty(num_test, dtype=int) # test data label

count = 0
```

In [114]:
```
for line in data_train:  #data train = train data

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

    list_label_train[count]     = label
    list_image_train[:, count]  = im_vector

    count += 1

count = 0
```

In [115]:
```
for line in data_test:

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

    list_label_test[count]      = label
    list_image_test[:, count]   = im_vector

    count += 1
```

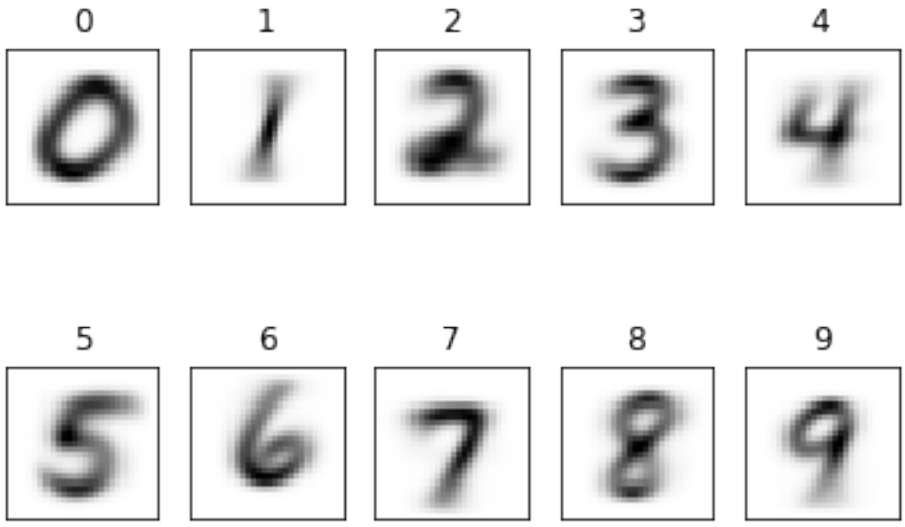
In [116]:
```
#
# plot first 150 images out of 10,000 with their labels
```

```
#
f1 = plt.figure(1)

for i in range(150):

    label       = list_label_train[i]
    im_vector   = list_image_train[:, i]
    im_matrix   = im_vector.reshape((size_row, size_col))

    plt.subplot(10, 15, i+1)
    plt.title(label)
    plt.imshow(im_matrix, cmap='Greys', interpolation='None')

    frame    = plt.gca()
    frame.axes.get_xaxis().set_visible(False)
    frame.axes.get_yaxis().set_visible(False)
#plt.show()
```



In [117]:
```
#
# plot the average image of all the images for each digit
#
f2 = plt.figure(2)

im_average  = np.zeros((size_row * size_col, 10), dtype=float)
im_count    = np.zeros(10, dtype=int)
```

```
for i in range(num_train):

    im_average[:, list_label_train[i]] += list_image_train[:, i]
    im_count[list_label_train[i]] += 1

for i in range(10):

    im_average[:, i] /= im_count[i]

    plt.subplot(2, 5, i+1)
    plt.title(i)
    plt.imshow(im_average[:,i].reshape((size_row, size_col)), cmap='Greys', interpola

    frame   = plt.gca()
    frame.axes.get_xaxis().set_visible(False)
    frame.axes.get_yaxis().set_visible(False)

plt.show()
```



1.4 Define Matrix A, image featrue matrix

```
In [118]: def build_matrix(x):  #784,6000
              col = np.shape(x)[0]
              row = np.shape(x)[1]
              matrix = np.empty((row,col), dtype=float)
              for i in range(row):
                  for j in range(col):
                      matrix[i,j] = feature_func(j+1, x[:,i])
              return matrix
```

1.5 Define Featrue function

```
In [119]: def feature_func(i, x):
              return x[i-1]
```

1.6 Define b, y values

```
In [120]: def build_y(y):
              num = np.shape(y)[0]
              b = y.reshape((num, 1))
              condlist = [b==0, b!=0]
              choicelist = [1, -1]
              return np.select(condlist, choicelist)
```

1.7 Define fucntion to approximate model parameter

```
In [121]: def approx(matrix,b):
              if np.shape(b)[0] != 1:
                  num = np.shape(b)[0]
                  b = b.reshape((num, 1))
              feature = np.shape(matrix)[1]
              theta = np.zeros((feature,1), dtype=float)
              Q,R = np.linalg.qr(list_image_train.T)
              Rsol = np.matmul(Q.T, b)
              for i in range(feature):
                  n = feature - i
                  if R[n-1, n-1] == 0:
                      theta[n-1,0] = 0
                  else:
                      rthetasum = 0
                      for j in range(feature-n):
                          l = feature-j
                          rthetasum += R[n-1, l-1]*theta[l-1,0]
                      theta[n-1, 0] = (Rsol[n-1,0] - rthetasum)/R[n-1,n-1]
              return theta
```

1.8 Plot model parameter

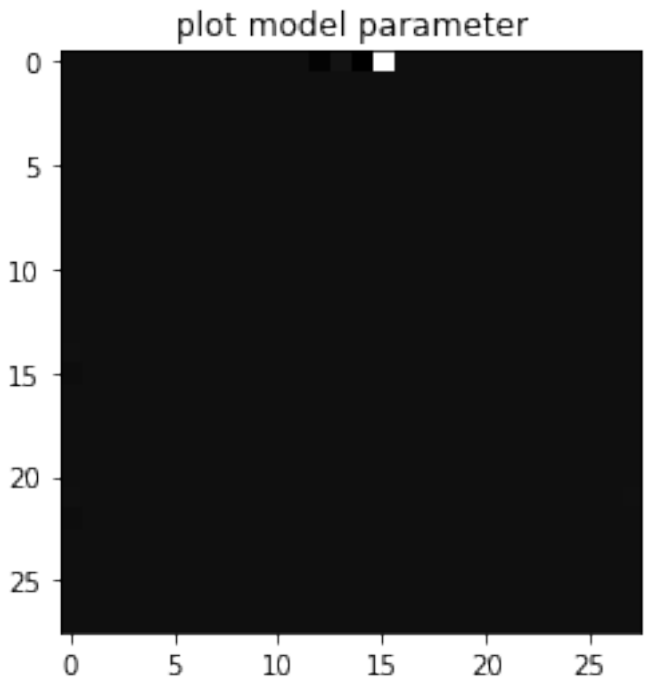
```
In [122]: def plot_theta(theta):
              data_normalized = (theta - min(theta)) / (max(theta) - min(theta))
              im_matrix = data_normalized.reshape((28,28))
              plt.imshow(im_matrix, cmap='Greys', interpolation='None')
              plt.title('plot model parameter')
              plt.show()
```

1.9 Define classifier

```
In [123]: def classifier(predict, b_test, Matrix_test):
              FN = []
              FP = []
              TN = []
              TP = []
              for i in range(0, len(predict)):
                  if(float(predict[i]) > 0):
                     if(int(b_test[i]) == 1):
                         TP.append(Matrix_test[i])
                     else:
                         FP.append(Matrix_test[i])
                  else:
                     if int(b_test[i]) == 1:
                         FN.append(Matrix_test[i])
                     else:
                         TN.append(Matrix_test[i])

              return FN,FP,TN,TP
```

1.10 Train and predict

```
In [124]: Matrix_train = build_matrix(list_image_train)
          b_train = build_y(list_label_train)
          model_parameter = approx(Matrix_train, b_train)
          plot_theta(model_parameter)
          Matrix_test = build_matrix(list_image_test)
          b_test = build_y(list_label_test)
          predict = np.matmul(Matrix_test,model_parameter)
          FN,FP,TN,TP = classifier(predict, b_test, Matrix_test)
```



1.11 Define function for getting average

```
In [125]: def average(x):
              x = np.mat(x)
              avg = np.mean(x, axis=0)

              return avg
```

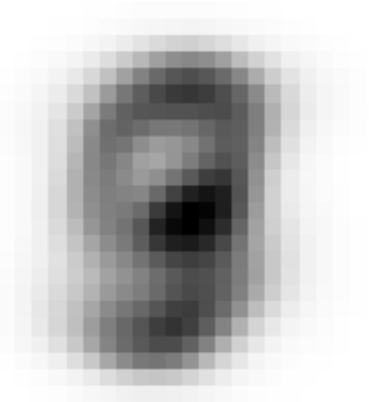
1.12 Plot average TP, FP, TN, FN

```
In [130]: plt.figure(figsize=(8,8))

          im_average = np.zeros((size_row*size_col, 10), dtype=float)
          im_count = np.zeros(10, dtype=int)

          P1 = plt.subplot(2,2,1)
          P1.set_title('FN')
          im_average = average(FN)
          plt.imshow(im_average.reshape((size_row, size_col)), cmap='Greys', interpolation='Nor
          plt.axis('off')

          P2 = plt.subplot(2,2,2)
          P2.set_title('TN')
          im_average = average(TN)
          plt.imshow(im_average.reshape((size_row, size_col)), cmap='Greys', interpolation='Nor
```

```
plt.axis('off')

P3 = plt.subplot(2,2,3)
P3.set_title('FP')
im_average = average(FP)
plt.imshow(im_average.reshape((size_row, size_col)), cmap='Greys', interpolation='Nor
plt.axis('off')


P4 = plt.subplot(2,2,4)
P1.set_title('TP')
im_average = average(TP)
plt.imshow(im_average.reshape((size_row, size_col)), cmap='Greys', interpolation='Nor
plt.axis('off')

plt.show()
```

TP

TN



FP