ML HW5

October 15, 2021

21500468 GyuSeok,Lee

I. Introduction

In homework 5, I have to implement K-means algorithm by using MNIST '3' and '9' images. I already know that MNIST dataset consists of 50000 x 784 dataset, and I can get 784 x 784 eigen vectors and 784 eigen values. However, in this homework, I should solve the problem by using the dimension of eigenspace of 2,5, and 10. In the experimental part, data were composed of 3 and 9 and pre-processed to explain how K-means were implemented. In the result part, the obtained clustering will be visualized and explained. Using K-means like this, we can divide the given data into k groups and perform unsupervised learning. Let's look at how we solved this problem below.

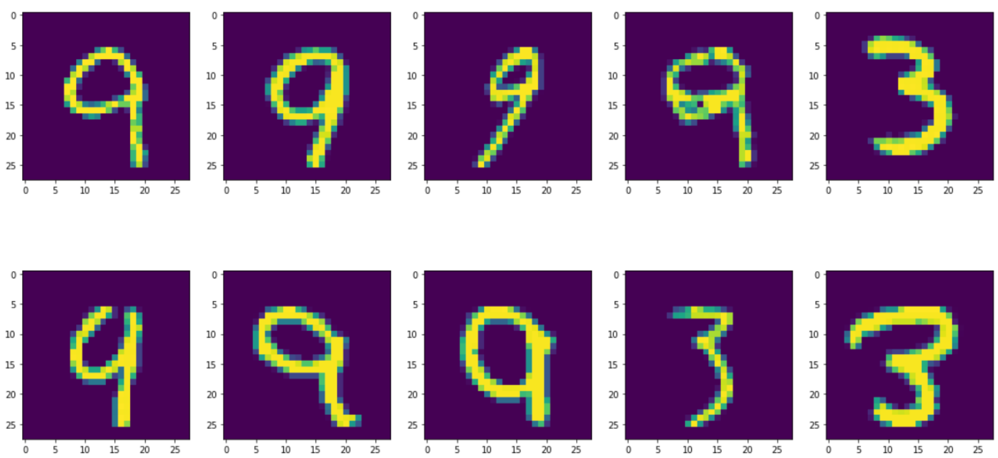
II. Experiment

First, I had to create a dataset consisting of 3 and 9. The size of the dataset thus obtained was 10,089, and 10 random samples were shown as images as follows.

Step1: Mnist 3 & 9 ¶

In [2]:
```
idx = np.where(np.isin(train_y,[3,9]))[0]
Mnist_39 = train_x[idx]

plt.figure(figsize= (20,10))
for i,image in enumerate(random.sample(range(0,Mnist_39.shape[
    plt.subplot(2,5,i+1)
    plt.imshow(Mnist_39[image].reshape(28,28))
```



Second, I should get eigen spaces with dimensions 2,5 and 10. At this time, covariance matrix was used to obtain eigen vectors, and projections with eigen vectors were used to obtain small dimensions of eigen space. The code for data preprocessing is as follows. (Mnist_39 means raw dataset)

Step2

Preprocessing: reduce its dimensionality

- By using EigenSpaces

```
cov = np.cov(Mnist_39.T)
print("Covarince",cov.shape)
```

```
Covarince (784, 784)
```

```
from scipy.linalg import eigh
cov = np.cov(Mnist_39.T)
def Reduce_dim(dim):
    e_values, e_vectors = eigh(cov,eigvals=(783-dim+1,783))
    return np.matmul(e_vectors.T, Mnist_39.T)

dim2 = Reduce_dim(2)
dim5 = Reduce_dim(5)
dim10 = Reduce_dim(10)
```

```
e_values, e_vectors = eigh(cov)
print("Eigen_values",e_values.shape)
print("Eigen_vectors",e_vectors.shape)

Mnist_39 = Mnist_39.T
print("\nMnist_39",Mnist_39.shape)
print("dim2",dim2.shape)
print("dim5",dim5.shape)
print("dim10",dim10.shape)
```

```
Eigen_values (784,)
Eigen_vectors (784, 784)

Mnist_39 (784, 10089)
dim2 (2, 10089)
dim5 (5, 10089)
dim10 (10, 10089)
```

Third, I implemented the K-means algorithm. K initial centers were set up, and clustering was implemented as the center with the shortest distance between data and centers using Euclidean distance. After that, the center was continuously changed using the average inside the cluster, and if there was no change in the labeling of the data, the work was stopped. The code is attached below.

Step3: K-means

```
def K_Means(k, data, iteration = 25):
    data = data.T

    # step1: Select k init value
    N = data.shape[0]
    label = np.zeros(N)
    init = random.sample(range(N),k)
    init_center = data[init]

    # step2: K Means
    while(1):

        # labeling
        for idx, center in enumerate(init_center):
            if idx == 0:
                base = np.sum((data - center)**2, axis = 1)
                base = np.expand_dims(base,0) # dim = (1,10089)
            else:
                temp = np.sum((data - center)**2, axis = 1)
                temp = np.expand_dims(temp,0)
                base = np.concatenate([base,temp], axis = 0) # dim = (k,10

        # Change labels
        before_label = label
        label = np.argmin(base, axis = 0)

        # Terminate condition
        if np.all(before_label == label) and np.unique(label).size == k:
            break

        # New Center
        for cluster in range(k):
            cluster_index = np.where(np.isin(label, cluster))
            if cluster == 0:
                temp = np.mean(data[cluster_index], axis = 0)
                new_center = np.expand_dims(temp,0)
            else:
                temp = np.mean(data[cluster_index], axis = 0)
                temp = np.expand_dims(temp,0)
                new_center = np.concatenate([new_center,temp], axis = 0)
        init_center = new_center
    return init_center, label
```

III. Results

In this part, I would like to show how clustering was performed through K-means through visualization. A plot was made for data with dimension 2, but for the rest of the data, the number of data belonging to each cluster and the Intra-cluster-variance value were expressed using a bar chart. The code for visualization is as follows. Title has k and mean of Intra-cluster variation.

Step4: Visualization

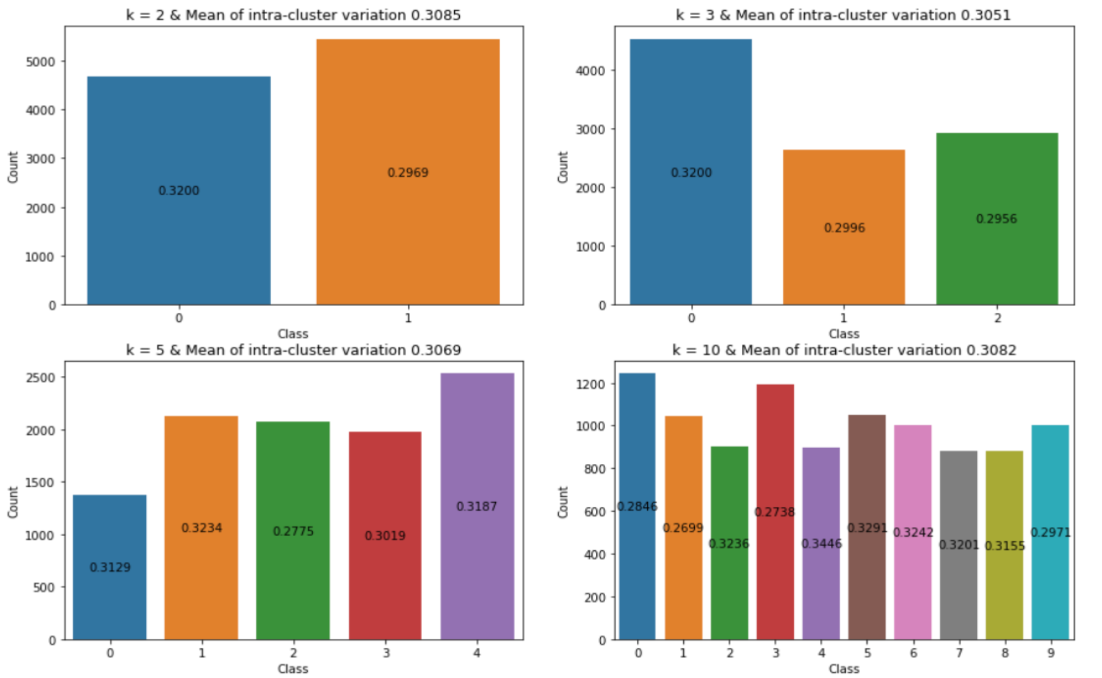
```
K_list = [2,3,5,10]

def Visualization(data):
    plt.figure(figsize= (15,10))
    for idx,k in enumerate(K_list):
        centers,labels = K_Means(k, data)
        plt.subplot(2,2,idx+1)
        X,Y = np.unique(labels, return_counts = True)
        ax = sns.barplot(X,Y)
        plt.xlabel("Class")
        plt.ylabel("Count")

        total_value = 0
        for cluster,p in enumerate(ax.patches):
            left, bottom, width, height = p.get_bbox().bounds
            index = np.where(np.isin(labels,cluster))[0]
            value = np.std(Mnist_39[:,index])
            total_value += value
            ax.annotate("%.4f"%(value), xy=(left+width/2, bottom+height/2)
        plt.title("k = {} & Mean of intra-cluster variation {:.4f}".format
```

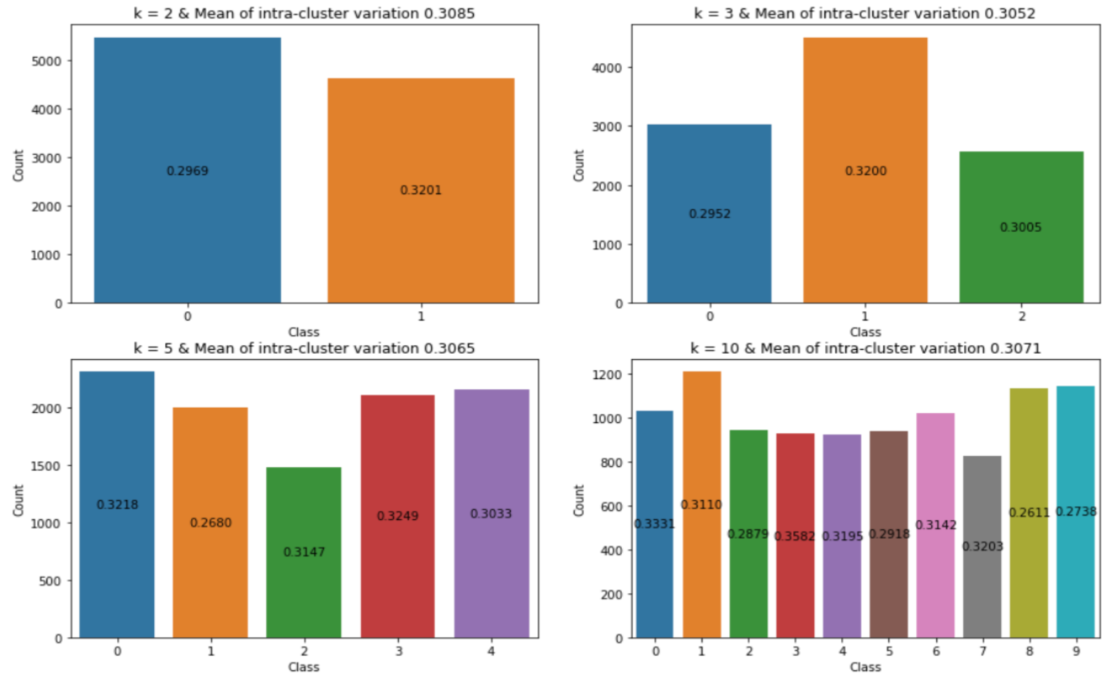
Raw data

```
Visualization(Mnist_39)
```



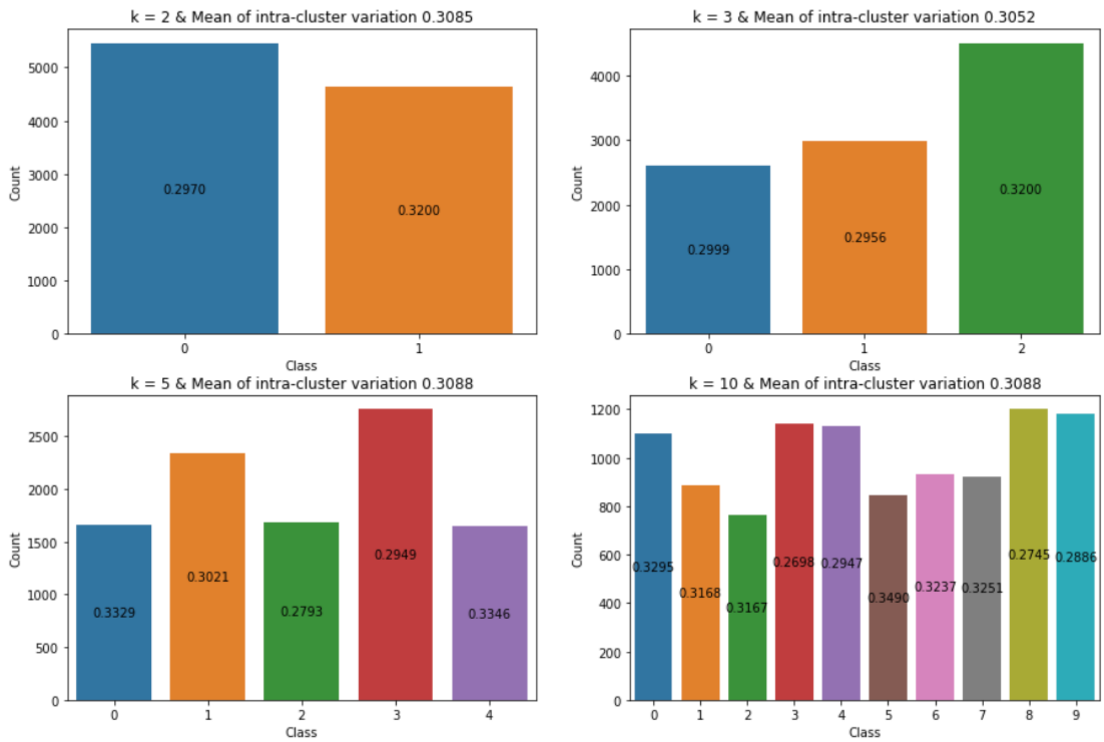
Dim = 5

```
Visualization(dim5)
```



Dim = 10

```
Visualization(dim10)
```



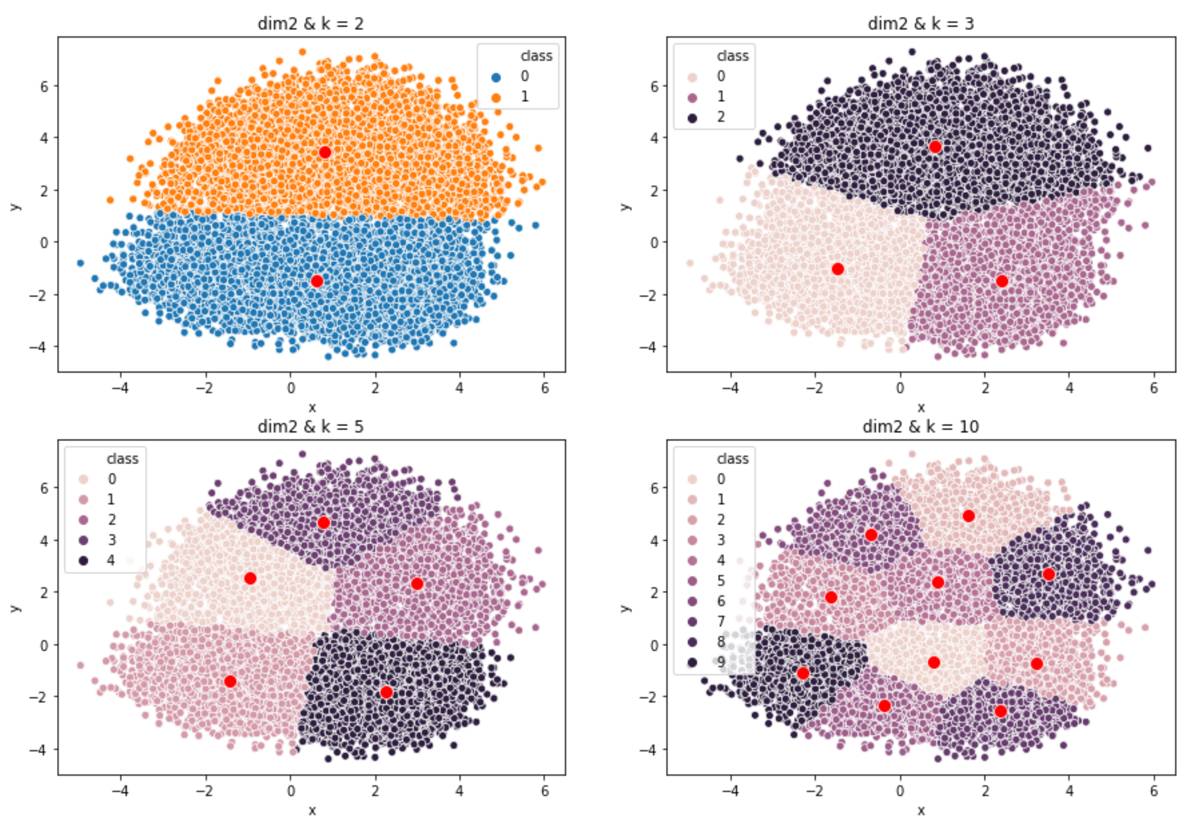
For a dataset with dimension 2, a plot could be drawn using a scatterplot. At this time, the center of the cluster was marked with a red dot.

dim = 2

- Plot the grouped clusters
- Each red dot means the center of cluster

```
plt.figure(figsize= (15,10))
for idx,k in enumerate(K_list):
    centers,labels = K_Means(k, dim2)
    #print(np.unique(labels, return_counts = True))
    plt.subplot(2,2,idx+1)

    # visualization
    df = pd.DataFrame(dim2.T)
    df = pd.concat([df, pd.Series(labels)], axis = 1)
    df.columns = ['x','y','class']
    sns.scatterplot(x = 'x', y= 'y', hue = 'class', data = df, legend= "ful
    sns.scatterplot(centers.T[0], centers.T[1], palette = 'red',color = "r'
    plt.title("dim2 & k = {}".format(k))
```



IV. Conclusions

The K-means algorithm literally refers to a method of obtaining K clusters using the average of each cluster. For datasets without labeling at all, we were able to solve a given problem through unsupervised learning. In KNN, which will be learned in the future, classification of given data can be performed using K neighbors, and the K-means algorithm can be similarly applied. In addition, by learning Euclidean Street, we perform a kind of Metric learning, which is a field of deep learning today, so it can be said that K-means has great significance.