Assignment04

October 11, 2018

0.1 Assignmnet04

0.1.1 StudentID : 20155212

0.1.2 Name : Choi Bowon

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

0.2 Import packages for project

```
In [144]: import matplotlib.pyplot as plt
          import numpy as np
          import random
```

0.3 "mnist_test.csv" file input and initilize number of image

```
In [145]: file_data = "mnist_test.csv"
          handle_file = open(file_data, "r")
          data = handle_file.readlines()
          handle_file.close()

          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
```

0.4 Normalization for input data to be [0, 1]

```
In [146]: def normalize(data):
              data_normalized = (data - min(data)) / (max(data) - min(data))

              return (data_normalized)
```

0.5 Initialization list of images(vector) and list of label

```
In [147]: list_image = np.empty((size_row * size_col, num_image), dtype=float)
          list_label = np.empty(num_image, dtype=int)
```

```
In [148]: 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
```

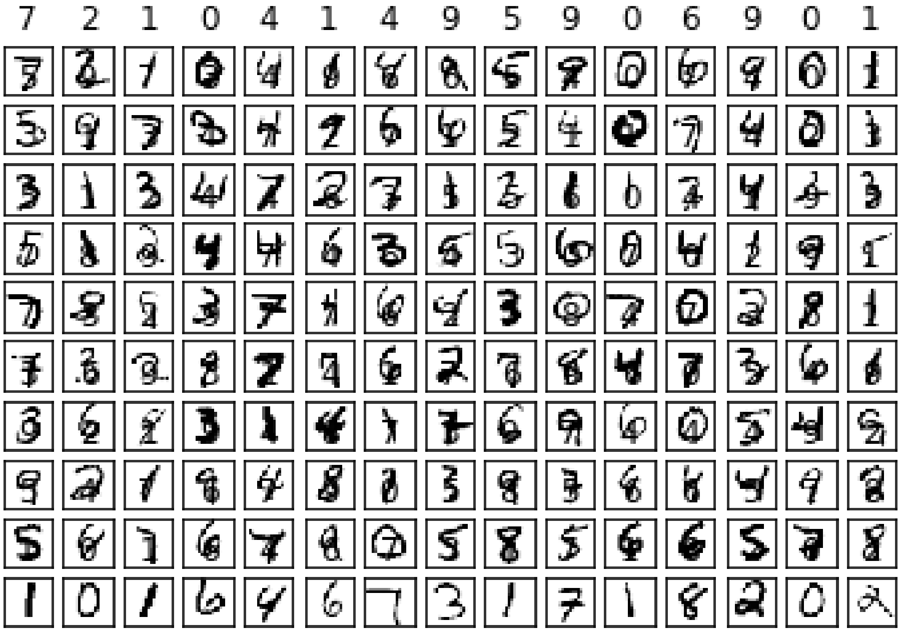
```
In [149]: f1 = plt.figure(1)
```

```
<Figure size 432x288 with 0 Axes>
```

```
In [150]: for i in range(150):
              label = list_label[i]
              im_vector = list_image[:, 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)
```



```
In [151]: f2 = plt.figure(2)
```

```
<Figure size 432x288 with 0 Axes>
```

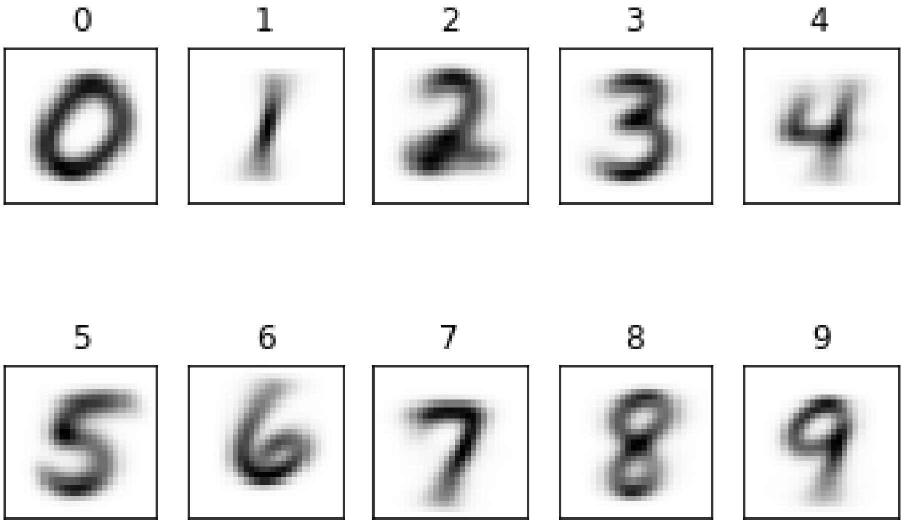
```
In [152]: im_average = np.zeros((size_row * size_col, 10), dtype=float)
          im_count = np.zeros(10, dtype=int)
```

```
In [153]: for i in range(num_image):
              im_average[:, list_label[i]] += list_image[:, i]
              im_count[list_label[i]] += 1
```

```
In [154]: 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', interpol

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



```
In [155]: plt.show()
```

1 It starts my k-means code

1.1 Input number k

```
In [156]: k = int(input("Input k number : "))
```

```
Input k number : 3
```

1.2 Compute distance between two images with vector

1.2.1 - 'd' is [k][10000] arrary

```
In [157]: def computeDistance(centroid):
              d = np.zeros((k, num_image))
              for i in range(k):
                  for j in range(num_image):
                      s = 0
                      sqrt = 0
                      for n in range(size_row*size_col):
                          s += (centroid[n][i]-list_image[n][j])**2
                          s_sqrt = np.sqrt(s)

                      d[i][j]=s_sqrt

              return d
```

1.3 Define initialiseLabel()

1.3.1 It gives random label for data

```
In [158]: label = np.empty(num_image, dtype=int)
          def initialiseLabel():
              for i in range(num_image):
                  label[i] = random.randrange(0,k)

              return label
```

1.4 Define ComputeCentroid()

1.4.1 It gives new centroid using difference with two images

```
In [159]: temp_arr = []
          centroid = np.empty((size_row * size_col, k), dtype=float)

          def computeCentroid():
              for i in range(k):
                  temp = [ j for j, x in enumerate( label ) if x == i ]

                  for j in range(size_row*size_col):
                      sum = 0
                      for h in range(len(temp)):
                          sum += list_image[j][temp[h]]
                      centroid[j][i] = sum / len(temp)


                  temp_arr.append(temp)

              print("new centroid: ", centroid)

              return centroid
```

1.5 Define assignLabel()

1.5.1 It gives new label for new clusters

```
In [160]: def assignLabel(label):
              d = computeDistance(centroid)
              a = []
              for i in range(num_image):
                  a = []
                  index = 0
                  for j in range(k):
                      a.append(d[j][i])
                      min_a = a[0]
                      for h in a:
                          if h < min_a:
                              min_a = h
                              index = a.index(h)
                  label[i] = index
              return label
```

1.6 Define computeEnergy()

1.6.1 It calculates energy according to iteraction for all image data

```
In [161]: energy = 0
          def computeEnergy(label):
              energy = 0
              for i in range(num_image):
                  for j in range(size_col*size_row):
                      energy += ((list_image[j][i] - centroid[j][label[i]])**2)**0.5
              energy /= num_image

              print("energy:",energy)
              return energy
```

1.7 Define computeAccuracy()

1.7.1 It calculates accuracy between real image label and data random label

```
In [162]: def computeAccuracy(label):
              k_acc = []
              accuracy = 0
              sum_k = 0
              for i in range(k):
                  temp_acc = []
                  temp_acc = [ j for j, x in enumerate( label ) if x == i ]
                  for h in temp_acc:
                      temp_label = [] #label      label
                      temp_label.append(list_label[h])
                  for p in range(len(temp_label)):
                      label_max = np.empty(10, dtype=int) #
                      if temp_label[p] == 0:
                          label_max[0] += 1
                      elif temp_label[p] == 1:
                          label_max[1] += 1
                      elif temp_label[p] == 2:
                          label_max[2] += 1
                      elif temp_label[p] == 3:
                          label_max[3] += 1
                      elif temp_label[p] == 4:
                          label_max[4] += 1
                      elif temp_label[p] == 5:
                          label_max[5] += 1
                      elif temp_label[p] == 6:
                          label_max[6] += 1
                      elif temp_label[p] == 7:
                          label_max[7] += 1
                      elif temp_label[p] == 8:
                          label_max[8] += 1
                      else :
                          label_max[9] += 1
                  k_acc.append(max(label_max)/len(temp_acc))


              for j in range(k):
                  sum_k += k_acc[j]
              accuracy = sum_k/num_image

              print("accuracy: ", accuracy)
```

1.8 K-means Algorithm start

```
In [ ]: initialiseLabel()
        computeCentroid()
        assignLabel(label)
        computeEnergy(label)
        computeAccuracy(label)

        CurrentEnergy = energy
        while ( energy <= float(0) ):
            computeCentroid()
            assignLabel(label)
            computeEnergy(label)
            computeAccuracy(label)
```

```
new centroid:  [[0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]
 ...
 [0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]]
energy: 117.39997893463892
accuracy:  343553332115.7848
new centroid:  [[0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]
 ...
 [0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]]
energy: 109.0237248436911
accuracy:  294215416442.79987
new centroid:  [[0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]
 ...
 [0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]]
energy: 107.55841068148764
accuracy:  273472484158.80026
new centroid:  [[0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]
 ...
 [0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]]
```