BU.330.775 Machine Learning: Design and Deployment

**Lab 6. Image Clustering using K-Means**

Learning Goal: practice using unsupervised machine learning model to cluster image data

Background: We will use the MNIST dataset for this lab. Please refer to Lab 3 instructions for information about the MNIST dataset.

1. Import the required packages.

from keras.datasets import mnist

from sklearn.cluster import MiniBatchKMeans

from sklearn.metrics import accuracy\_score

import matplotlib.pyplot as plt

import numpy as np

1. First let’s load the MNIST dataset and check the size of the dataset, namely the number of training images, number of testing images, size of each image, and the minimun and maximum values of training data.

(x\_train, y\_train), (x\_test, y\_test) = mnist.load\_data()

print(x\_train.shape)

print(x\_test.shape)

print(x\_train.min())

print(x\_train.max())

1. Then we will plot 9 sample images from the dataset.

plt.gray() # B/W Images

plt.figure(figsize = (10,9)) # Adjusting figure size

# Displaying a grid of 3x3 images

for i in range(9):

plt.subplot(3,3,i+1)

plt.imshow(x\_train[i])

1. We convert the data to float type, and normalize the vectors from 0-255 to range 0-1 for computation efficiency. We will check the minimum and maximum values again after normalization.



# Conversion to float

x\_train = x\_train.astype('float32')

x\_test = x\_test.astype('float32')

# Normalization

x\_train = x\_train/255.0

x\_test = x\_test/255.0

# Checking the minimum and maximum values of x\_train

print(x\_train.min())

print(x\_train.max())

1. The original input data is 3 dimensions: (60000, 28, 28) for training data, and (10000, 28, 28) for testing data. We need to convert it to 2 dimensional format for K-means clustering algorithm. After reshaping the data, the dimensions for training data will be (60000, 784) and for testing data (10000, 784), since 28x28=784.

# Reshaping input data

X\_train = x\_train.reshape(len(x\_train),-1)

X\_test = x\_test.reshape(len(x\_test),-1)

# Checking the shape

print(X\_train.shape)

print(X\_test.shape)

1. Now we are ready to apply the K-means. First, we will define a help function to map cluster labels to the most frequent class labels (from y\_train) in that cluster. Then we will initialize the K-means model with 10 clusters, and we use minibatch version of K-Means.

def retrieve\_info(cluster\_labels,y\_train):

# Initializing

reference\_labels = {}

# For loop to run through each label of cluster label

for i in range(len(np.unique(kmeans.labels\_))):

index = np.where(cluster\_labels == i,1,0)

num = np.bincount(y\_train[index==1]).argmax()

reference\_labels[i] = num

return reference\_labels

total\_clusters = len(np.unique(y\_train))

# Initialize the K-Means model

kmeans = MiniBatchKMeans(n\_clusters = total\_clusters)

# Fitting the model to training set

kmeans.fit(X\_train)

1. After that, we can retrieve the labels and let’s compare the first 20 labels, that is, comparing our K-means prediction with the true label.

reference\_labels = retrieve\_info(kmeans.labels\_,y\_train)

number\_labels = np.random.rand(len(kmeans.labels\_))

for i in range(len(kmeans.labels\_)):

number\_labels[i] = reference\_labels[kmeans.labels\_[i]]

# Comparing Predicted values and Actual values

print(number\_labels[:20].astype('int'))

print(y\_train[:20])

1. We can calculate the overall accuracy score.

# Calculating accuracy score

print(accuracy\_score(number\_labels,y\_train))

1. Now let’s increase the number of clusters (the k value) to 50, and check whether the accuracy improves.

# Increase to 50 clusters, and fit the model

kmeans = MiniBatchKMeans(n\_clusters = 50)

kmeans.fit(X\_train)

# Calculating the reference\_labels

reference\_labels = retrieve\_info(kmeans.labels\_,y\_train)

# ‘number\_labels’ is a list which denotes the number displayed in image

number\_labels = np.random.rand(len(kmeans.labels\_))

for i in range(len(kmeans.labels\_)):

number\_labels[i] = reference\_labels[kmeans.labels\_[i]]

print('Accuracy score : {}'.format(accuracy\_score(number\_labels,y\_train)))

print('\n')

1. Finally, we can visualize the cluster centers to get a better idea about the algorithm.

# Cluster centroids is stored in ‘centroids’

centroids = kmeans.cluster\_centers\_

centroids.shape

centroids = centroids.reshape(50,28,28)

centroids = centroids \* 255

plt.figure(figsize = (10,10))

bottom = 0.35

for i in range(50):

plt.subplots\_adjust(bottom)

plt.subplot(5,10,i+1)

plt.title('Num:{}'.format(reference\_labels[i]),fontsize = 10)

plt.imshow(centroids[i])

**Homework Question 1 (1pt):** Compare the accuracy of 10 clusters vs that of 50 clusters, which one is better?

**Homework Question 2 (1pt):**  Inspect the centroids in step j, discuss why increasing the number of clusters in this case has a positive/negative impact on the model performance.

**Homework Question 3 (1pt):** Comment on the performance of K-means in MNIST image clustering. What insight(s) can we draw?

**Competition Question 1 (2pt):** Describe your steps including data preprocessing and modeling approaches.

**Competition Question 2 (2pt):** Evaluate your model performance compared to the baseline model.

**Submission:** Complete and submit on Canvas by the beginning of Class 7. Use homework6\_yourname.ipynb, and Competition\_yourname.ipynb, respectively, as the file names.

Reference:

<https://medium.com/@joel_34096/k-means-clustering-for-image-classification-a648f28bdc47>