```python
##################################
# TITLE: ex3 machine learning BIU
# WRITER: Raphael Haehnel
# DATE: 8/12/2022
###################################

import os
import matplotlib.pyplot as plt
import numpy as np
import functools
import scipy.io
import shutil
from tqdm import tqdm

LIMIT_ITER = 100


def generate_data(n, c, dim):
    means = 12*(np.random.rand(c, dim) - 0.5)
    means = means.repeat(n, axis=0)

    data = means + np.random.randn(n*c, dim)
    col = np.zeros((n*c, 1))
    data = np.append(data, col, axis=1)
    return data


def generate_means(k, dim):
    means = 12*(np.random.rand(k, dim) - 0.5)
    label = np.array([np.array(range(k))]).T
    return np.append(means, label, axis=1)


def generate_means_pixels(k, dim):
    means = np.random.rand(k, dim)
    label = np.array([np.array(range(k))]).T
    return np.append(means, label, axis=1)


def show_plot(data, mu, i, save, dim):
    if dim == 2:
        plt.scatter(x=data[:, 0], y=data[:, 1], c=data[:, 2], s=20)
        plt.scatter(x=mu[:, 0], y=mu[:, 1], c=mu[:, 2],
                    marker="*", edgecolors="black", s=100)
        plt.xlim([-10, 10])
        plt.ylim([-10, 10])
    if dim == 3:
        fig = plt.figure()
        ax = fig.add_subplot(projection='3d')

        ax.scatter(data[:, 0], data[:, 1], data[:, 2], c=data[:, 3], s=20)
        ax.scatter(mu[:, 0], mu[:, 1], mu[:, 2], c=mu[:, 3],
                   marker="*", edgecolors="black", s=100)

    if save:
        if not os.path.exists('./output'):
            os.mkdir('./output')
        plt.savefig(f'./output/{i}')
        plt.close()
    else:
        plt.show()


def duplicate_mu(data, mu, dim):
    n_data = data.shape[0]
    n_mu = mu.shape[0]

    # Duplicate the values of mu along the vertical axis
    mu_v = np.apply_along_axis(functools.partial(
        np.repeat, repeats=n_data, axis=0), axis=0, arr=mu[:, :dim])

    # Split the array into multiple sub-arrays vertically
    return np.vsplit(mu_v, n_mu)


def assignment(data, mu, dim):

    mu_duplicated = duplicate_mu(data, mu, dim)
```

```python
    x = data[:, :dim]
    result = np.zeros((1, data.shape[0]))

    for i in tqdm(range(mu.shape[0])):
        diff = x-mu_duplicated[i]
        result = np.vstack([result, np.apply_along_axis(
            lambda a: np.sum(a**2), 1, diff)])
    result = result[1:]
    arg_min = np.argmin(result, 0)

    data[:, dim] = arg_min
    cost = np.sum(np.min(result, 0))

    return data, cost


def helper_sum(data, k, dim):
    result = np.zeros((k, dim))
    result[int(data[dim])] = data[:-1]

    return result


def cout_data_labels(data, k, dim):

    count = np.zeros((1, k))
    for row in data:
        count[0, int(row[dim])] += 1

    return count


def centroid_update(data, k, dim):
    cout_labels = cout_data_labels(data, k, dim)
    new_mu = np.apply_along_axis(
        functools.partial(helper_sum, k=k, dim=dim), 1, data)

    new_mu = np.sum(new_mu, axis=0)
    new_mu = np.divide(new_mu, cout_labels.repeat(dim, 0).T, out=np.zeros_like(
        new_mu), where=cout_labels.repeat(dim, 0).T != 0)
    label = np.array([np.array(range(k))]).T
    new_mu = np.append(new_mu, label, axis=1)

    return new_mu


def run_clustering(data, k, mu, dim, save):

    total_cost = np.array([])

    for i in range(LIMIT_ITER):
        data, cost = assignment(data, mu, dim)
        total_cost = np.append(total_cost, cost)
        new_mu = centroid_update(data, k, dim)
        if np.array_equal(new_mu, mu):
            break
        mu = new_mu
        show_plot(data, mu, i, save=save, dim=dim)

    return data, total_cost


def success_rate(y_train, y_output):
    return np.count_nonzero(y_output == y_train)/len(y_output)*100


def normalize(array):
    return array / array.max()


def retrive_data(max_n):
    mat = scipy.io.loadmat('mnist_all.mat')

    x0 = mat.get('train0')[:max_n, :]
    x1 = mat.get('train1')[:max_n, :]
    x2 = mat.get('train2')[:max_n, :]
    x3 = mat.get('train3')[:max_n, :]
    x4 = mat.get('train4')[:max_n, :]
    x5 = mat.get('train5')[:max_n, :]
```

```python
    x6 = mat.get('train6')[:max_n, :]
    x7 = mat.get('train7')[:max_n, :]
    x8 = mat.get('train8')[:max_n, :]
    x9 = mat.get('train9')[:max_n, :]
    X_train = np.concatenate((x0, x1, x2, x3, x4, x5, x6, x7, x8, x9), axis=0)

    y0 = 0*np.ones(len(x0), dtype=int)
    y1 = 1*np.ones(len(x1), dtype=int)
    y2 = 2*np.ones(len(x2), dtype=int)
    y3 = 3*np.ones(len(x3), dtype=int)
    y4 = 4*np.ones(len(x4), dtype=int)
    y5 = 5*np.ones(len(x5), dtype=int)
    y6 = 6*np.ones(len(x6), dtype=int)
    y7 = 7*np.ones(len(x7), dtype=int)
    y8 = 8*np.ones(len(x8), dtype=int)
    y9 = 9*np.ones(len(x9), dtype=int)
    y_train = np.concatenate((y0, y1, y2, y3, y4, y5, y6, y7, y8, y9), axis=0)

    x0_test = mat.get('test0')[:max_n, :]
    x1_test = mat.get('test1')[:max_n, :]
    x2_test = mat.get('test2')[:max_n, :]
    x3_test = mat.get('test3')[:max_n, :]
    x4_test = mat.get('test4')[:max_n, :]
    x5_test = mat.get('test5')[:max_n, :]
    x6_test = mat.get('test6')[:max_n, :]
    x7_test = mat.get('test7')[:max_n, :]
    x8_test = mat.get('test8')[:max_n, :]
    x9_test = mat.get('test9')[:max_n, :]
    X_test = np.concatenate((x0_test, x1_test, x2_test, x3_test,
                            x4_test, x5_test, x6_test, x7_test, x8_test, x9_test), axis=0)

    y0_test = 0*np.ones(len(x0_test), dtype=int)
    y1_test = 1*np.ones(len(x1_test), dtype=int)
    y2_test = 2*np.ones(len(x2_test), dtype=int)
    y3_test = 3*np.ones(len(x3_test), dtype=int)
    y4_test = 4*np.ones(len(x4_test), dtype=int)
    y5_test = 5*np.ones(len(x5_test), dtype=int)
    y6_test = 6*np.ones(len(x6_test), dtype=int)
    y7_test = 7*np.ones(len(x7_test), dtype=int)
    y8_test = 8*np.ones(len(x8_test), dtype=int)
    y9_test = 9*np.ones(len(x9_test), dtype=int)
    y_test = np.concatenate((y0_test, y1_test, y2_test, y3_test,
                            y4_test, y5_test, y6_test, y7_test, y8_test, y9_test), axis=0)

    return normalize(X_train), normalize(y_train), normalize(X_test), normalize(y_test)


if __name__ == "__main__":

    k = 10
    c = k

    if os.path.exists('./output'):
        shutil.rmtree('./output')

    X_train, y_train, X_test, y_test = retrive_data(max_n=100)

    #######################################
    # To use the MNIST data, use this code:
    n = X_train.shape[0]
    dim = X_train.shape[1]
    data = np.append(X_train, np.array([y_train]).T, axis=1)
    mu = generate_means_pixels(k, dim)
    # To generate your own data, use this code:
    # n = 100
    # dim = 3
    # data = generate_data(n, c, dim)
    # mu = generate_means(k, dim)
    #######################################

    output, total_cost = run_clustering(data, k, mu, dim, save=True)
    result_success = success_rate(y_train, output[:, -1])
    print(f'Success rate = {result_success}%')
    plt.plot(total_cost, marker="o")
    plt.show()
```