Assignment04

October 18, 2018

1 Syed Farhan Alam Zaidi

2 2018210031

3 Assignment 04

3.1 k-Means Clustering Algorithm on MNIST dataset

Github Link: https://github.com/farhan-93/assignment04.git

In this portion, the required libraries or packages are import for the execution of the code.

```
In [1]: import matplotlib.pyplot as plt
        import numpy as np
        import random
        import copy
```

The ComputerDistance function computes the distance from each point to each center. Here data is the set of image's vector, and center is the vector set of each digit from 0-9.

```
In [2]: def computeDistance(data, centers,distances):
            for i in range(k):
                    distances[:,i] = np.linalg.norm(data - centers[i], axis=1)
            return distances
```

This section computes the energy, error, or cost between the two centoroid images. We aim to minimize the energy.

$$||A||_F = [\sum_{i,j} abs(a_{i,j})^2]^{1/2}$$

```
In [4]: def computeEnergy(centers_new,centers_old):
            error = np.linalg.norm(centers_new - centers_old)
            return error
```

Computecenteroid calculate the new centers by calculating the average of the data images of each culusters

```
In [5]: def computeCentoroid(k,data, clusters):
            for i in range(k):
                    centers_new[i] = np.mean(data[clusters == i], axis=0)
            return centers_new
```

assignlabels computes the minimus distance and assign the labels according to closest centoried image's labels.

```
In [6]: def assignLable(distances,centers_new):
            clusters = np.argmin(distances, axis = 1)
            centers_old = copy.deepcopy(centers_new) ####### copy the old centers
            return clusters, centers_old
```

kMeans is the function that use all above define function it is iterative function that is handel from main function. It will execute till error=0.

```
In [7]: def kMeans(k, data, centers, centers_old, centers_new, distances,error):

            distances = computeDistance(data, centers,distances)
            clusters, centers_old = assignLable(distances,centers_new)
            centers_new = computeCentoroid(k,data, clusters)
            error = computeEnergy(centers_new,centers_old)



            return error,clusters,data,centers_new
```

This function calculates the accuracy of clustering. Here list_labels is used as ground truth while clusters labels are compared with these ground truth to calculate the accuracy.

```
In [8]: def acc(list_labels, clusters):
            accuracy=0
            for i in range(len(list_label)):
                    if list_label[i]== clusters[i]:
                         accuracy +=1
            return (accuracy/num_image)*100
```

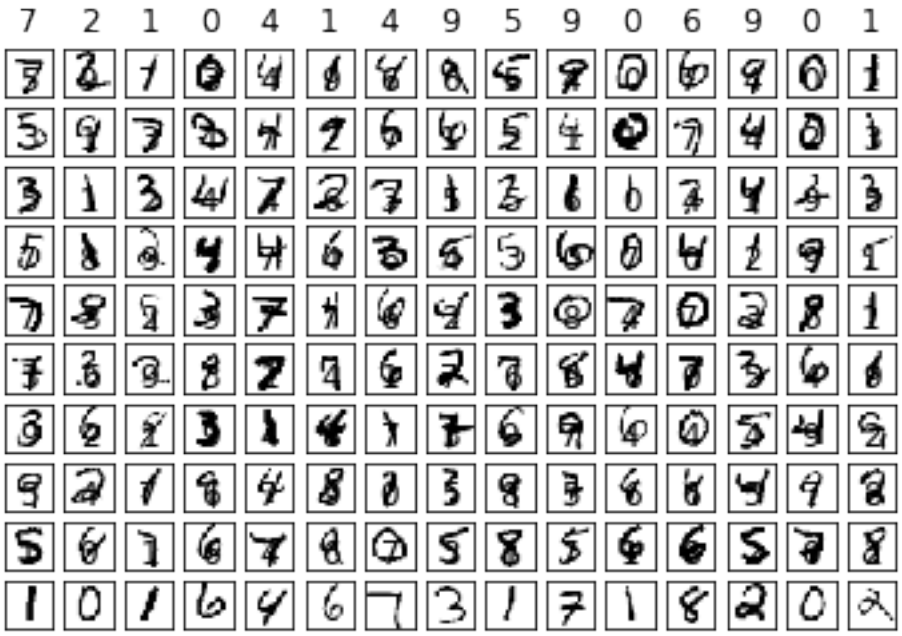
Below is the main function. It is the starting point of the kMeans clustering program's and p is the points to be generate as training data.

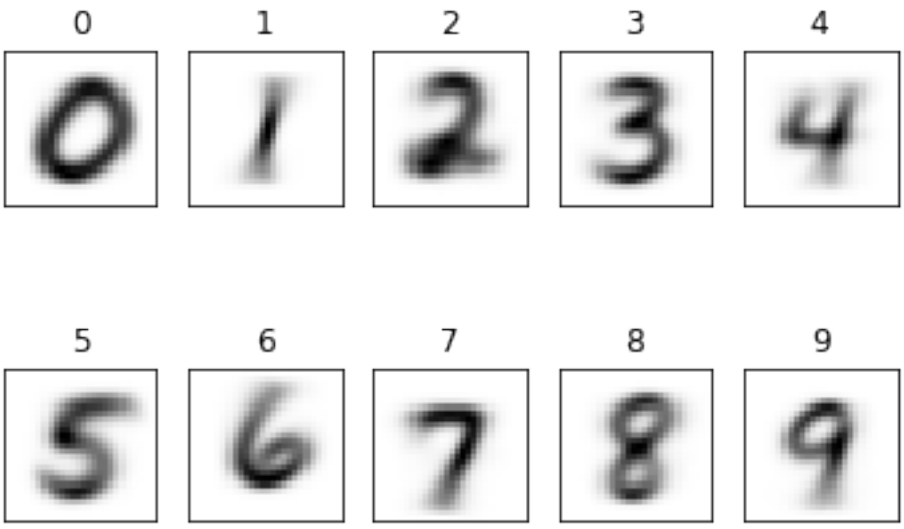
```
In [9]: if __name__== "__main__":
            k=10

            accuracy=[]
            energy=[]
            ################ opening the file
            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

#
# 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)
#

#
# make a matrix each column of which represents an image in a vector form
#
    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 (im_vector)

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

        count += 1

#
# plot first 150 images out of 10,000 with their labels
#
    f1 = plt.figure (1)

    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)

# plt.show()

#
# plot the average image of all images for each digit
#
    f2 = plt.figure(2)

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

    for i in range(num_image):

        centers[:, list_label[i]] += list_image[:, i]
        im_count[list_label[i]] += 1

    for i in range(k):

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

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

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

    plt.show()
    print(centers)

    centers=np.transpose(centers)
    data=list_image
    data=np.transpose(data)
    c = data.shape[1]
    n = data.shape[0]

    clusters = np.zeros(n)
    distances = np.zeros((n,k))
    centers_old = np.zeros(centers.shape) # stores old centers
    centers_new = copy.deepcopy(centers)



    error = computeEnergy(centers_new,centers_old)

    energy=np.append(energy,error)
```

```
print ('Results: ')
print ('Initial Energy (iteration 0):',error)

a=1
######################## Loop to iterate kMeans function. Loop will terminate when
accu= acc(list_label, clusters)
print ('Accuracy: ',accu, '%')
accuracy=np.append(accuracy,accu)
while error != 0:
    print ('############ iteration', a)

    error,clusters,data,centers_new= kMeans(k, data, centers, centers_old, centers
    print ('Centeroid: ',str(centers_new))

    print ('Energy:',error)
    energy=np.append(energy,error)
    accu= acc(list_label, clusters)
    accuracy=np.append(accuracy,accu)
    print ('Accuracy: ',accu, '%')
centers_new=np.transpose(centers_new)
data=np.transpose(data)
```



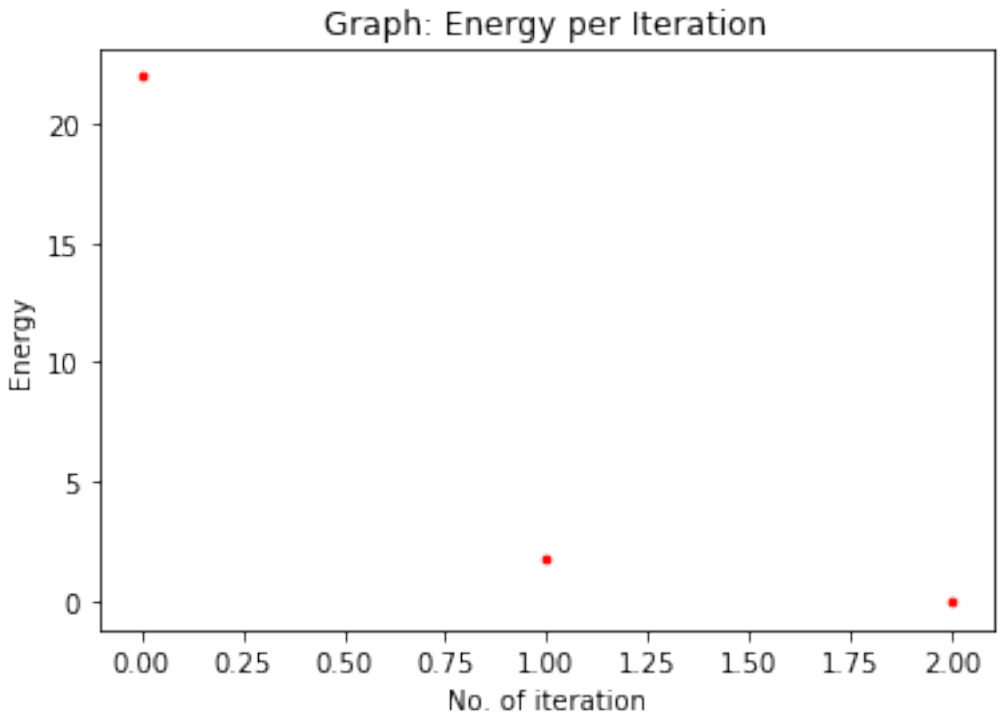


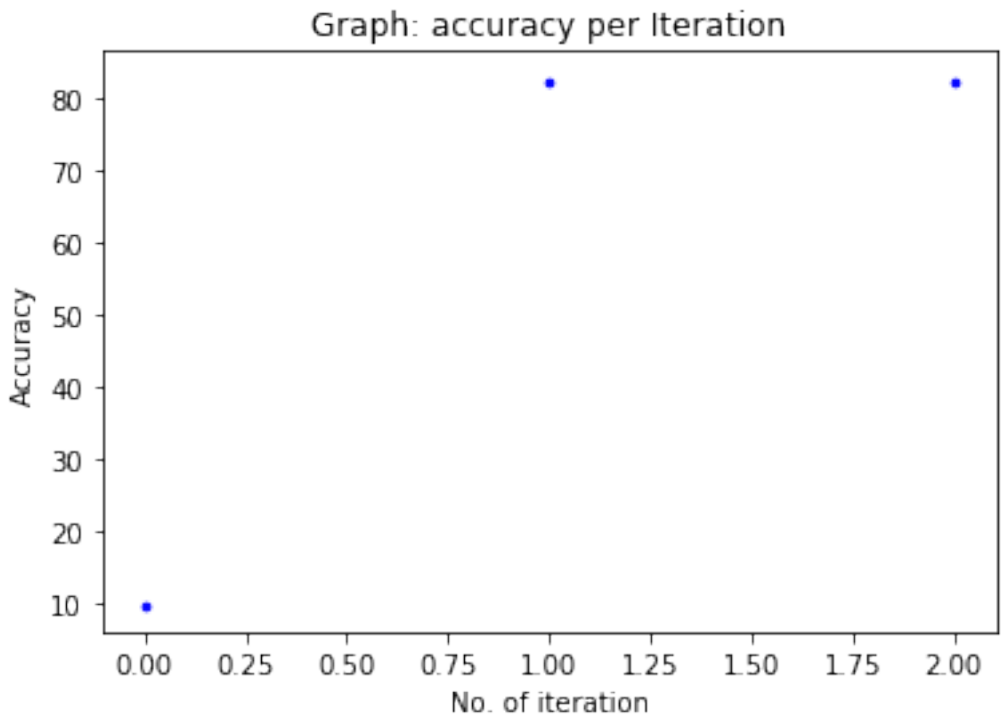

```
[[0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 ...
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]]
Results:
Initial Energy (iteration 0): 21.98706380957593
Accuracy:  9.8 %
############ iteration 1
Centeroid:  [[0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 ...
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]]
Energy: 1.7218734857744802
Accuracy:  82.28999999999999 %
############ iteration 1
Centeroid:  [[0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 ...
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]]
Energy: 0.0
```

```
Accuracy:  82.28999999999999 %
```

Following are the two plots. First plots shows the energy per iteration. and seconf plot shows the accuracy per iteration.

```
In [11]: plt.figure(5)
         for x in range(int(len(energy))):
             plt.scatter(x, energy[x], c='r', s=7)

         plt.title("Graph: Energy per Iteration");
         plt.xlabel("No. of iteration")
         plt.ylabel("Energy");
         plt.show()

         plt.figure(6)
         for x in range(int(len(accuracy))):
             plt.scatter(x, accuracy[x], c='b', s=7)

         plt.title("Graph: accuracy per Iteration");
         plt.xlabel("No. of iteration")
         plt.ylabel("Accuracy");
         plt.show()
```

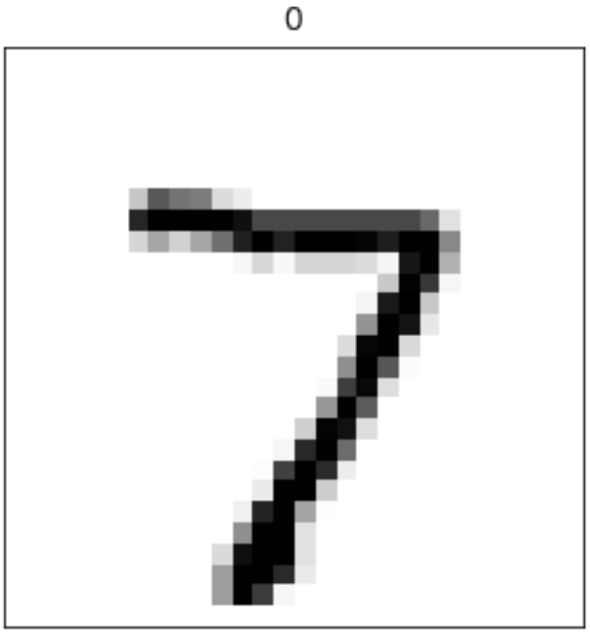




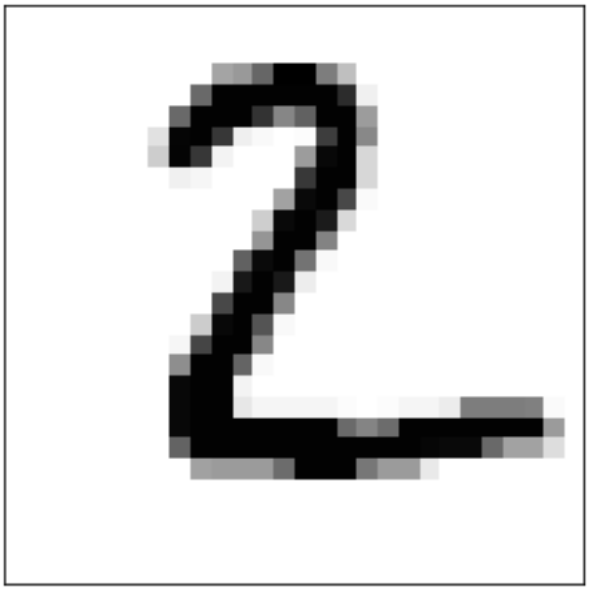

```
In [34]: for i in range (20):
                plt.figure(i+10)
                #clusters [:, i] /= im_count [i]

                #plt.plot (2, 5, clusters[i])
                print('Picture Number: ',i,'Cluster No.: ',clusters[i], 'Lable No.: ',list_lab
                plt.title (i)
                plt.imshow (data [:, i] .reshape ((size_row, size_col)), cmap = 'Greys', inter

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

        plt.show ()
```

```
Picture Number:  0 Cluster No.:  7 Lable No.:  7
Picture Number:  1 Cluster No.:  2 Lable No.:  2
Picture Number:  2 Cluster No.:  1 Lable No.:  1
Picture Number:  3 Cluster No.:  0 Lable No.:  0
Picture Number:  4 Cluster No.:  4 Lable No.:  4
Picture Number:  5 Cluster No.:  1 Lable No.:  1
Picture Number:  6 Cluster No.:  4 Lable No.:  4
Picture Number:  7 Cluster No.:  9 Lable No.:  9
Picture Number:  8 Cluster No.:  4 Lable No.:  5
```

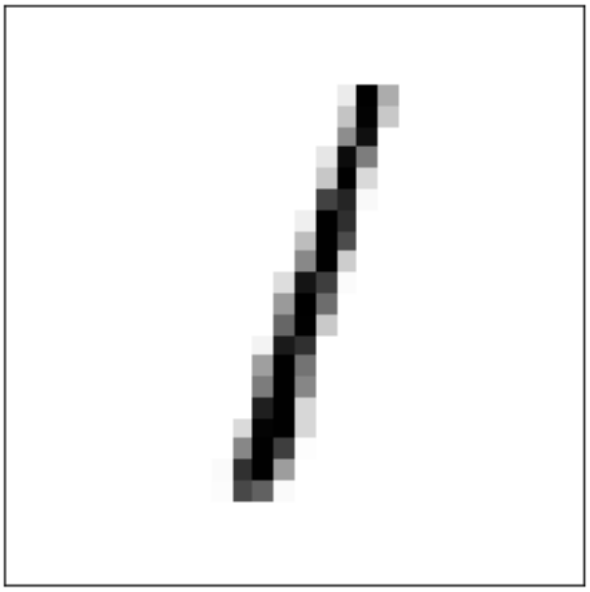
```
Picture Number:  9 Cluster No.:  9 Lable No.:  9
Picture Number:  10 Cluster No.:  0 Lable No.:  0
Picture Number:  11 Cluster No.:  6 Lable No.:  6
Picture Number:  12 Cluster No.:  9 Lable No.:  9
Picture Number:  13 Cluster No.:  0 Lable No.:  0
Picture Number:  14 Cluster No.:  1 Lable No.:  1
Picture Number:  15 Cluster No.:  5 Lable No.:  5
Picture Number:  16 Cluster No.:  9 Lable No.:  9
Picture Number:  17 Cluster No.:  7 Lable No.:  7
Picture Number:  18 Cluster No.:  3 Lable No.:  3
Picture Number:  19 Cluster No.:  4 Lable No.:  4
```



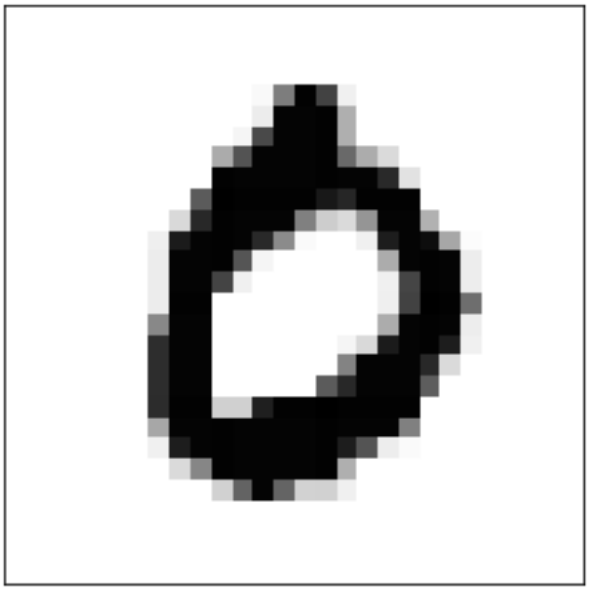
1



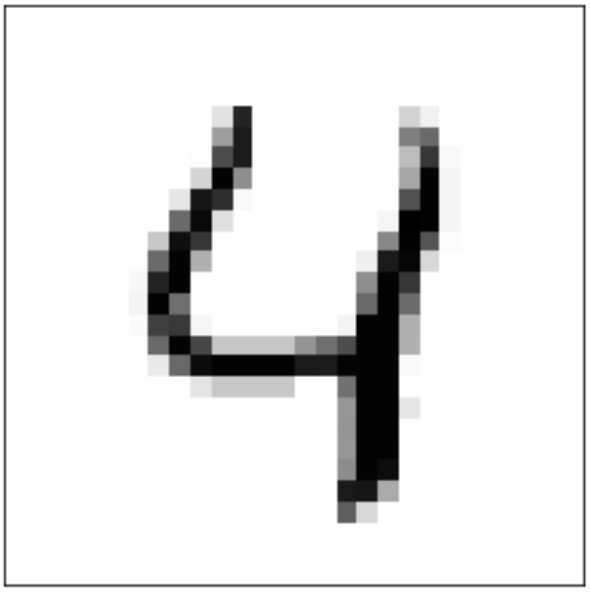
2



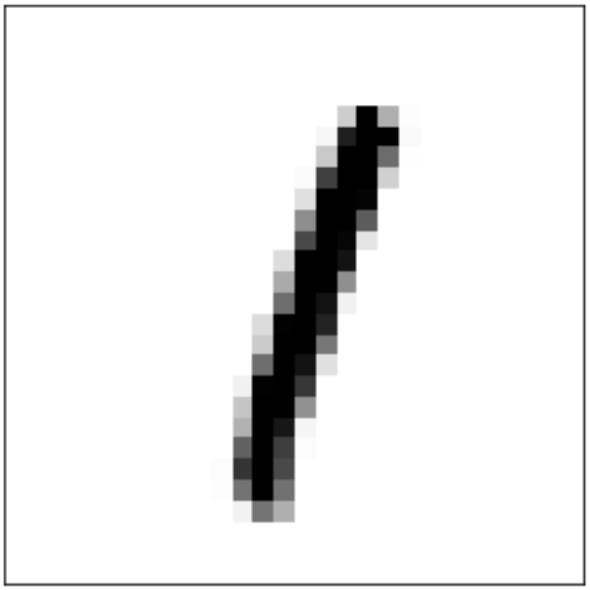
3



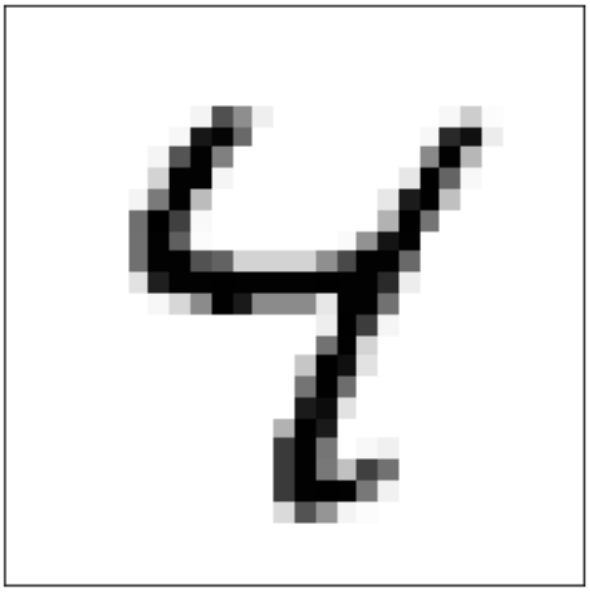
4



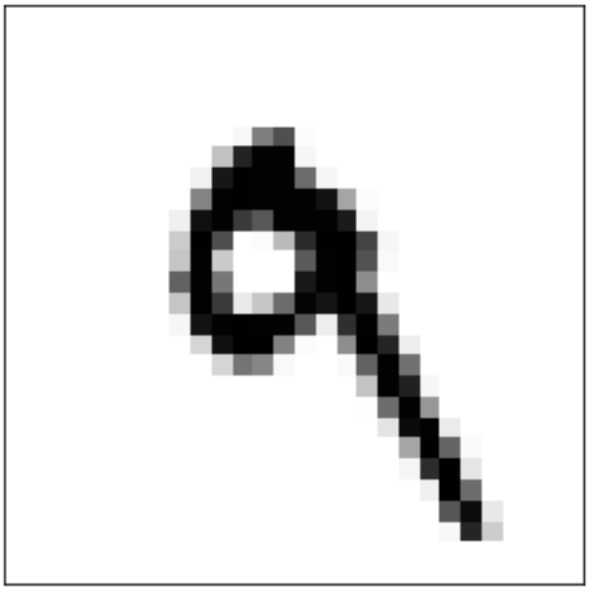
5



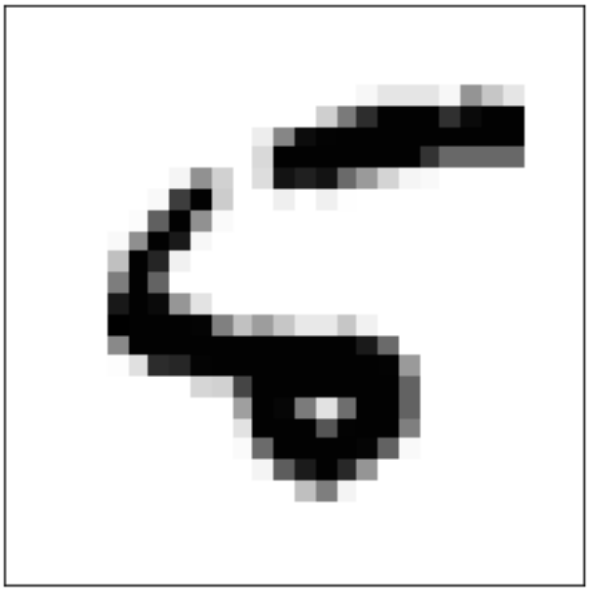
6



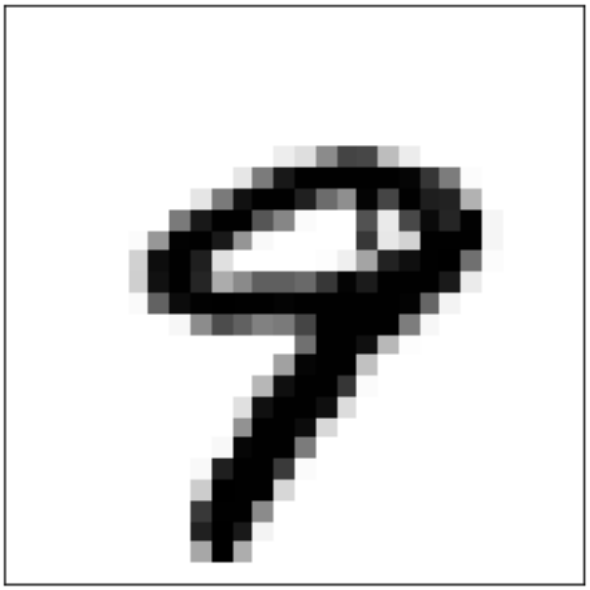
7



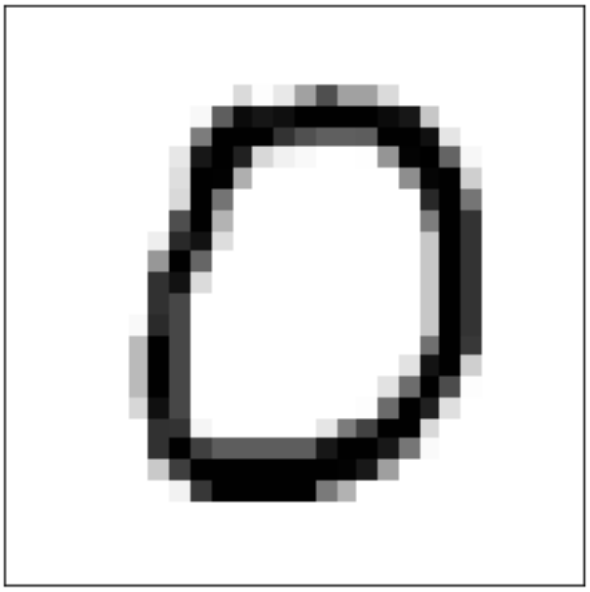
8



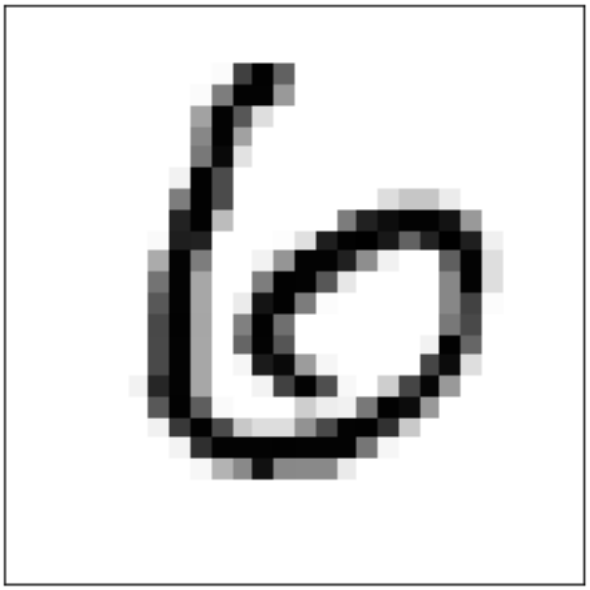
9



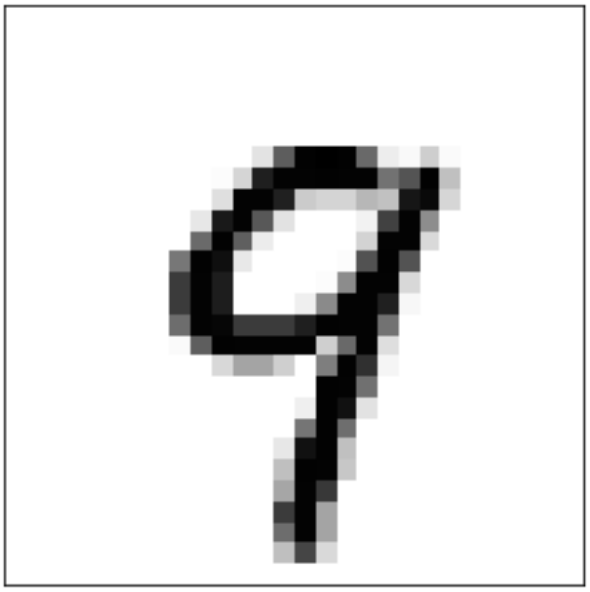
10



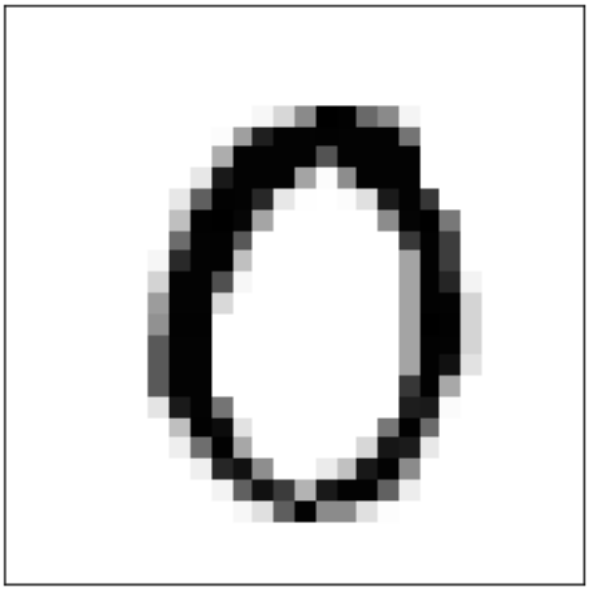
11



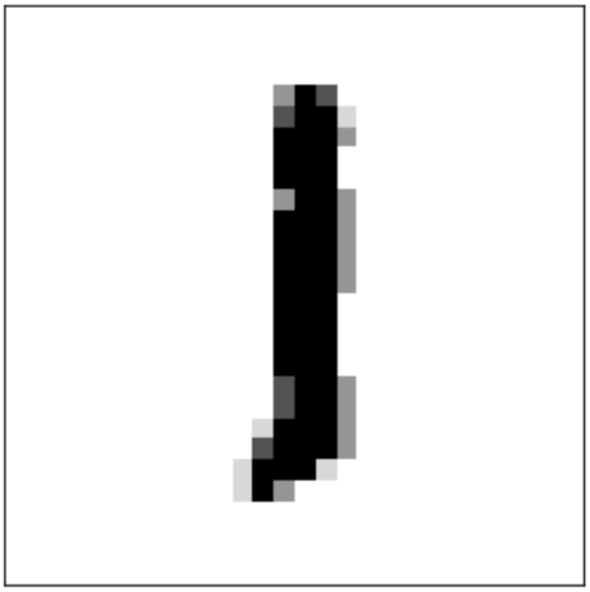
12



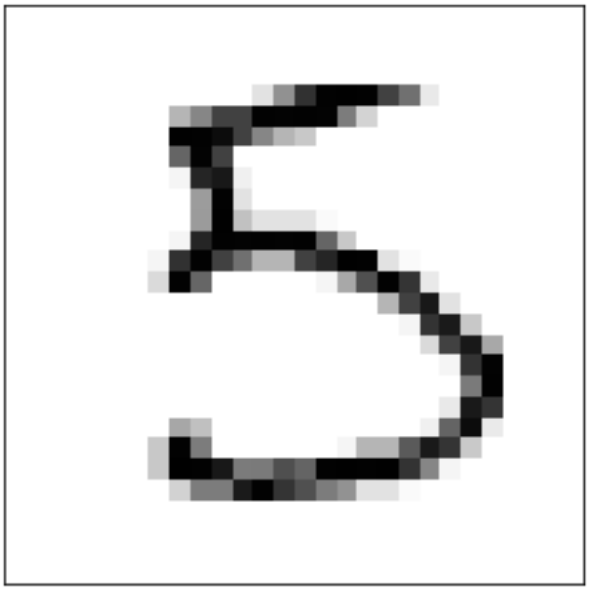
13



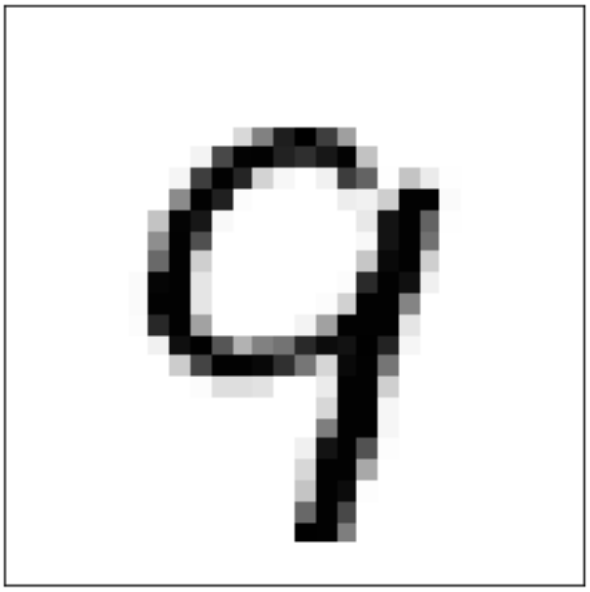
14



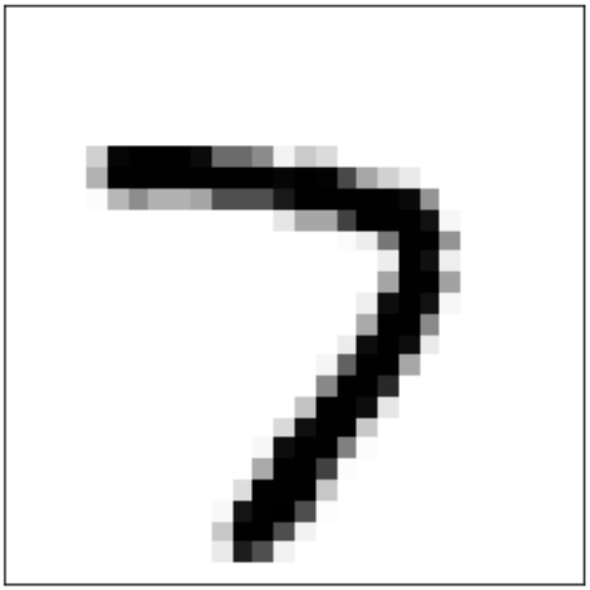
15



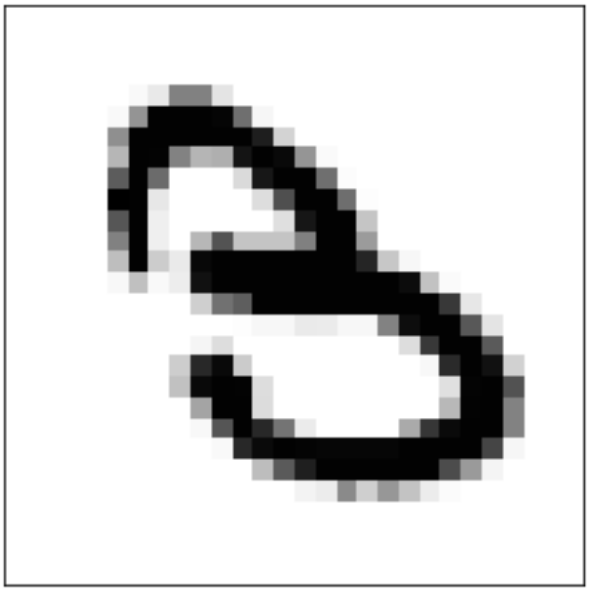
16

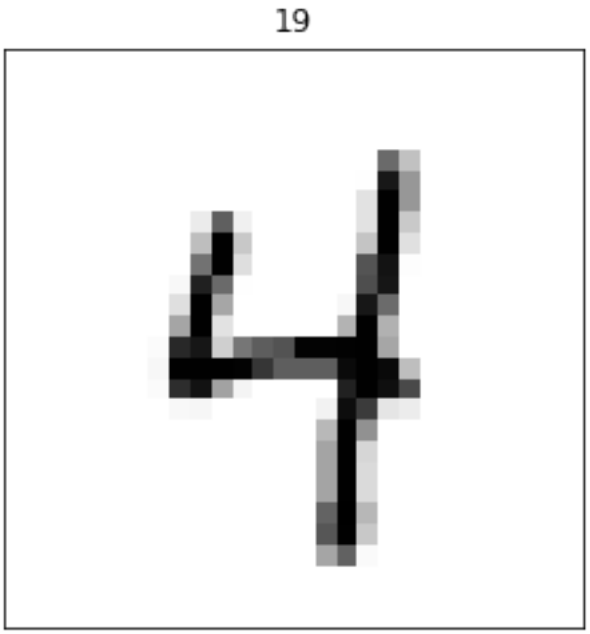


17



18





In above example the Picture no. 8 is not classified correctly. It is labeld as 5 but situated in the cluster 4. The above are the lists that shows the picture number, the cluster label and ground truth label.