Assignment 04

K-means clustering

모듈 정의

그래프를 그리기 위해 Python3 matplotlib module 을 사용합니다

accuracy 측정을 위하여 Counter를 사용합니다.

처음 랜덤한 Centroid를 정하기 위하여 random 사용합니다.

In [176]:

```
import matplotlib.pyplot as plt
import numpy as np
import random
from collections import Counter
```

1. Apply K-means clustering to MNIST training dataset with different K = 5, 10, 15, 20 and present the following results for each K.

- K-means clustering을 K = 5,10,15,20으로 MNIST training dataset에 적용하고 각 K에 대해 결과를 제시합니다.

- mnist data를 사용하기 위해서 "mnist_train.csv"를 불러옵니다.

In [177]:

```
file_data = "mnist_train.csv"
handle_file = open(file_data, "r")
data = handle_file.readlines()
handle_file.close()
```

- 필요 함수 및 변수 선언

In [178]:

```
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 images in a vector form
#
list_image  = np.empty((size_row * size_col, num_image), dtype=float)
list_label  = np.empty(num_image, dtype=int)
```

- data를 한 줄씩(feature data) 받아서 정규화하여 list_label, list_image로 구분하여 데이터 저장하고 cluster를 K 개수 만큼 생성 및 초기화

In [179]:

```
def init():
    for line in data:
        line_data   = line.split(',')
        label       = line_data[0]
        im_vector   = np.asfarray(line_data[1:])
        im_vector   = normalize(im_vector)
        global count
        list_label[count]       = label
        list_image[:, count]    = im_vector
        count += 1
```

- 모든 image 데이터(list_image) 중 k개 만큼의 centroid를 정합니다. (랜덤으로) 정하고 cluster_centroid안에 각각 넣습니다. cluster_centroid의 index는 cluster의 index와 같습니다.

In [180]:

```
# list_image안에 있는 데이터 중 k개의 centroid를 random 으로 select 한다.
def init_centroid_select_random(count,k):
    global cluster_centroid
    global list_image
    for i in range(1 , k+1):
        cluster_centroid[i] = list_image[:, random.randrange(1,count)]
```

- 각 cluster의 centroid를 정했다면 새로 들어오는 image에 대하여 centroid의 거리를 비교하여 가장 최소가 되는 거리의 cluster로 이동시킵니다. 반복되는 과정에서 cluster를 모두 비우고 채워 나갑니다. 즉, cluster를 비우고 연산 후 채워 넣고 centroid 계산하고 cluster를 비우고 연산 후 채우고 centroid를 계산하고.... 이 연산의 반복입니다.

In [181]:

```
# centroid를 정했으면 centroids 과의 거리를 계산하여 그 해당 cluster로 이동시킨다.
def image2cluster(list_image,cluster,cluster_centroid,K):
    global count
    for i in range(1,K+1):
        cluster[i] = []
    minArr = []
    for i in range(0,count):
        for j in cluster:
            minArr.append(np.linalg.norm(list_image[:, i]-cluster_centroid[j]))
        cluster_index = minArr.index(min(minArr)) + 1
        cluster[cluster_index].append((list_label[i], list_image[:, i]))
        minArr = []
```

- init_centroid_select_random 함수와 다르게 cluster 안에 있는 data의 centroid를 구하게 됩니다. (random이 아님)

$$M_k = \frac{\sum_{i \in C_k} x_i}{N}$$

M_k는 centroid 입니다. N은 해당 클러스터 안의 모든 image의 수입니다. C_k는 해당 클러스터입니다

centroid는 해당 클러스터안에 모든 image vector를 더하여 해당 클러스터의 모든 image의 수로 나눈 값입니다.

위의 연산을 구현하여 각 클러스터 별로 centroid를 구하여 cluster_centroid에 넣습니다.

In [182]:

```
def centroid_select(list_image,cluster,cluster_centroid):
    for i in cluster:
        labels,images = zip(*cluster[i])
        centroid = np.sum(np.array(images), axis = 0) / len(images)
        cluster_centroid[i] = centroid
```

- energy function의 값을 구하는 함수입니다.

$$\frac{1}{K} \sum_{k=1}^{K} \| x_i - c_{k_i} \|^2$$

- where k_i denotes the category of x_i, and c_{k_i} denotes the centroid of category x_i.

위 연산을 구현한 것으로 해당 클러스터의 image vectors와 centroid의 norm을 구하고 클러스터별 value를 모두 더하여 K만큼 나눕니다.

In [183]:

```
def get_energyfunction_val(cluster,cluster_centroid,K):
    val = 0
    for i in cluster:
        labels,images = zip(*cluster[i])
        val =  val + np.linalg.norm(images - cluster_centroid[i])
    return val / K
```

- accuracy를 구하는 함수입니다.

$$\frac{\sum_{k=1}^{K} m_k}{N}$$

- where N denotes the total number of data, and m_k denotes the number of data with majority for category k.

위 연산을 구현한 것으로 Counter의 most_common() 함수로 가장 많이 포함되어있는 category의 수를 모든 image data 로 나누어 구합니다.

1에 가까울수록 정확하다고 볼 수 있습니다.

In [184]:

```
def get_accuracy_val(cluster):
    most = 0
    labels_count = 0

    for i in cluster:
        labels,images = zip(*cluster[i])
        most = most + Counter(labels).most_common()[0][1]
        labels_count = labels_count + len(labels)
    return most/labels_count
```

---

2. Visualize K centroid images for each category.

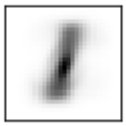
- K 중심 이미지를 시각화합니다.

- iterator는 10으로 설정합니다.

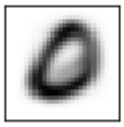
In [185]:

```
K_list = [5,10,15,20]

iterator = 10

init()

for K in K_list:
    cluster = {}
    cluster_centroid = {}

    for i in range(1,K+1):
        cluster[i] = []

    init_centroid_select_random(count,K)
    image2cluster(list_image,cluster,cluster_centroid,K)

    energy = []
    accuracy = []
    for x in range(0,iterator):
        centroid_select(list_image,cluster,cluster_centroid)
        image2cluster(list_image,cluster,cluster_centroid,K)
        energy.append(get_energyfunction_val(cluster,cluster_centroid,K))
        accuracy.append(get_accuracy_val(cluster))

    for i in cluster:
        labels,images = zip(*cluster[i])
        most = Counter(labels).most_common()[0][0]
        im_matrix   = cluster_centroid[i].reshape((size_row, size_col))
        plt.subplot(1, 5, 1)
        plt.title("K : " + str(K)+" / "+str(i)+"cluster centroid/major : "+ str(
most))
        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()
```

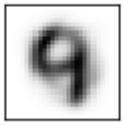
K : 5 / 1cluster centroid/major : 1



K : 5 / 2cluster centroid/major : 0

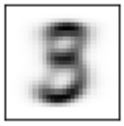


K : 5 / 3cluster centroid/major : 4

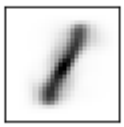


K : 5 / 4cluster centroid/major : 7

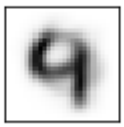
K : 5 / 5cluster centroid/major : 3



K : 10 / 1cluster centroid/major : 1



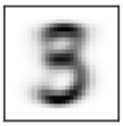
K : 10 / 2cluster centroid/major : 4



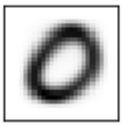
K : 10 / 3cluster centroid/major : 6

K : 10 / 4cluster centroid/major : 7

K : 10 / 5cluster centroid/major : 3



K : 10 / 6cluster centroid/major : 0



K : 10 / 7cluster centroid/major : 7

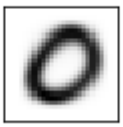
K : 10 / 8cluster centroid/major : 4

K : 10 / 9cluster centroid/major : 1

K : 10 / 10cluster centroid/major : 2

K : 15 / 1cluster centroid/major : 4

K : 15 / 2cluster centroid/major : 0



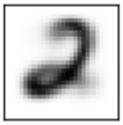
K : 15 / 3cluster centroid/major : 1

K : 15 / 4cluster centroid/major : 1

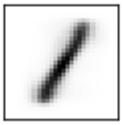
K : 15 / 5cluster centroid/major : 2

K : 15 / 6cluster centroid/major : 4

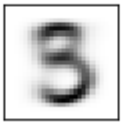
K : 15 / 7cluster centroid/major : 2



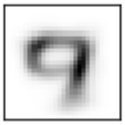
K : 15 / 8cluster centroid/major : 1



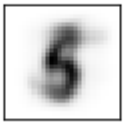
K : 15 / 9cluster centroid/major : 3



K : 15 / 10cluster centroid/major : 7

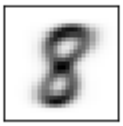


K : 15 / 11cluster centroid/major : 5



K : 15 / 12cluster centroid/major : 6

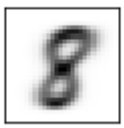
K : 15 / 13cluster centroid/major : 8



K : 15 / 14cluster centroid/major : 3

K : 15 / 15cluster centroid/major : 7

K : 20 / 1cluster centroid/major : 8



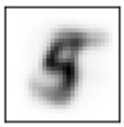
K : 20 / 2cluster centroid/major : 6

K : 20 / 3cluster centroid/major : 3

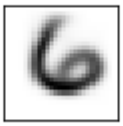
K : 20 / 4cluster centroid/major : 6

K : 20 / 5cluster centroid/major : 7

K : 20 / 6cluster centroid/major : 5



K : 20 / 7cluster centroid/major : 6

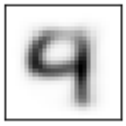


K : 20 / 8cluster centroid/major : 4

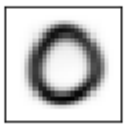
K : 20 / 9cluster centroid/major : 2

K : 20 / 10cluster centroid/major : 9

K : 20 / 11cluster centroid/major : 4

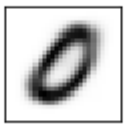


K : 20 / 12cluster centroid/major : 0

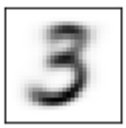


K : 20 / 13cluster centroid/major : 1

K : 20 / 14cluster centroid/major : 0



K : 20 / 15cluster centroid/major : 3

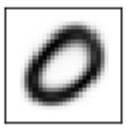


K : 20 / 16cluster centroid/major : 6

K : 20 / 17cluster centroid/major : 1

K : 20 / 18cluster centroid/major : 7

K : 20 / 19cluster centroid/major : 0



K : 20 / 20cluster centroid/major : 7

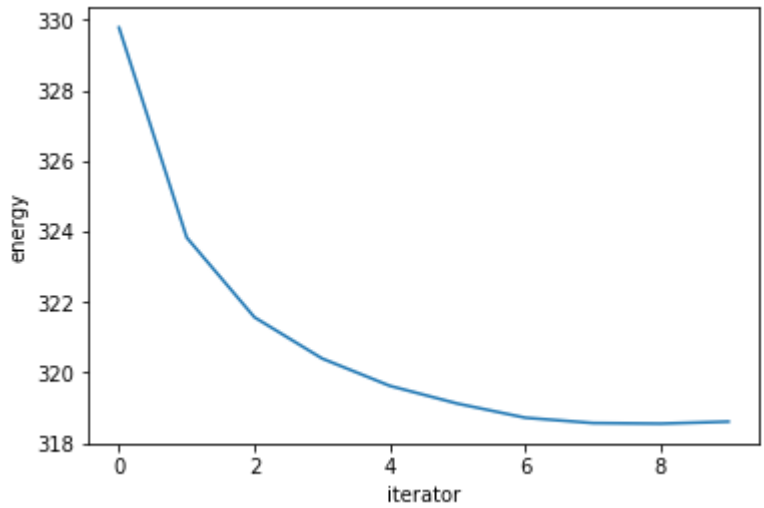
3. Plot the training energy per optimization iteration.

- 반복하며 training energy 를 plot합니다.

- (training energy) is computed on the training dataset.

In [187]:

```
plt.xlabel("iterator")
plt.ylabel("energy")
plt.plot(range(0,iterator), energy)
plt.show()
```



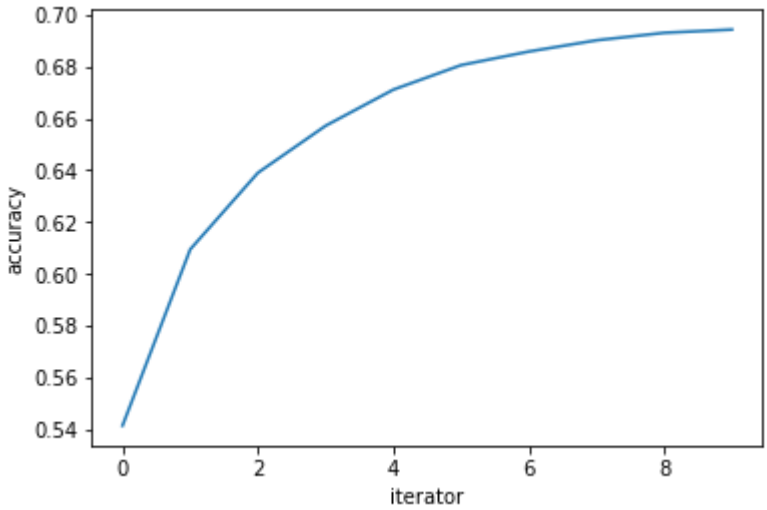
4. Plot the training accuracy per optimization iteration.

- 반복하며 training accuracy를 plot합니다.

- (training accuracy) is computed on the training dataset.

In [188]:

```
plt.xlabel("iterator")
plt.ylabel("accuracy")
plt.plot(range(0,iterator), accuracy)
plt.show()
```



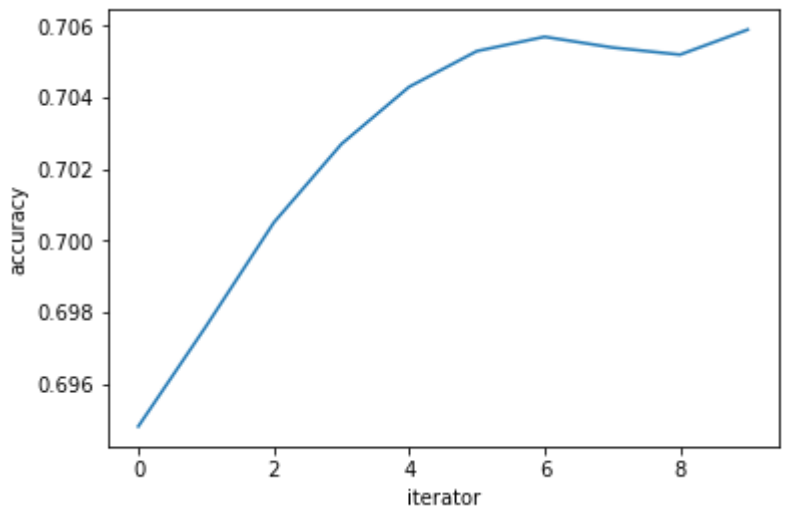
5. Plot the testing accuracy per optimization iteration.

- 반복하며 testing accuracy를 plot합니다.

- (testing accuracy) is computed on the testing dataset.

In [189]:

```
file_data   = "mnist_test.csv"
handle_file = open(file_data, "r")
data        = handle_file.readlines()
handle_file.close()

num_image   = len(data)
count       = 0

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


accuracy_test = []
image2cluster(list_image,cluster,cluster_centroid,K)

for x in range(0,iterator):
    centroid_select(list_image,cluster,cluster_centroid)
    image2cluster(list_image,cluster,cluster_centroid,K)
    accuracy_test.append(get_accuracy_val(cluster))

plt.clf()
plt.xlabel("iterator")
plt.ylabel("accuracy")
plt.plot(range(0,iterator), accuracy_test)
plt.show()
```



In [ ]: