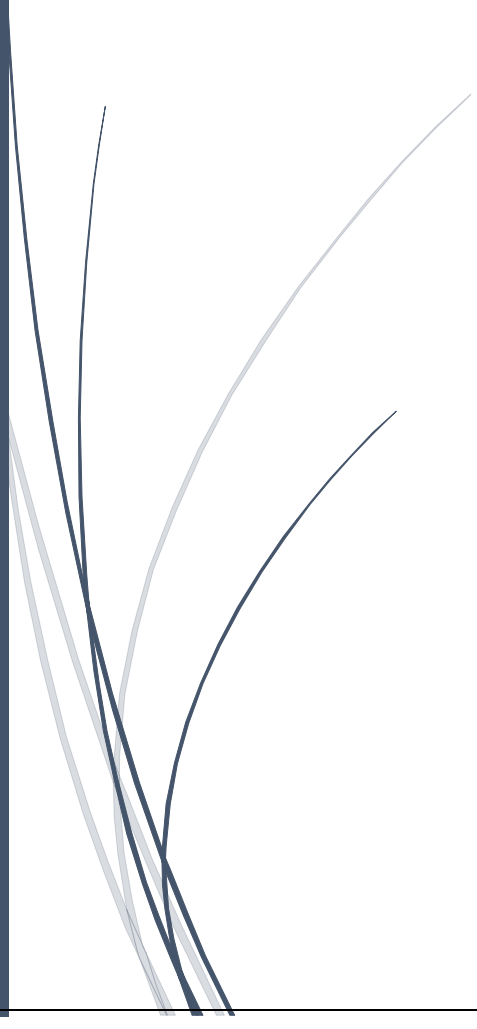
Assignment 3

K-MEANS Clustering

MNIST data set

Name : Monique Ehab
ID : 4928



Commented code snippets

- readMNIST () class is to read the MNIST data set found in the pycharm project file as (train-images.idx3-ubyte) and (train-labels.idx1-ubyte)

1)First function : initialize the root direction of the file and its type as mnist

2)Second function : reshape_to_plot : reshape in numpy is used to to extract the image array to an array of scalars and to give a new shape to an array without changing its data
So it reshape the array of data as type uni8(unsigned integer array) to be plotted

3)Third function: plot_imgs : using the reshape_to_plot function it reshapes the array of data and plots the images to a figure size of (5,5) and shows them on one plot each part as a subplot of the whole image

```
class readMNIST:
    def __init__(self, root_dir, type='mnist'):
        self.root_dir = root_dir
        self.type = type
    def reshape_to_plot(self, data):
        return data.reshape(data.shape[0], 28, 28).astype("uint8")
    def plot_imgs(self, in_data, n, random=False):
       data = np.array([d for d in in_data])
        data = self.reshape_to_plot(data)
        x1 = min(n//2, 5)
        if x1 == 0:
            x1 = 1
        y1 = (n//x1)
        x = min(x1, y1)
        y = max(x1, y1)
        fig, ax = plt.subplots(x, y, figsize=(5, 5))
        i = 0
        for j in range(x):
            for k in range(y):
                if random:
                    i = np.random.choice(range(len(data)))
                ax[j][k].set_axis_off()
                ax[j][k].imshow(data[i:i+1][0])
                i += 1
        plt.show()
```

4)Fourth function: get_train_data : it is used to read the two mnist data files , the images and the labels, assigning the name 'images' to the image file and 'labels' to the label file
-Then we define the expected data types : ubyte (for mnist data set), bytes,(>i2) > means 'big endian' and i2 means 'signed 2-byte integer, (>i4) 32-bit big-endian integer and (>f) is floating big endian
-Then using the open(join()) we join one or more path components as this method concatenates various path components
-The image and label files are read as 'binary files' using 'rb'
-Using image.seek() we set the file's current position at the offset which is zero here
-Using st.unpack we unpack the string according to the given format, it is unpacked to magic and we check if it is the expected data type or not
-Using the magic array we store the number of image of every label in nImg, image array dimension in nDim and its number of rows in nR and columns in nC and use them to calculate the number of bytes in nBytes

And we fill the images_array and labels_array with the corresponding values from the files we read and unpacked into an array

```
def get_train_data(self):
    filename = {'images': 'train-images.idx3-ubyte',
                'labels': 'train-labels.idx1-ubyte'}
    labels_array = np.array([])
    data_types = {
        0x08: ('ubyte', 'B', 1),
        0x09: ('byte', 'b', 1),
        0x0B: ('>i2', 'h', 2),
        0x0C: ('>i4', 'i', 4),
        0x0D: ('>f4', 'f', 4),
        0x0E: ('>f8', 'd', 8)}
    for name in filename.keys():
        if name == 'images':
            imagesfile = open(join(self.root_dir, filename[name]), 'rb')
        if name == 'labels':
            labelsfile = open(join(self.root_dir, filename[name]), 'rb')
   imagesfile.seek(0)
    magic = st.unpack('>4B', imagesfile.read(4))
    if(magic[0] and magic[1])or(magic[2] not in data_types):
        raise ValueError("File Format not correct")
```

```
nDim = magic[3]
imagesfile.seek(4)
nImg = st.unpack('>I', imagesfile.read(4))[0]  # num of images/labels
nR = st.unpack('>I', imagesfile.read(4))[0]  # num of rows
nC = st.unpack('>I', imagesfile.read(4))[0]  # num of columns
nBytes = nImg*nR*nC
labelsfile.seek(8)
images_array = 255 - \
    np.asarray(st.unpack('>'+'B'*nBytes,
                         imagesfile.read(nBytes))).reshape((nImg, nR, nC))
labels_array = np.asarray(
    st.unpack('>'+'B'*nImg, labelsfile.read(nImg))).reshape((nImg, 1))
labels_array = [l[0] for l in labels_array]
return images_array.reshape(60000, 28*28), labels_array, None
```

-clustering() class : it is used to implement the k-means clustering algorithm on the data set

1)First function : it is used to initialize the number of clusters and number of iterations and determine the loss

2)Second function : init_centroids : randomly initialize the centroids

3)Third function: init_clusters :initialize the array of data and arrays for each label with the data that will be associated with it

4)Fourth function : fit_data : it is used to fit the data with the closest label according to K

-We randomly initialize the centroids calling init_centroids, and initialize the iterations with zero and the predicted labels with none
-The old centroids take values from the cluster and as long as the number of iterations is still not max we calculate the distance until we reach the minimum then we assign its label to the predicted label set and assign this image or data to the predicted label then we call three functions reshape cluster, update centroids and calculate loss
-After we are done with all images and all data have been assigned to labels we calculate the accuracy

```python
import numpy as np
import copy
from tqdm import tqdm


class clustering:

    def __init__(self, n_clusters=10, max_iter=500):
        self.n_clusters = n_clusters
        self.max_iter = max_iter
        self.loss_per_iteration = []

    def init_centroids(self):
        np.random.seed(np.random.randint(0, 100000))
        self.centroids = []
        for i in range(self.n_clusters):
            rand_index = np.random.choice(range(len(self.fit_data)))
            self.centroids.append(self.fit_data[rand_index])

    def init_clusters(self):
        self.clusters = {'data': {i: [] for i in range(self.n_clusters)}}
        self.clusters['labels'] = {i: [] for i in range(self.n_clusters)}
```

```python
def fit(self, fit_data, fit_labels):
    self.fit_data = fit_data
    self.fit_labels = fit_labels
    self.predicted_labels = [None for _ in range(self.fit_data.shape[0])]
    self.init_centroids()
    self.iterations = 0
    old_centroids = [np.zeros(shape=(fit_data.shape[1],))
                     for _ in range(self.n_clusters)]
    while not self.converged(self.iterations, old_centroids, self.centroids):
        old_centroids = copy.deepcopy(self.centroids)
        self.init_clusters()
        for j, sample in tqdm(enumerate(self.fit_data)):
            min_dist = float('inf')
            for i, centroid in enumerate(self.centroids):
                dist = np.linalg.norm(sample-centroid)
                if dist < min_dist:
                    min_dist = dist
                    self.predicted_labels[j] = i
            if self.predicted_labels[j] is not None:
                self.clusters['data'][self.predicted_labels[j]].append(
                    sample)
                self.clusters['labels'][self.predicted_labels[j]].append(
                    self.fit_labels[j])
```

5)Fifth function: update_centroids(): if the cluster is empty we assign to it random centroids, else we assign the mean of all data points of the cluster

6)Sixth function : reshape_cluster(): add the new data or image to the cluster of the predicted label

7)Seventh function : converged() : It is used to check if the number of iterations if greater than maximum then we stop or if the centroids barely change then this is optimum and we stop else we continue to iterate

8)Eighth function : calculate_loss(): We calculate the difference between the predicted cluster and the centroids of the image cluster and sum them up

9)Ninth function : calculate_accuracy():We compare the predicted label with the correct label of the data if it is equal to it we add 1 to int occur then we calculate acc as number of correct predictions out of the total and sum all of accuracies of all clusters and divide by the number of formed clusters which is K

```
        self.reshape_cluster()
        self.update_centroids()
        self.calculate_loss()
        print("\nIteration:", self.iterations, 'Loss:',
              self.loss, 'Difference:', self.centroids_dist)
        self.iterations += 1
    self.calculate_accuracy()

def update_centroids(self):
    for i in range(self.n_clusters):
        cluster = self.clusters['data'][i]
        if cluster == []:
            self.centroids[i] = self.fit_data[np.random.choice(
                range(len(self.fit_data)))]
        else:
            self.centroids[i] = np.mean(
                np.vstack((self.centroids[i], cluster)), axis=0)
```

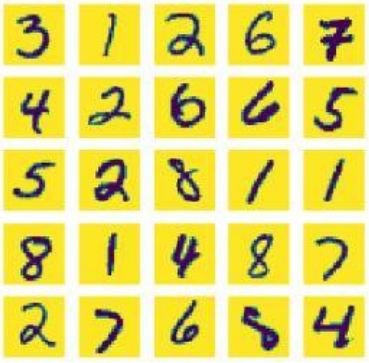
```
def reshape_cluster(self):
    for id, mat in list(self.clusters['data'].items()):
        self.clusters['data'][id] = np.array(mat)

def converged(self, iterations, centroids, updated_centroids):
    if iterations > self.max_iter:
        return True
    self.centroids_dist = np.linalg.norm(
        np.array(updated_centroids)-np.array(centroids))
    if self.centroids_dist <= 1e-10:
        print("Converged! With distance:", self.centroids_dist)
        return True
    return False

def calculate_loss(self):
    self.loss = 0
    for key, value in list(self.clusters['data'].items()):
        if value is not None:
            for v in value:
                self.loss += np.linalg.norm(v-self.centroids[key])
    self.loss_per_iteration.append(self.loss)
```

```
def calculate_accuracy(self):
    self.clusters_labels = []
    self.clusters_info = []
    self.clusters_accuracy = []
    for clust, labels in list(self.clusters['labels'].items()):
        if isinstance(labels[0], (np.ndarray)):
            labels = [l[0] for l in labels]
        occur = 0
        max_label = max(set(labels), key=labels.count)
        self.clusters_labels.append(max_label)
        for label in labels:
            if label == max_label:
                occur += 1
        acc = occur/len(list(labels))
        self.clusters_info.append(
            [max_label, occur, len(list(labels)), acc])
        self.clusters_accuracy.append(acc)
        self.accuracy = sum(self.clusters_accuracy)/self.n_clusters
    self.labels_ = []
    for i in range(len(self.predicted_labels)):
        self.labels_.append(self.clusters_labels[self.predicted_labels[i]])
    print('[cluster_label,no_occurence_of_label,total_samples_in_cluster,cluster_accuracy]', self.clusters_info)
    print('Accuracy:', self.accuracy)
```

Use Jupyter Notebook to run the python classes, plot the graphs and show the clusters:

```
In [2]: import os
        os.chdir('kmeans')
        from readMNIST import readMNIST
        from clustering import clustering
        import matplotlib.pyplot as plt
        import numpy as np
```

```
In [3]: data_reader = readMNIST('../')
        tr_data, tr_class_labels, tr_subclass_labels = data_reader.get_train_data()
        data_reader.plot_imgs(tr_data,25,True)
```



Sample image read from the file

For k=5:

1)The iterations=34

The accuracy=51.6%

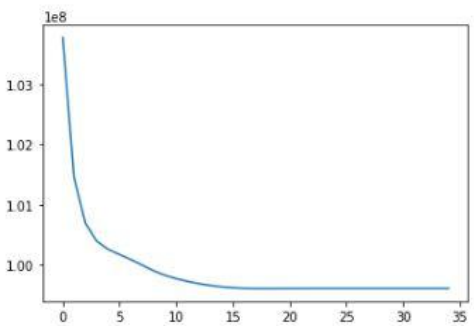
```
In [14]: kmeans = KMeans(n_clusters=5,max_iter=200)
         kmeans.fit(tr_data,tr_class_labels)
```

```
49166it [00:07, 8901.95it/s]
50114it [00:07, 9067.68it/s]
51024it [00:07, 8787.90it/s]
52009it [00:07, 9081.53it/s]
52959it [00:07, 9202.99it/s]
53884it [00:07, 8897.70it/s]
54856it [00:08, 9129.26it/s]
55774it [00:08, 9036.25it/s]
56799it [00:08, 9368.93it/s]
57742it [00:08, 9114.79it/s]
58660it [00:08, 7499.77it/s]
60000it [00:08, 6802.33it/s]

Iteration: 34 Loss: 99607708.88768701 Difference: 1.3537957889800064e-08
Converged! With distance: 2.5278650079517237e-12
[cluster_label,no_occurence_of_label,total_samples_in_cluster,cluster_accuracy] [[3, 5041, 12555, 0.40151334129828753], [6, 5
070, 10804, 0.4692706405035172], [7, 5366, 17307, 0.3100479574738545], [0, 4977, 5415, 0.9191135734072022], [1, 6684, 13919,
0.48020691141605]]
Accuracy: 0.5160304848197823
```

2)The graph between iterations and loss (it never increases)

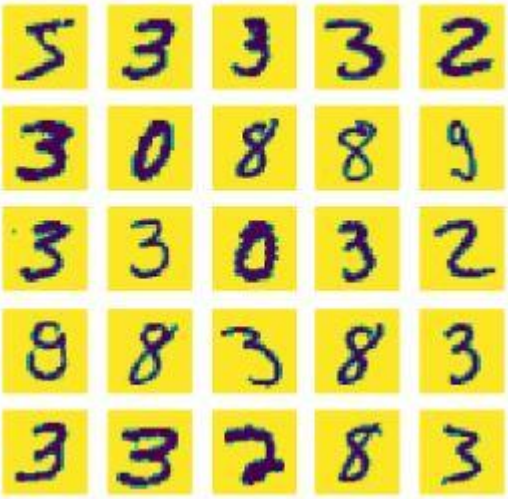
```
In [15]: print('Loss')
         plt.plot(range(kmeans.iterations),kmeans.loss_per_iteration)
         plt.show()
```

```
Loss
```

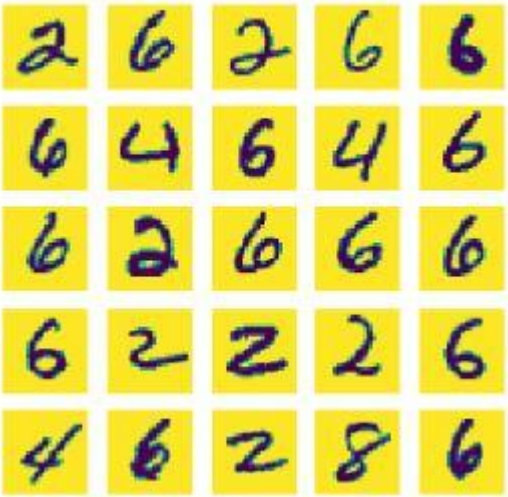


3) The mean images and formed clusters

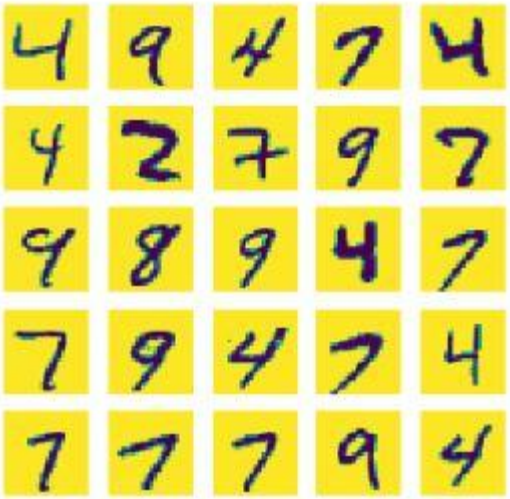
Cluster: 0 Label: 3



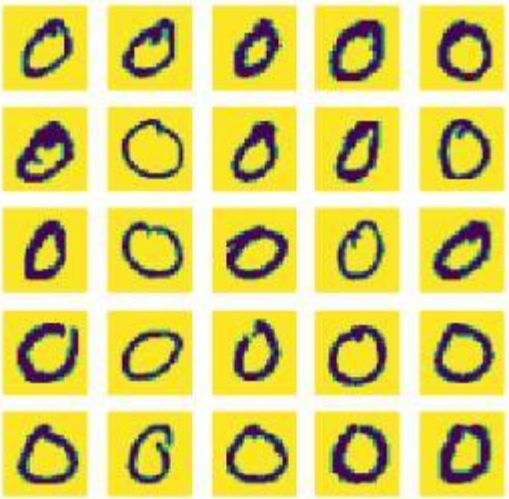
Cluster: 1 Label: 6



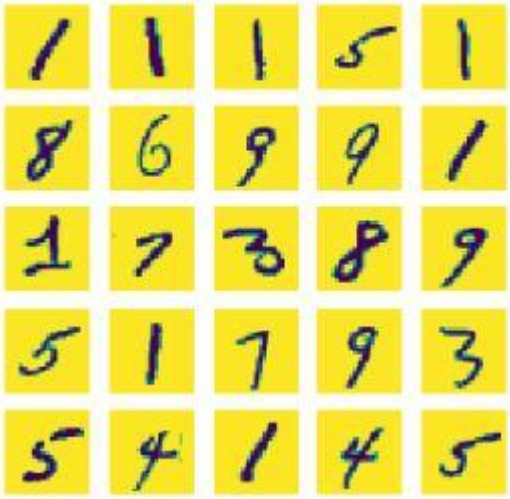
Cluster: 2 Label: 7



Cluster: 3 Label: 0



Cluster: 4 Label: 1



For k=10:

1) The iterations :57

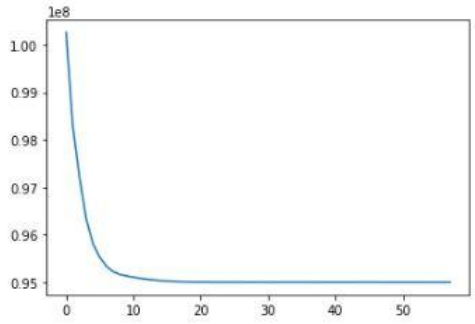
The accuracy :59.7 %

```
In [4]: kmeans = KMeans(n_clusters=10,max_iter=200)
        kmeans.fit(tr_data,tr_class_labels)

60000it [00:08, 6722.16it/s]

Iteration: 55 Loss: 94996446.30948892 Difference: 0.7263730114755078

60000it [00:17, 3456.81it/s]

Iteration: 56 Loss: 94996446.30949122 Difference: 0.00016893739485082052

60000it [00:15, 3903.72it/s]

Iteration: 57 Loss: 94996446.30949119 Difference: 4.537448184066472e-08
Converged! With distance: 1.2951790840647125e-11
[cluster_label,no_occurence_of_label,total_samples_in_cluster,cluster_accuracy] [[9, 2626, 5986, 0.43869027731373206], [8, 33
04, 6376, 0.5181932245922208], [4, 2044, 4767, 0.4287812041116006], [0, 2827, 3065, 0.9223491027732463], [3, 3996, 7832, 0.51
02145045965271], [7, 1729, 6022, 0.28711391564264366], [6, 4881, 9275, 0.5262533692722372], [7, 3156, 3399, 0.928508384819064
4], [1, 6576, 9957, 0.6604398915335945], [0, 2515, 3321, 0.7573020174646191]]
Accuracy: 0.5977845892119487
```

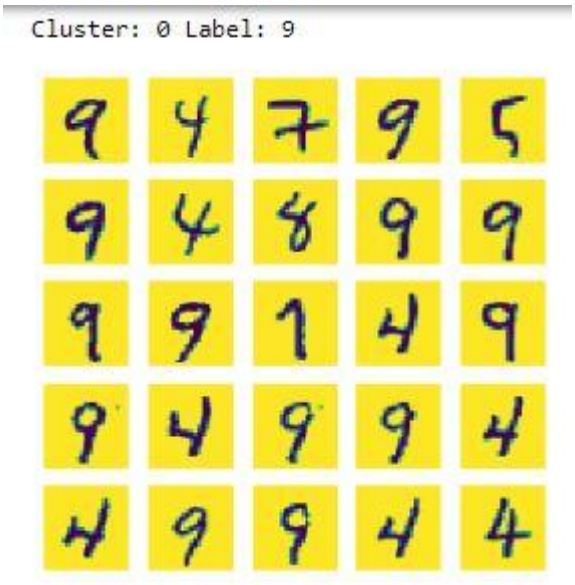
2)The graph between iterations and loss (it never increases)

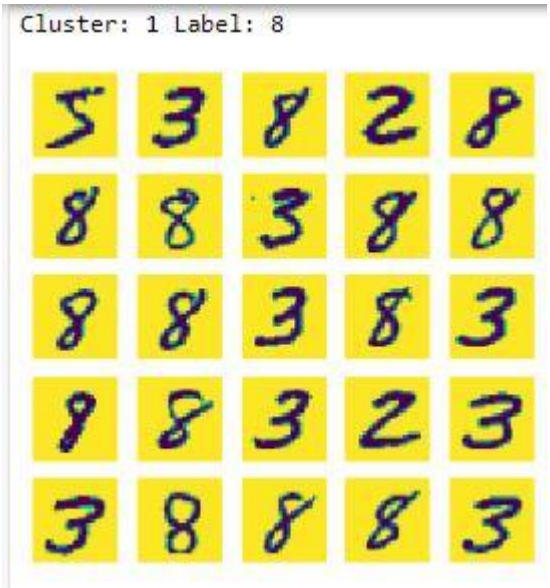
```
In [5]: print('Loss')
        plt.plot(range(kmeans.iterations),kmeans.loss_per_iteration)
        plt.show()
```

Loss

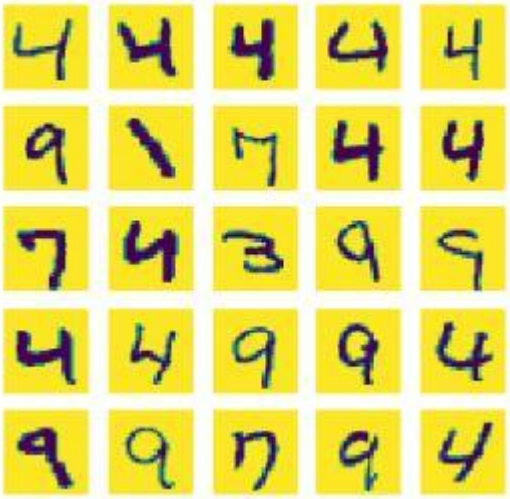


3)The mean images and the formed clusters :

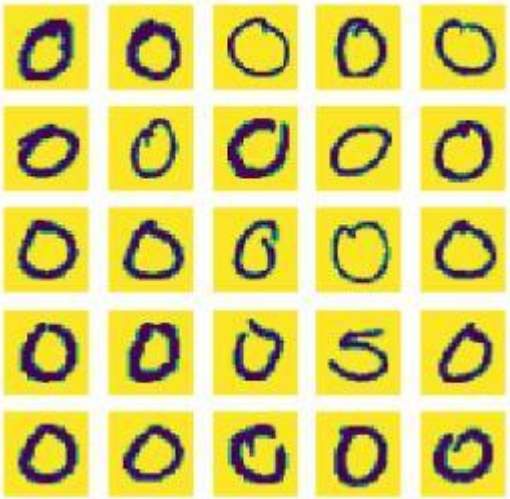




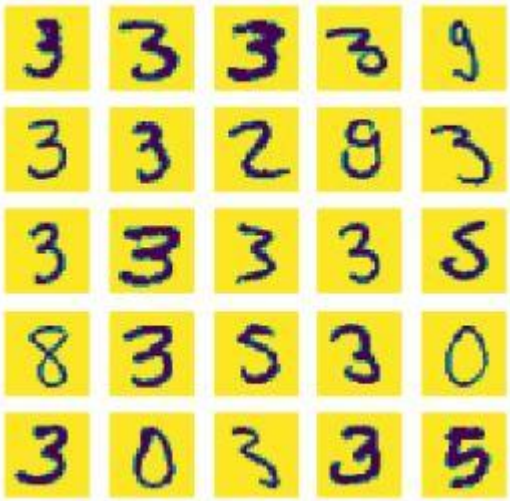
Cluster: 2 Label: 4



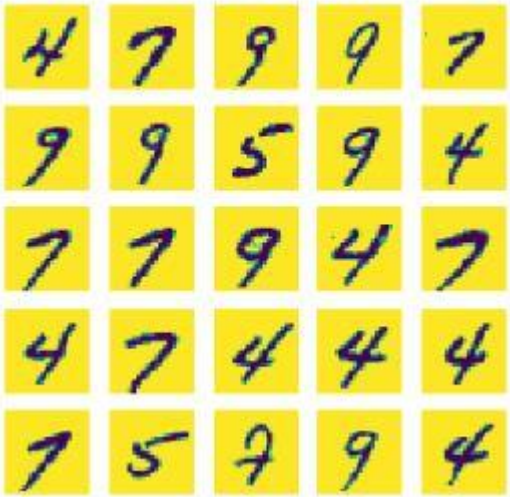
Cluster: 3 Label: 0



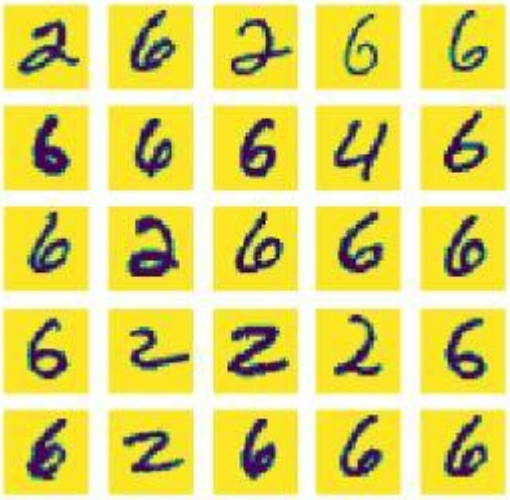
Cluster: 4 Label: 3



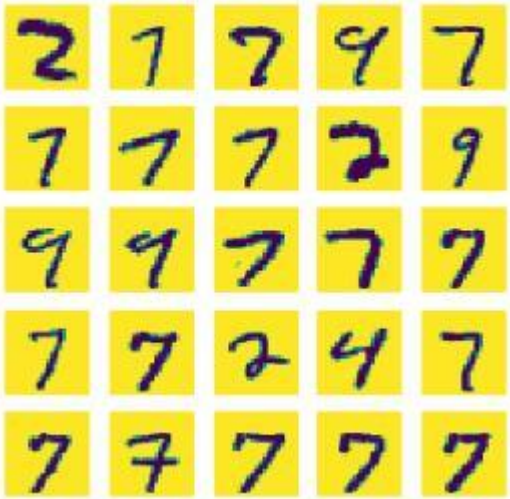
Cluster: 5 Label: 7

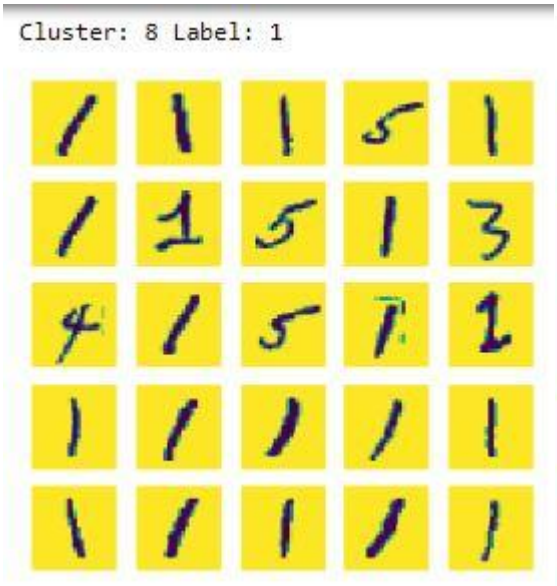


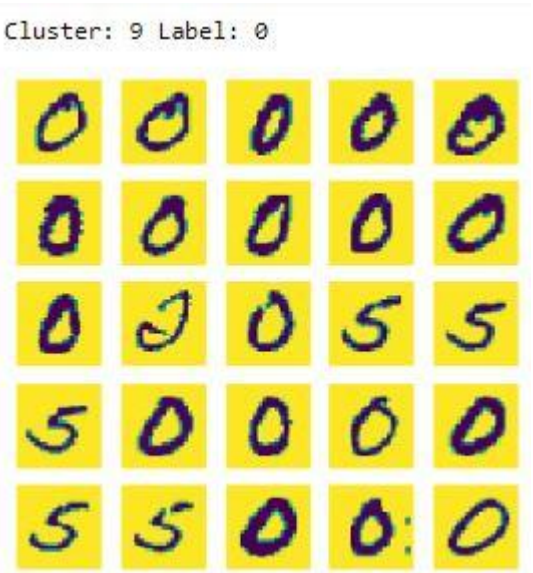
Cluster: 6 Label: 6



Cluster: 7 Label: 7







For k=20 :

1)The iterations =20

The accuracy =73.5%

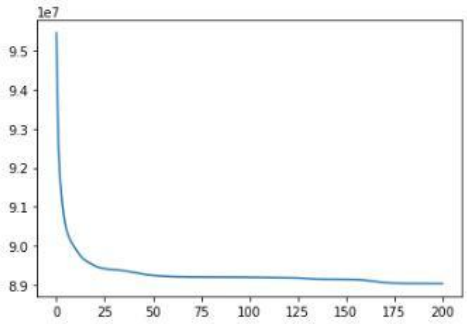
```
In [10]: kmeans = KMeans(n_clusters=20,max_iter=200)
         kmeans.fit(tr_data,tr_class_labels)
```

```
57610it [00:26, 2588.02it/s]
57877it [00:26, 2612.04it/s]
58139it [00:26, 2598.81it/s]
58400it [00:26, 2571.35it/s]
58658it [00:26, 2528.49it/s]
58913it [00:26, 2534.86it/s]
59186it [00:26, 2590.37it/s]
59450it [00:27, 2605.02it/s]
60000it [00:27, 2206.08it/s]

Iteration: 200 Loss: 89029701.2148025 Difference: 2.6236205679944646
[cluster_label,no_occurence_of_label,total_samples_in_cluster,cluster_accuracy] [[1, 2994, 3517, 0.8512937162354279], [6, 296
7, 3383, 0.8770322199231452], [8, 2581, 3170, 0.8141955835962145], [3, 2441, 3427, 0.7122847971987161], [5, 1740, 3117, 0.558
2290664100096], [4, 1916, 3028, 0.6327608982826949], [9, 1398, 3561, 0.392586352148273], [8, 1660, 4044, 0.410484668644906],
[6, 2390, 2572, 0.9292379471228616], [9, 1691, 3576, 0.4728747203579418], [4, 1161, 2557, 0.45404771216269063], [2, 2231, 241
1, 0.9253421816673579], [7, 2009, 3078, 0.6526965562053282], [1, 3618, 4346, 0.8324896456511734], [0, 1783, 1919, 0.929129755
0807712], [0, 1561, 1742, 0.8960964408725602], [2, 2574, 2729, 0.9432026383290583], [3, 1756, 3489, 0.5032960733734595], [7,
2143, 2261, 0.9478107032286599], [0, 1987, 2073, 0.958514230583695 1]]
Accuracy: 0.7346802953537472
```

2)The graph between iterations and loss (it never increases)

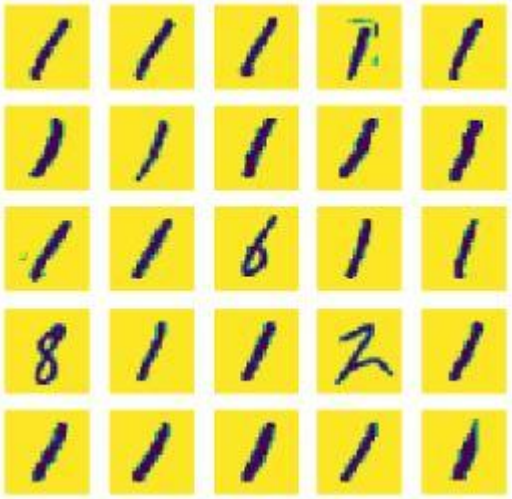
```
In [11]: print('Loss')
         plt.plot(range(kmeans.iterations),kmeans.loss_per_iteration)
         plt.show()
```

```
Loss
```

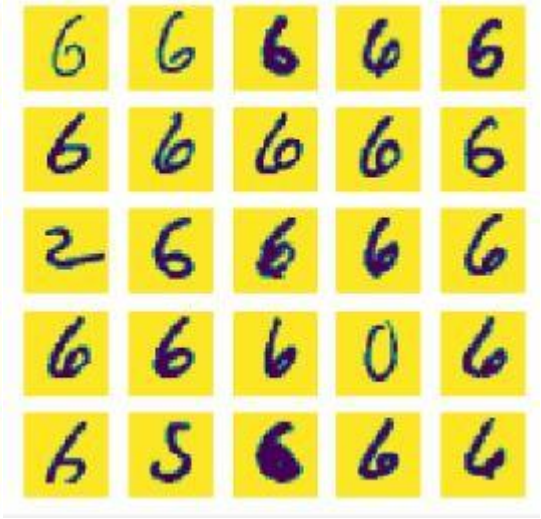


3) The mean images and the formed clusters

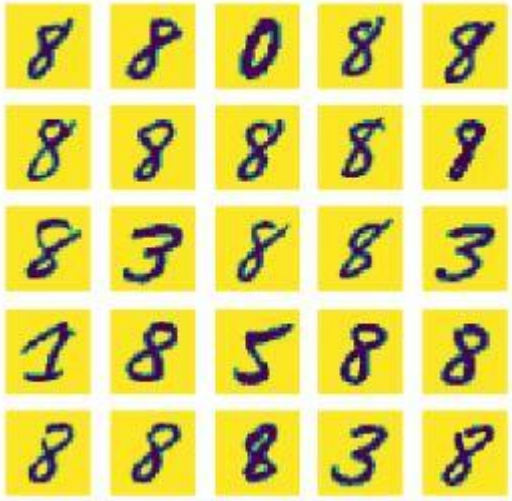
Cluster: 0 Label: 1



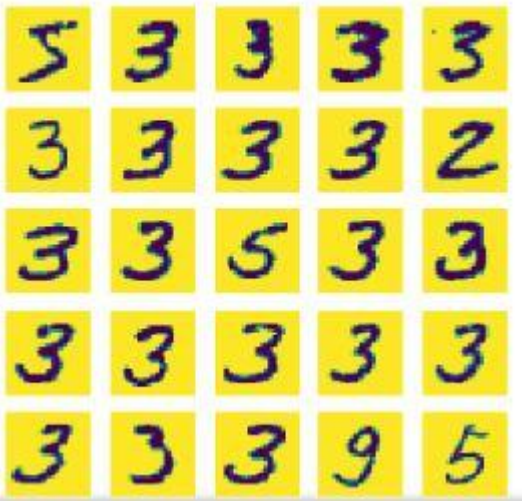
Cluster: 1 Label: 6



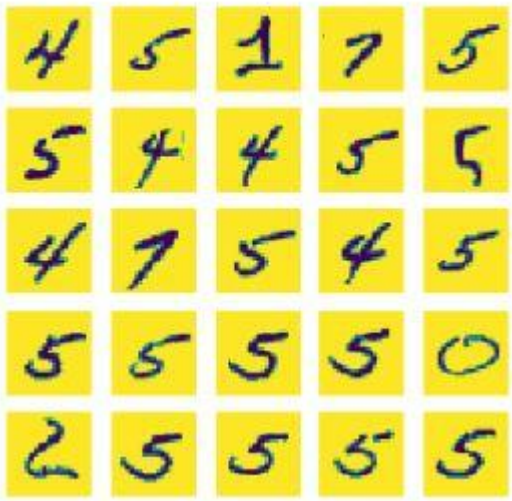
Cluster: 2 Label: 8



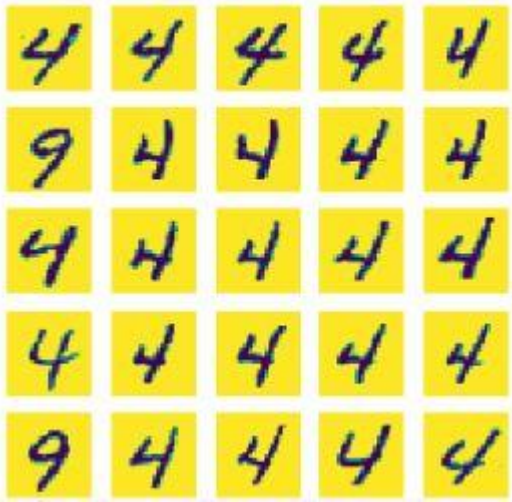
Cluster: 3 Label: 3



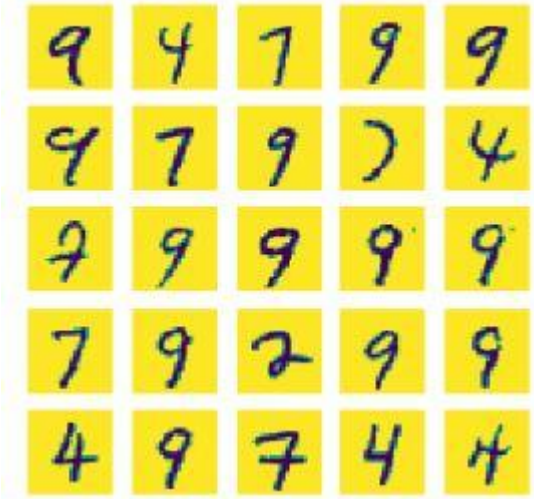
Cluster: 4 Label: 5



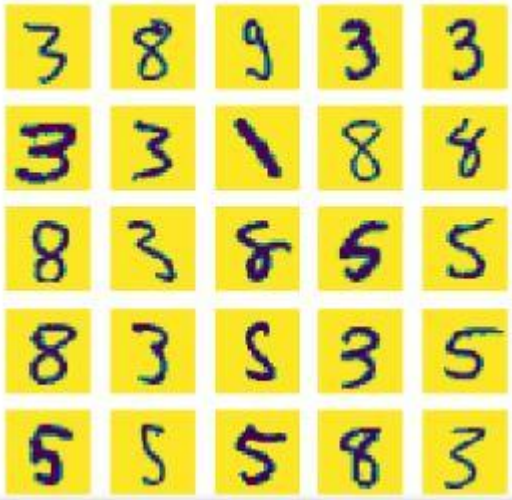
Cluster: 5 Label: 4



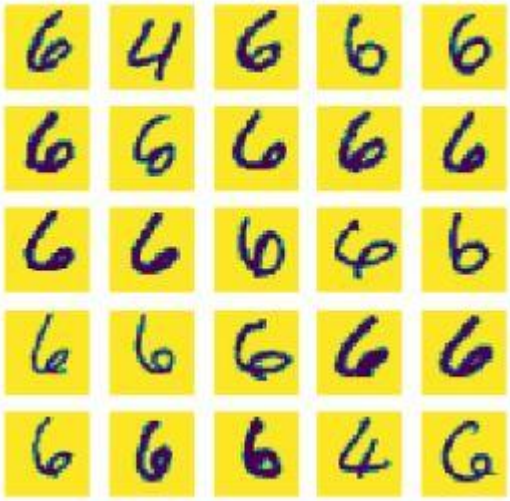
Cluster: 6 Label: 9



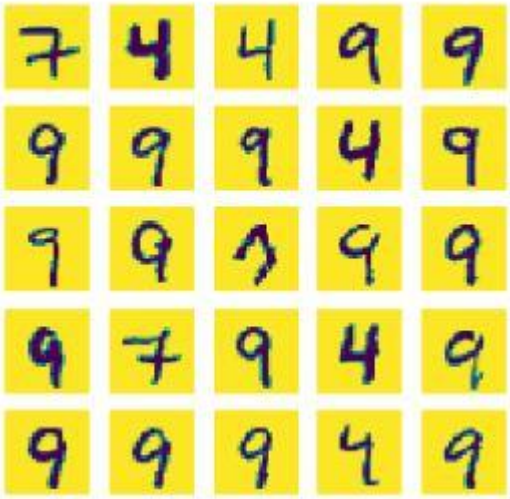
Cluster: 7 Label: 8



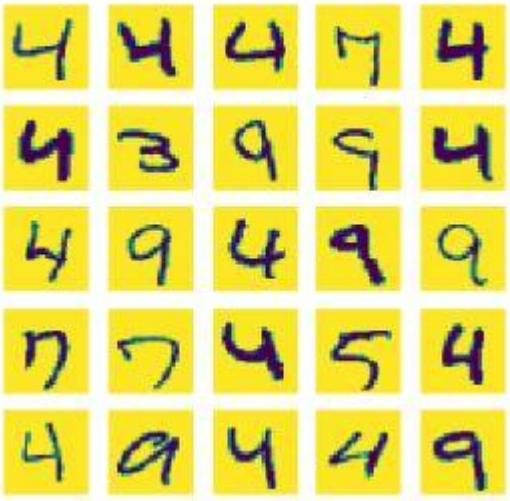
Cluster: 8 Label: 6



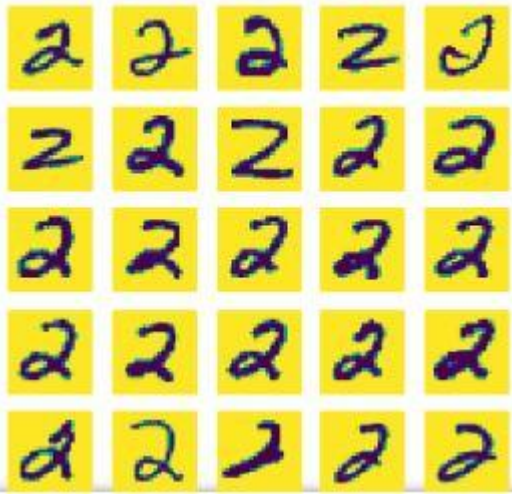
Cluster: 9 Label: 9



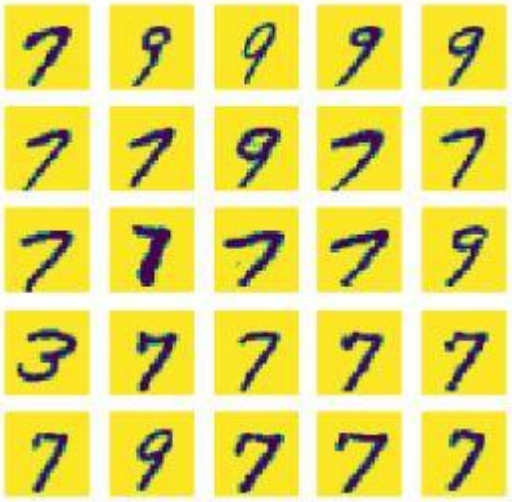
Cluster: 10 Label: 4



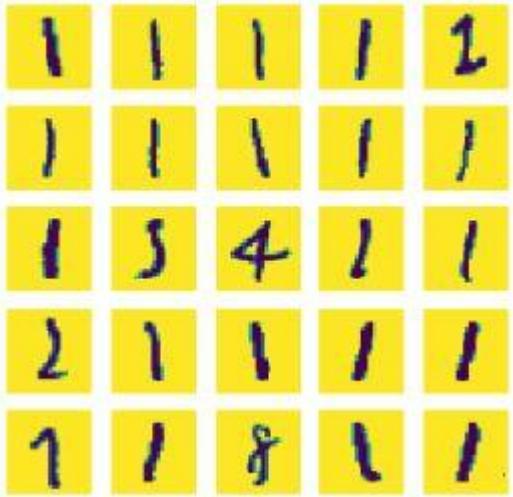
Cluster: 11 Label: 2



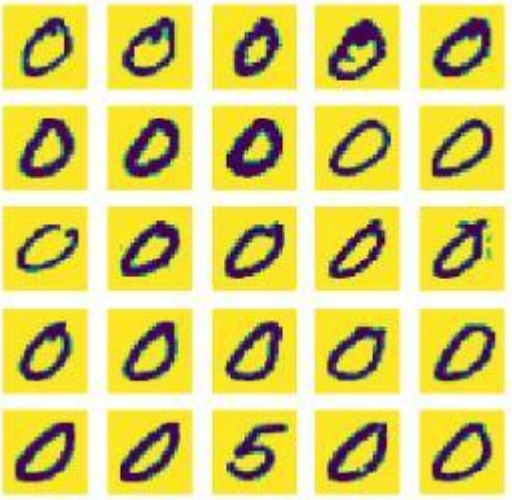
Cluster: 12 Label: 7



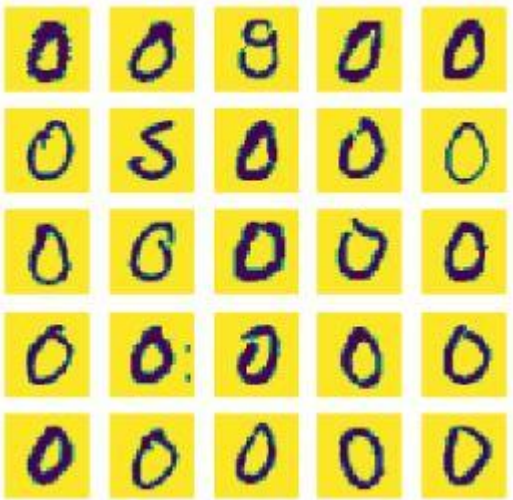
Cluster: 13 Label: 1



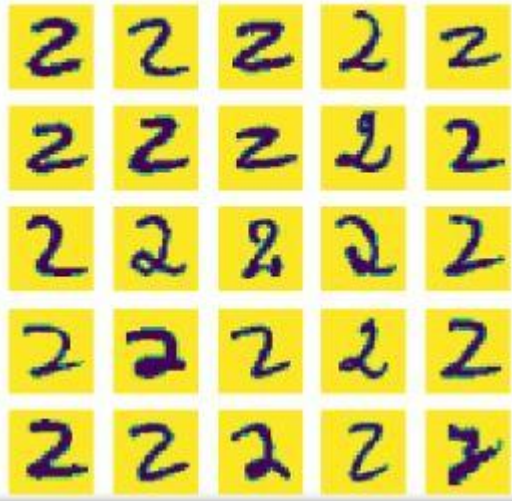
Cluster: 14 Label: 0



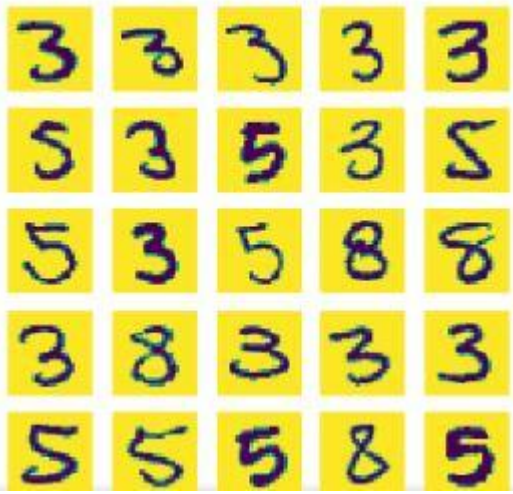
Cluster: 15 Label: 0

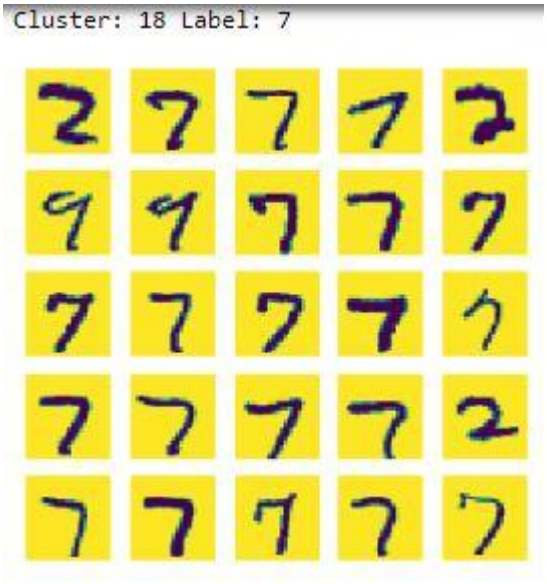


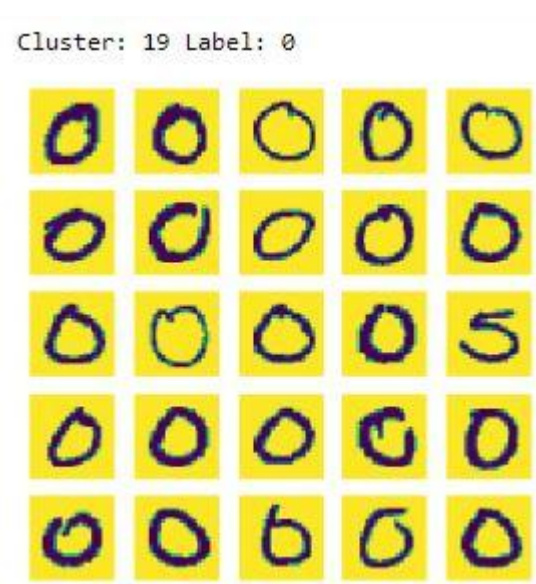
Cluster: 16 Label: 2



Cluster: 17 Label: 3







CONCLUSION :

1)As the K increases the accuracy increases significantly (for k=5, accuracy=51%), (for k=10, accuracy=59%), (for k=20, accuracy=73%)

2) As the number of iterations increase the graph never increases it reaches a stable state where the centroids barely change