Problem1: Spectral Clustering

In this problem, I am going to run the spectral clustering algorithm by using the Carnegie Mellon Motion Capture dataset.

(i) Minor data preparation: I am about to load "aca2.mat" and "aca5.mat", where matrix X contains the data points as columns and the vector s contains the true class of each data point. To help reduce the size of the data, I will pick every other column of X and s. As usual, I will normalize the columns of X so that all columns have unit Euclidean norm. The Python code is shown as follows:

```
def input_data(datafile_w):
    """
    Inputs:
        the location of the data file.
    Outputs:
        feature matrix X.
        label vector y.
    """
    aca = loadmat(datafile_w)
    aca_X = [[row.flat[0] for row in line] for line in aca['X']]
    aca_s = [[row.flat[0] for row in line] for line in aca['s']]
    Data_X = np.array(aca_X)
    Data_y = np.array(aca_s)
    Data_X = Data_X[:, ::2]
    Data_y = Data_y[:, ::2]

    return Data_X.T, Data_y.T


def normalize_data(Data_X):
    """
    Inputs:
        the feature matrix X.
    Outputs:
        normalized feature matrix X.
    """
    Data_L2 = expand_dims(sqrt(sum(square(Data_X), axis=1)), axis=1)
    Data_L2[Data_L2 == 0] = 1
    Data_X = true_divide(Data_X, Data_L2)

    return Data_X
```

(ii) Build the following kernel

$$K_{ij} := k(x_i, x_j) = e^{-\gamma \|x_i - x_j\|_{l_2}^2}.$$

Use this as the weight matrix but only pick the top k entries in each column of the matrix K. The weight matrix W picked in this way is not symmetric so I will symmetrize it by using

$$W = (W + W^T)/2.$$

The Python code is:

```
self.K = np.zeros((Data_x.shape[0], Data_x.shape[0]), dtype='float')
for m in range(Data_x.shape[0]):
    for n in range(Data_x.shape[0]):
        if m == n:
```

```
            self.K[m, n] = 1
        else:
            self.K[m, n] = exp(- r * sum(square(Data_x[m] - Data_x[n])))
```

(iii) Next, run the spectral clustering algorithm on these two datasets using the weight matrix W as defined above. The algorithm is

Step1: Define D to be the diagonal matrix whose (i, i)-element is the sum of W's i-th row and construct the matrix $L = D^{-1/2}AD^{-1/2}$.

Step2: Find $x_1, x_2, \dots, x_k$, the k largest eigenvectors of L (chosen to be orthogonal to each other in the case of repeated eigenvalues), and form the matrix $X = [x_1, x_2, \dots, x_k] \in R^{n \times k}$ by stacking the eigenvectors in columns.

Step3: Form the matrix Y from X by renormalizing each of X's rows to have unit length (i.e. $Y_{ij} = X_{ij}/(\sum_j X_{ij}^2)^{1/2}$).

Step4: Treating each row of Y as a point in R^k, cluster them into k clusters via K-means or any other algorithm.

Step5: Assign the original point s_i to cluster j if and only if row i of the matrix Y was assigned to cluster j.

The Python code is:

```
class hw4_specClustering(object):
    def __init__(self, Data_x, r):
        self.Data_x = Data_x    # The feature matrix

        self.K = np.zeros((Data_x.shape[0], Data_x.shape[0]), dtype='float')
        for m in range(Data_x.shape[0]):
            for n in range(Data_x.shape[0]):
                if m == n:
                    self.K[m, n] = 1
                else:
                    self.K[m, n] = exp(- r * sum(square(Data_x[m] - Data_x[n])))

    def get_labels(self, k, label_num):
        """
            Get predicted labels after spectral clustering.
        :param k: The value of k to denote how many entries to choose.
        :param label_num: The number of clusters to do K-means.
        :return: The labels for each data point after spectral clustering.
        """
        """
           Build the weight matrix
        """
        W = copy.deepcopy(self.K)
        for row_i in range(self.K.shape[0]):
            sorted_array = np.sort(self.K[row_i])
            kth_value = sorted_array[self.K.shape[0] - k]
            W[row_i][W[row_i] < kth_value] = 0
        # index = np.argsort(self.K, axis=0)
        # index[index < self.Data_x.shape[0] - k] = 0
        # index[index >= self.Data_x.shape[0] - k] = 1
        # W = multiply(self.K, index)
        W = (W + W.T) / 2
```

```
        # print("W.shape", W.shape)

        """
           Get the diagonal matrix
        """
        D = np.zeros((self.Data_x.shape[0], self.Data_x.shape[0]), dtype='float')
        for i in range(self.Data_x.shape[0]):
            D[i, i] = sum(W[i])

        """
           Construct the L matrix
        """
        # identity_matrix = np.identity(self.Data_x.shape[0])
        # L = identity_matrix - dot(dot(inv(sqrt(D)), W), inv(sqrt(D)))
        L = dot(dot(inv(sqrt(D)), W), inv(sqrt(D)))
        # print("L.shape", L.shape)

        """
           Find the k largest eigenvectors of L
        """
        _, V = np.linalg.eigh(L)
        X = V[:, L.shape[0] - label_num:L.shape[0]]
        # X = V[:, 0: label_num]
        # print("The shape of the k largest eigenvectors X is", X.shape)
        # print("The value of k is", k)

        """
           Form the matrix Y by renormalizing each of X's rows to have unit
length
        """
        Y = normalize_data(X)
        # print("The shape of the normalized matrix X is", Y.shape)

        """
           Do K-means clustering to the matix Y
        """
        from sklearn.cluster import KMeans
        my_kmeans = KMeans(n_clusters=int(label_num), init='k-means++',
max_iter=500, tol=0.00001)
        my_kmeans.fit(Y)

        return my_kmeans.labels_
import numpy as np
import scipy.io as sio

# Step1: Read the data from "aca2.mat" and "aca5.mat"
from hw4_utils import input_data
Data_X1, Data_y1 = input_data(datafile_w='aca2.mat')
Data_X2, Data_y2 = input_data(datafile_w='aca5.mat')
print("Data_X1.shape", Data_X1.shape)
print("Data_y1.shape", Data_y1.shape)
print("Data_X2.shape", Data_X2.shape)
```

```
print("Data_y2.shape", Data_y2.shape)
save_s1, save_s2 = 'aca2_labels.mat', 'aca5_labels.mat'
save_array_s1, save_array_s2 = Data_y1, Data_y2
sio.savemat(save_s1, {'array': save_array_s1})
sio.savemat(save_s2, {'array': save_array_s2})
# Step2: Normalize our data
from hw4_utils import normalize_data
Data_x1 = normalize_data(Data_X1)
Data_x2 = normalize_data(Data_X2)
print("Data_x1.shape", Data_x1.shape)
print("Data_x2.shape", Data_x2.shape)

# Step3: Implement the spectral clustering
from hw4_utils import hw4_specClustering


total_num = 109 * 49
# For the dataset "aca2.mat"
total_results1 = np.zeros((total_num, Data_x1.shape[0]), dtype='int')
label_num1 = max(Data_y1)
iter_i1 = 0
for r in range(1, 9 + 1):
    hw4_solver1 = hw4_specClustering(Data_x=Data_x1, r=r/10)
    for k in range(2, 50 + 1):
        pre_labels1 = hw4_solver1.get_labels(k=k, label_num=label_num1)
        total_results1[iter_i1] = pre_labels1 + 1
        print("For the dataset 'aca2.mat', the iteration is %04d/%04d," %
(iter_i1+1, total_num))
        iter_i1 += 1
for r in range(1, 100 + 1):
    hw4_solver1 = hw4_specClustering(Data_x=Data_x1, r=r)
    for k in range(2, 50 + 1):
        pre_labels1 = hw4_solver1.get_labels(k=k, label_num=label_num1)
        total_results1[iter_i1] = pre_labels1 + 1
        print("For the dataset 'aca2.mat', the iteration is %04d/%04d," %
(iter_i1+1, total_num))
        iter_i1 += 1
save_fn1 = 'total_results1.mat'
save_array1 = total_results1
sio.savemat(save_fn1, {'array': save_array1})
print("The operation of dataset 'aca2.mat' is finished!")


# For the dataset "aca5.mat"
total_results2 = np.zeros((total_num, Data_x2.shape[0]), dtype='int')
label_num2 = max(Data_y2)
iter_i2 = 0
for r in range(1, 9 + 1):
    hw4_solver2 = hw4_specClustering(Data_x=Data_x2, r=r/10)
    for k in range(2, 50 + 1):
        pre_labels2 = hw4_solver2.get_labels(k=k, label_num=label_num2)
        total_results2[iter_i2] = pre_labels2 + 1
```

```
        print("For the dataset 'aca5.mat', the iteration is %04d/%04d," %
(iter_i2+1, total_num))
        iter_i2 += 1
for r in range(1, 100 + 1):
    hw4_solver2 = hw4_specClustering(Data_x=Data_x2, r=r)
    for k in range(2, 50 + 1):
        pre_labels2 = hw4_solver2.get_labels(k=k, label_num=label_num2)
        total_results2[iter_i2] = pre_labels2 + 1
        print("For the dataset 'aca5.mat', the iteration is %04d/%04d," %
(iter_i2+1, total_num))
        iter_i2 += 1
save_fn2 = 'total_results2.mat'
save_array2 = total_results2
sio.savemat(save_fn2, {'array': save_array2})
print("The operation of dataset 'aca5.mat' is finished!")
```

Then, use the following values for $\gamma = 0.1, 0.2, \ldots, 0.9, 1, 2, \ldots, 100$ and $k = 2, 3, 4, \ldots, 50$. Use MATLAB file "Misclassication.m" to record the minimum misclassification error for all of these parameters. For the dataset "aca2.mat", the minimum misclassification error is 0.0771, where all values of misclassification rate are shown in the Figure 1 and minimum values of misclassification among r for each k are shown in the Figure 2. For the dataset "aca5.mat", the minimum misclassification error is 0.0744, where all values of misclassification rate are shown in the Figure 3 and minimum values of misclassification among r for each k are shown in the Figure 4.

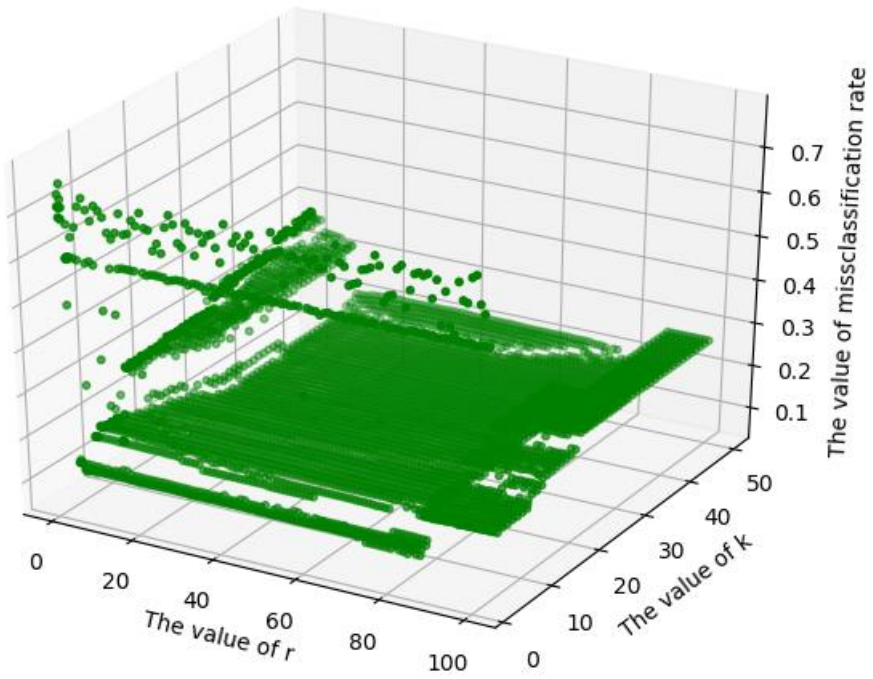

Figure 1: The values of misclassification
of the dataset "aca2.mat" among parameters r and k

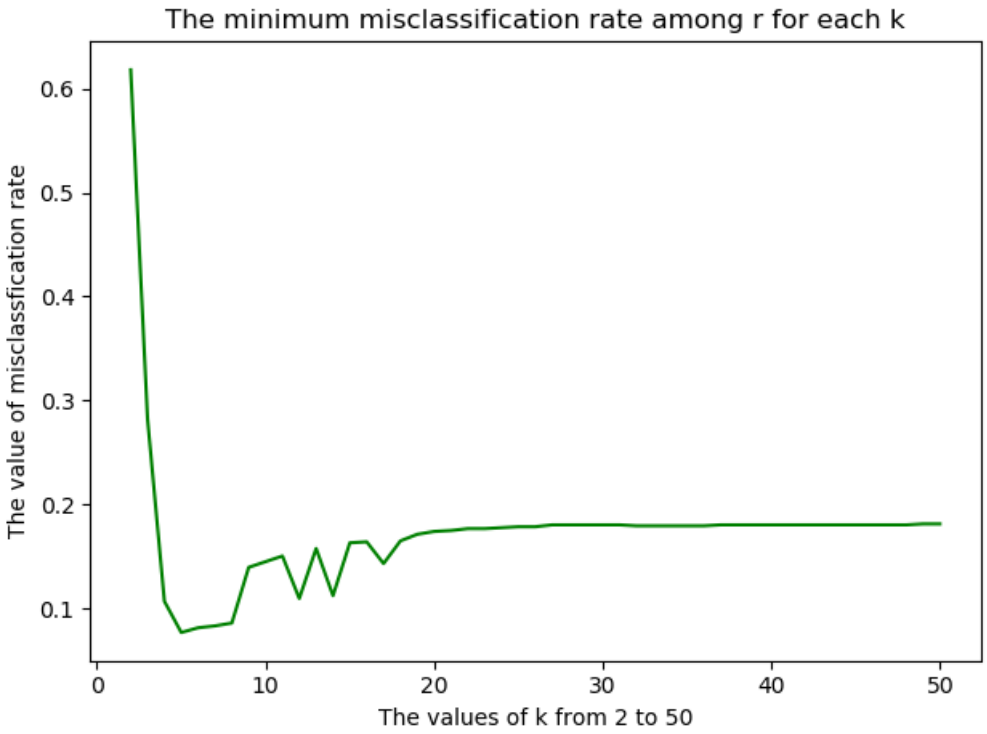

Figure 2: The minimum values of misclassification
of the dataset “aca2.mat” among r for each k

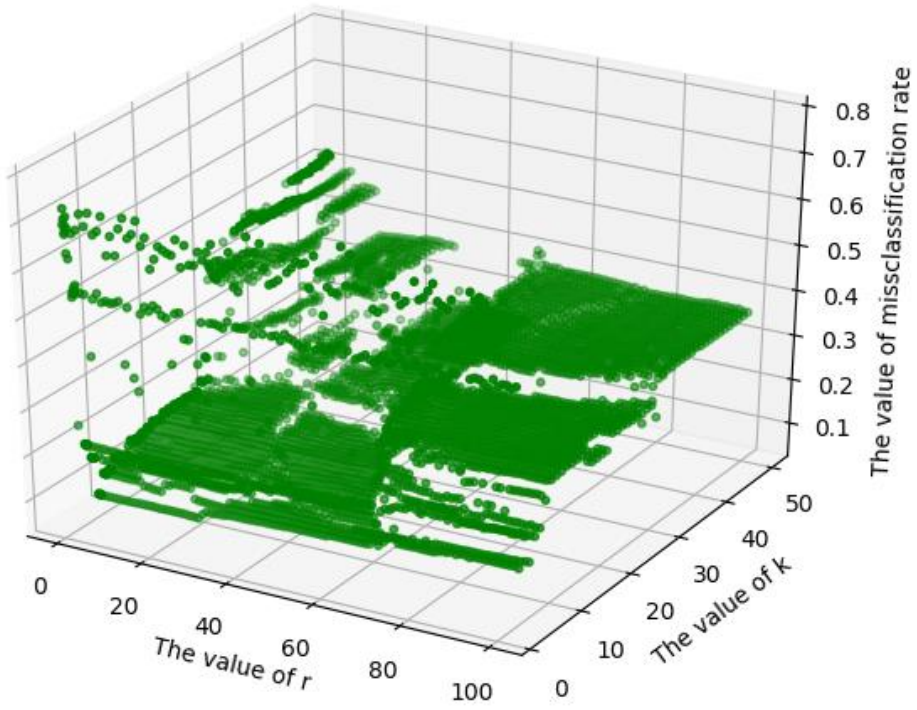

Figure 3: The values of misclassification
of the dataset “aca5.mat” among parameters r and k

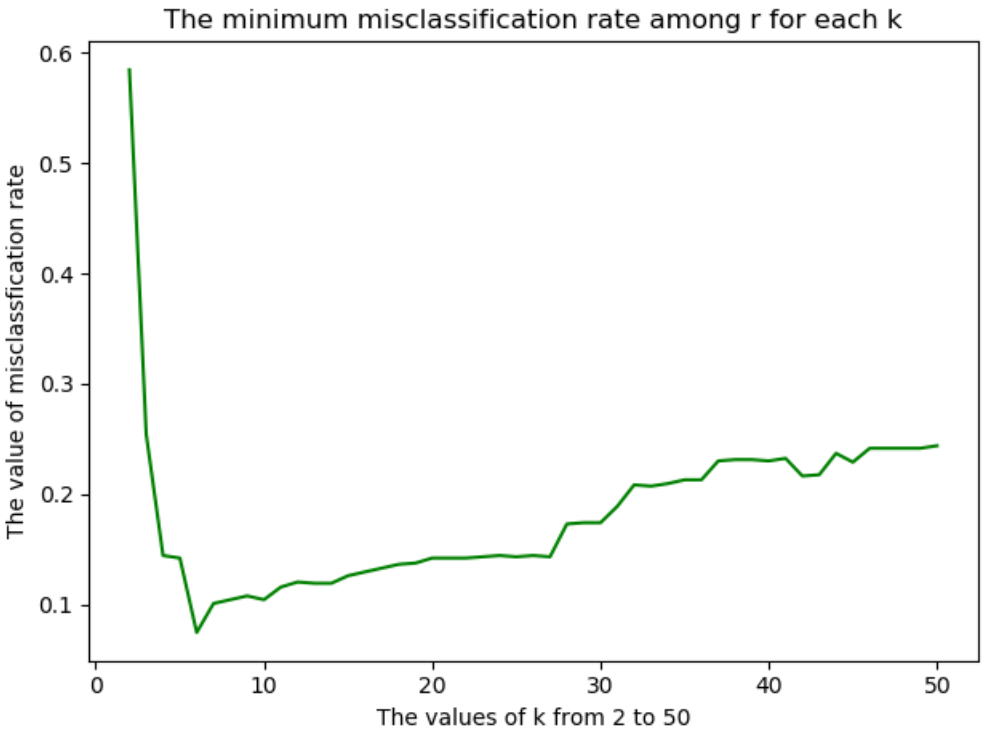

Figure 4: The minimum values of misclassification
of the dataset “aca5.mat” among r for each k