20145271 Geum Kang Hyeon

K-means clustering

1. Apply K-means clustering to MNIST training dataset with different K = 5, 10, 15, 20 and present the following results for each K.
2. Visualize K centroid images for each category.
3. Plot the training energy per optimization iteration.
4. Plot the training accuracy per optimization iteration.
5. Plot the testing accuracy per optimization iteration.

[energy]

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

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

[accuracy]

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

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

Code Explaination

As the length of the code became longer, the functions were separated by def

First, Compile below code and initialize variables. Second, Click on the desired code set and then compile.

The process consists of four main stages.

1. After reading the data, assign a random label to each picture
2. Find a centroid vector from randomly labeled images.
3. The distance is expressed by the L2 Norm between the actual image and the centeroid vector.
4. Update labeling with the value of argmin in distance.
5. Calculate energy and accuracy according to the expression provided above
6. Repeat until the difference between the previous and current energies is less than 1 (about 0.0001%).
7. Plot K centroid images for each Category and Plot the tendency of energy and accuracy

In [357]:

```
import matplotlib.pyplot as plt
import numpy as np
import math, random

size_row        = 28    # height of the image
size_col        = 28    # width of the image

num_image       = len(data)

list_image  = np.empty((size_row * size_col, num_image), dtype=float)
list_label  = np.empty(num_image, dtype=int)

random_label = np.empty(num_image, dtype=int)
backup_label = np.empty(num_image, dtype=int)

energy = 0
energy_data = []
accuracy_data = []

accuracy = 0
new_energy = 0

# centroid image
centroid_vector = np.zeros(((size_row * size_col, k)), dtype=float)
centroid_vector_count = np.zeros(k, dtype=int)

def init():
    list_image  = np.empty((size_row * size_col, num_image), dtype=float)
    list_label  = np.empty(num_image, dtype=int)

    random_label = np.empty(num_image, dtype=int)
    backup_label = np.empty(num_image, dtype=int)

    energy = 0
    energy_data = []
    accuracy_data = []

    accuracy = 0
    new_energy = 0

    # centroid image
    centroid_vector = np.zeros(((size_row * size_col, k)), dtype=float)
    centroid_vector_count = np.zeros(k, dtype=int)

def readData(file_name):
    file_data           = file_name
    handle_file = open(file_data, "r")
    data                        = handle_file.readlines()
    handle_file.close()

    return data

def L2_distance(source, target):
    sum = sum_of_square(source, target)
    return math.sqrt(sum)

def sum_of_square(source, target):
    sum =  0
    sum += (source[:] - target[:]) ** 2
```

```
    return np.sum(sum)

def getData(k):
    count = 0
    for line in data:
        line_data   = line.split(',')
        label       = line_data[0]
        im_vector   = np.asfarray(line_data[1:])

        list_label[count]       = label
        random_label[count]     = np.random.randint(0,k,1)
        list_image[:, count]    = im_vector

        count += 1

def clustering(random_label, energy, type):
    for loop in range(100):

        accuracy = 0
        new_energy = 0

        # centroid image
        centroid_vector = np.zeros(((size_row * size_col, k)), dtype=float)
        centroid_vector_count = np.zeros(k, dtype=int)

        for i in range(num_image):
            centroid_vector[:, int(random_label[i])] += list_image[:, i]
            centroid_vector_count[int(random_label[i])] += 1

        for i in range(k):
            centroid_vector[:, i] /= centroid_vector_count[i]

        # compare image
        for i in range(num_image):
            temp = list_image[:, i]
            distance = np.zeros(k, dtype=int)

            for index in range(k):
                distance[index] = L2_distance(temp, centroid_vector[:,index])

            label = np.argmin(distance)

            new_energy += sum_of_square(temp, centroid_vector[:, label])

            backup_label[i] = label

        random_label = backup_label

        # calculate energy
        new_energy /= num_image
        energy_data.append(new_energy)


        # caculate accuracy
        accuracy_mat = np.zeros((k, 10), dtype=int)
        m_k = 0

        for i in range(num_image):
            accuracy_mat[int(random_label[i]), int(list_label[i])] += 1

        for i in range(k):
```

```
            m_k += np.amax(accuracy_mat[i])

        accuracy = m_k / num_image
        accuracy_data.append(accuracy)

        if energy == 0:
            energy = new_energy
        elif abs(new_energy - energy) < 1:
            plot(centroid_vector, type)
            break
        else:
            energy = new_energy

# type = 0 (train) / type = 1 (test)
def plot(centroid_vector, type):
    if type == 0:
        # plot image
        f1 = plt.figure(1)

        for i in range(k):
            label       = i
            im_vector   = centroid_vector[:, i]
            im_matrix   = im_vector.reshape((size_row, size_col))

            plt.subplot(1, k, i+1)
            plt.title(label)
            plt.imshow(im_matrix, cmap='Greys', interpolation='None', vmin = 0, vmax = 255)

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

        plt.show()

        plt.title("Energy")
        plt.plot(energy_data)
        plt.show()

    plt.title("Accuracy")
    plt.plot(accuracy_data)
    plt.show()
```

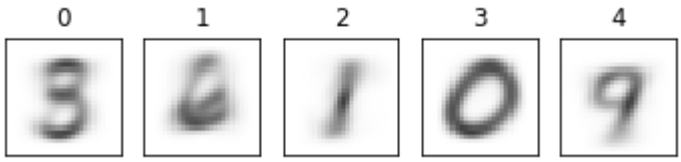
1_(1). K is 5. Run with train set.

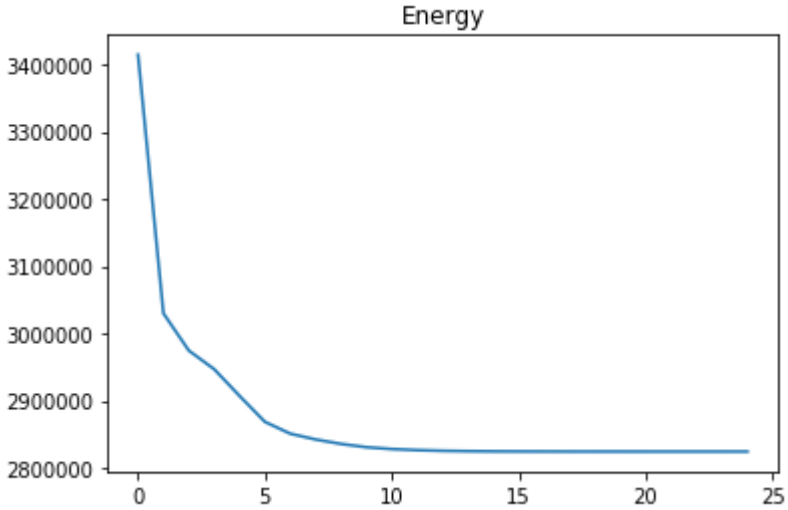
In [323]:

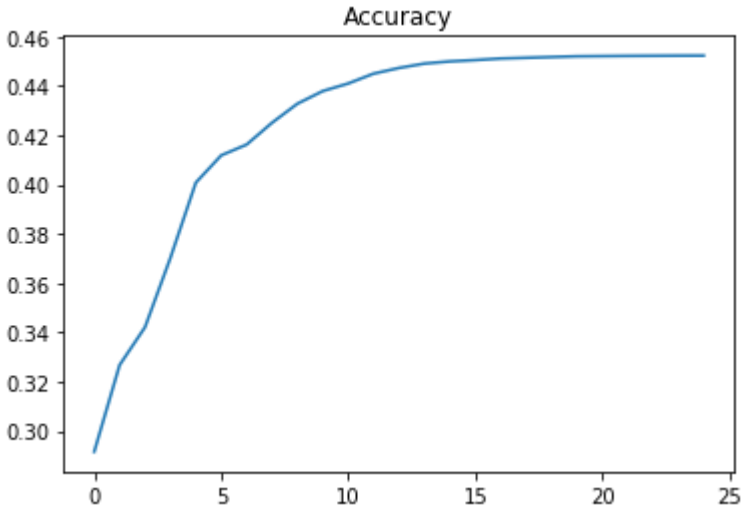
```
data = readData("mnist_train.csv")

k = 5

init()
getData(k)
clustering(random_label, energy, 0)
```







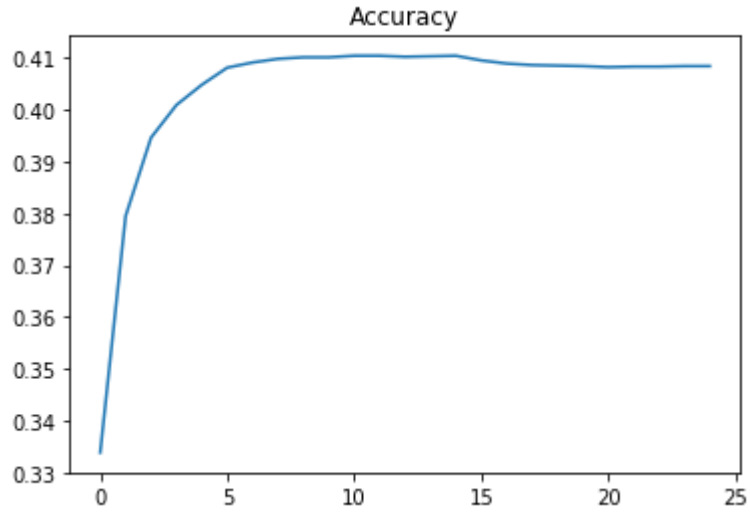
1_(2). K is 5, Run with test set.

In [358]:

```
data = readData("mnist_test.csv")

k = 5

init()
getData(k)
clustering(random_label, energy, 1)
```



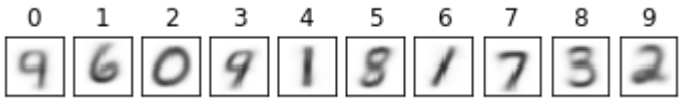
2_(1). K is 10, Run with train set.

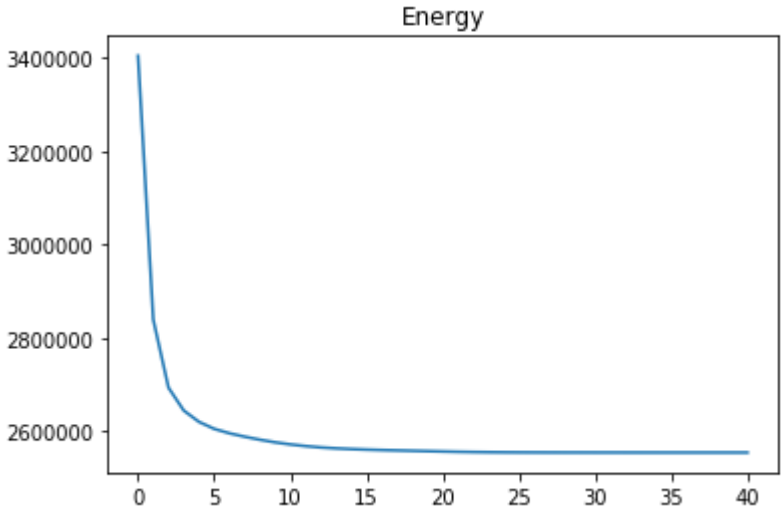
In [311]:

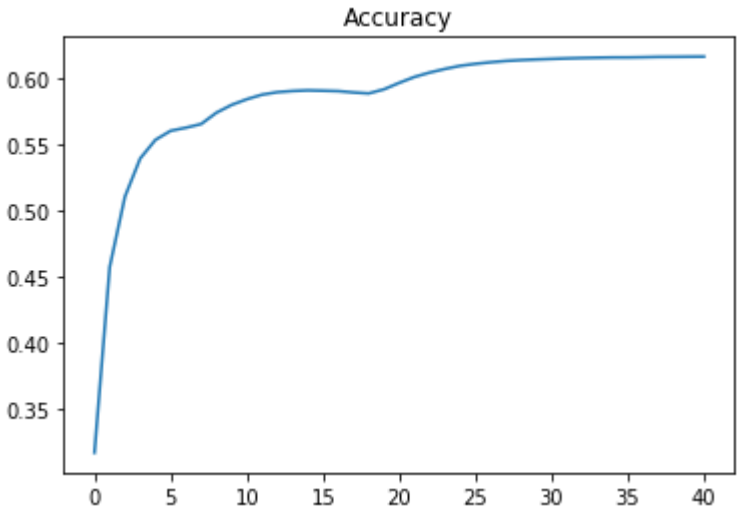
```
data = readData("mnist_train.csv")

k = 10

init()
getData(k)
clustering(random_label, energy, 0)
```







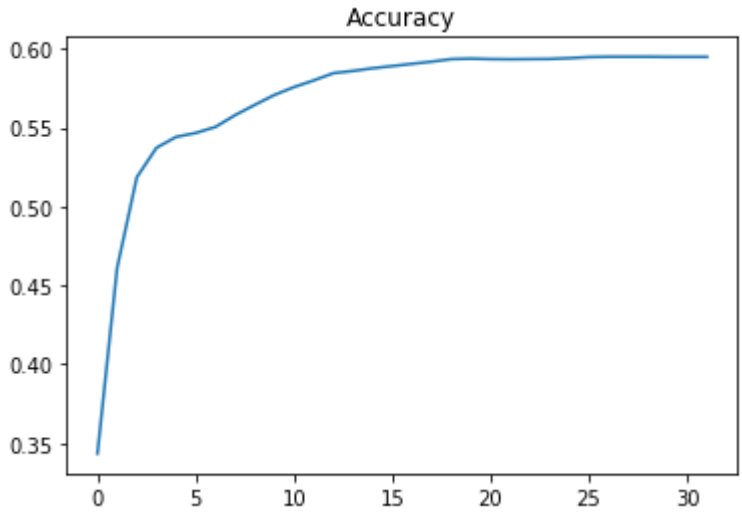
2_(2). K is 10. Run with test set.

In [334]:

```
data = readData("mnist_test.csv")

k = 10

init()
getData(k)
clustering(random_label, energy, 1)
```



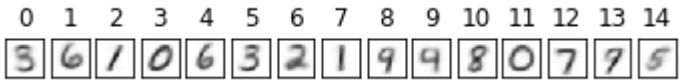
3_(1). K is 15, Run with train set.

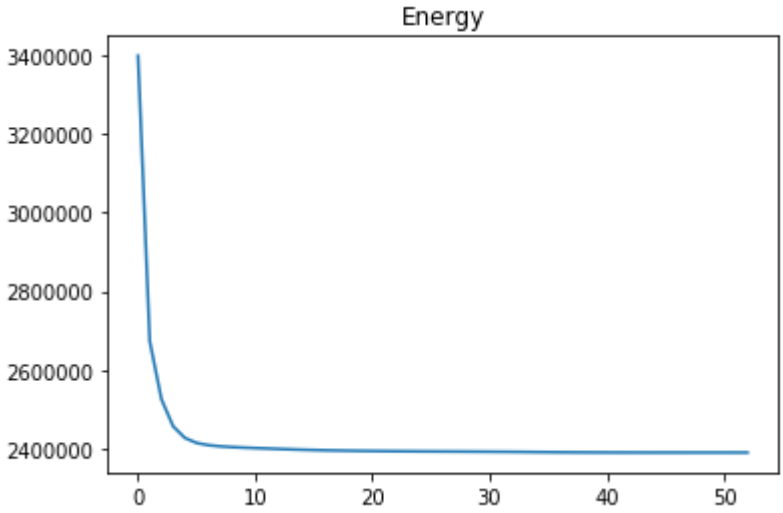
In [340]:

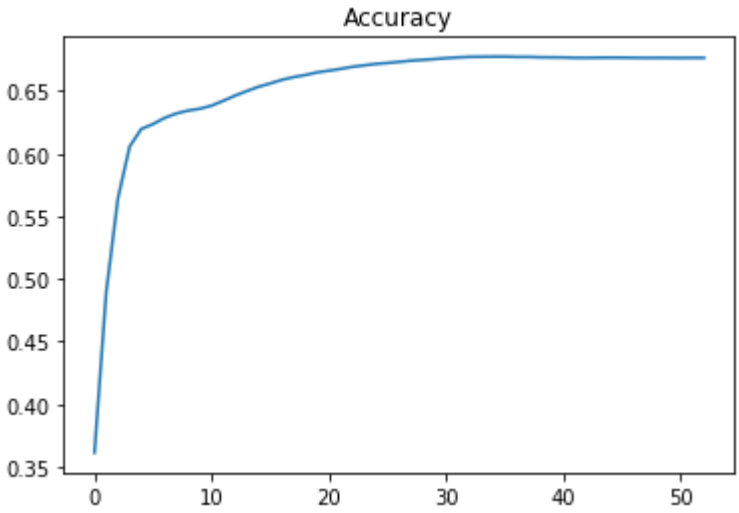
```
data = readData("mnist_train.csv")

k = 15

init()
getData(k)
clustering(random_label, energy, 0)
```







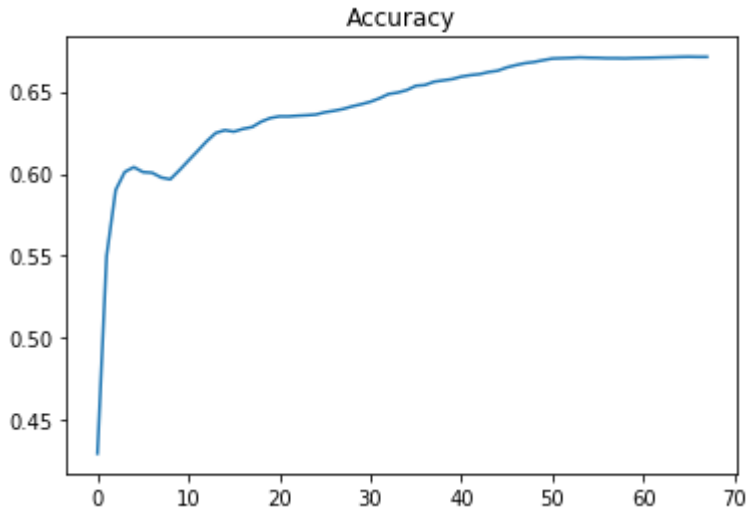
3_(2). K is 15, Run with test set.

In [344]:

```
data = readData("mnist_test.csv")

k = 15

init()
getData(k)
clustering(random_label, energy, 1)
```



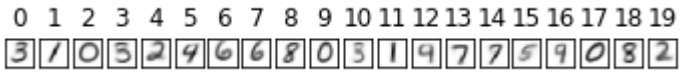
4_(1). K is 20, Run with train set.

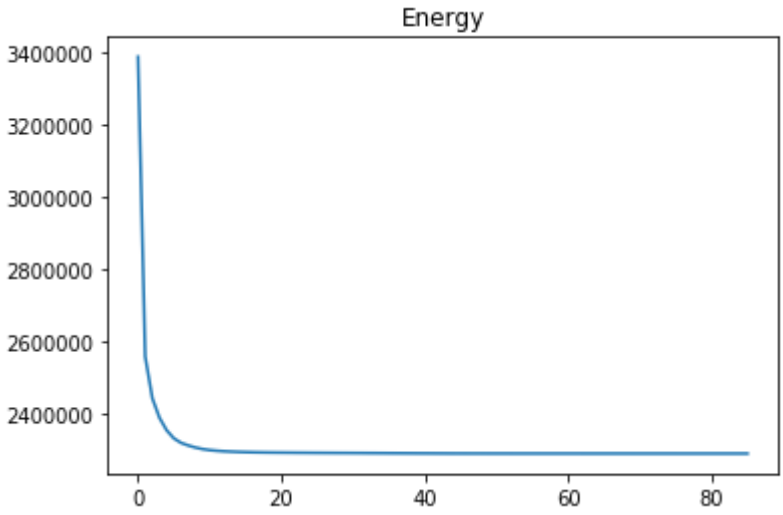
In [350]:

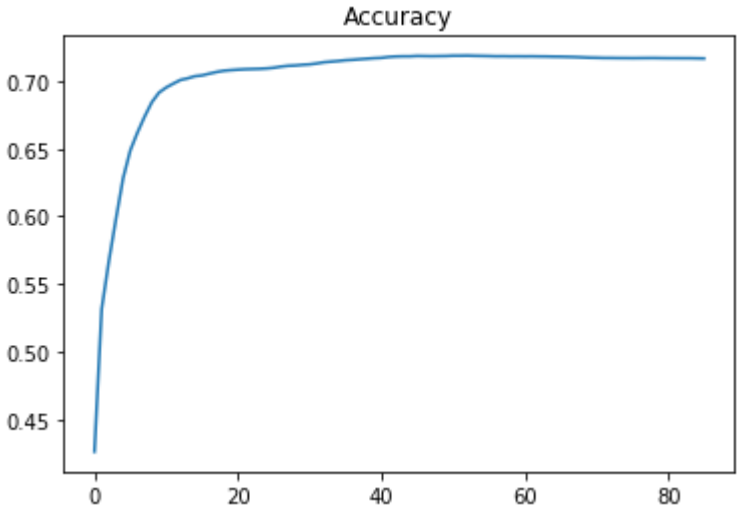
```
data = readData("mnist_train.csv")

k = 20

init()
getData(k)
clustering(random_label, energy, 0)
```







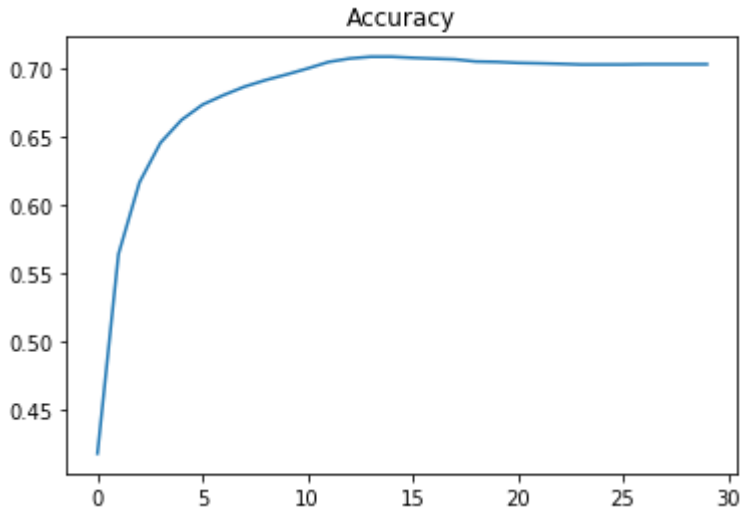
4_(2). K is 20, Run with test set.

In [354]:

```
data = readData("mnist_test.csv")

k = 20

init()
getData(k)
clustering(random_label, energy, 1)
```



We can check below result from above Calculation

- As the optimization iteration progressed, we could see that the centroid images became clearer and clearer.
- Energy and accuracy have a inverse relationship.
- The difference of energy and previous energy sharply decreases within 10 comparisons.
- Also the Acuuracy is up to 70 percent.
- The more K, the more accurate the accuracy.

I think it's a little vague way. We need to compare the propensity of distribution by pixel. but we get one representative value by the L2 Norm.