1. Spectral Clustering

Platform: Jupyter Notebook & Python3

I. Minor data preparation

Pick every other column of X and s to reduce size.
Normalize the columns of X.

In [1]:
```
# Homework_4 / Nov. 2020  / Kangyan Xu
# np.set_printoptions(threshold = 1200)

import numpy as np
from sklearn.cluster import KMeans
from munkres import Munkres
import time
```

In [2]:
```
from scipy.io import loadmat

start_time = time.time()
data = loadmat('aca2.mat')
cluster_num = 8

# shape[n_features ,n_samples]
X = data['X'] # (42, 2204) data points
s = data['s'] # (1, 2204) true class

# pick every other column
X = X[:, ::2] # (42, 1102)
s = s[:, ::2] # (1, 1102)
```

In [3]:
```
from sklearn.preprocessing import normalize
X = normalize(X, axis = 0, norm = 'l2')

# Transpose
X = X.T # (1102,42)
s = s.flatten() # (1102,)

# print(np.sum(X[0]**2))
```

II. Build the kernel (Spectral Clustering)

Build the kernel K.
Construct the Laplace matrix L.
Find *k* largest eigen vectors of L and build matrix X.
Form matrix Y by renormalizing X's rows.
Treating each row of Y as points, cluster into *n* clusters (*n* in 'aca2' is 8, in 'aca5' is 7).

```
In [4]: # Build the kernel K part I

        data_num = X.shape[0] # 1102
        l2_dist = np.zeros((data_num, data_num))

        for i in range(data_num):
            for j in range(data_num):
                l2_dist[i,j] = np.square(np.linalg.norm(X[i]-X[j]))
```

```
In [5]: def Spectral_Cluster(X, r, k, cluster_num):
            data_num = X.shape[0] # 1102
            K = np.zeros((data_num, data_num))

            # Build the kernel K part II
            for i in range(data_num):
                for j in range(data_num):
                    K[i,j] = np.exp(-r * X[i,j])

            # Pick the top k entries in each column
            sort_index = np.argsort(-K, axis=0) # index array sorted from large to small

            W_temp = np.zeros((data_num, data_num))
            for i in range(data_num):
                W_temp[0:k,i] = K[sort_index[0:k,i],i] # Pick the top k largest entries in K

            W = (W_temp + W_temp.T)/2.0

            # Calcualte diagonal matrix D whose (i,i) element is the sum of W's i'th row
            # Calculate L = D^(-1/2) A D^(-1/2)

            D_entry = np.sum(W, axis=1)
            D_pow = np.diag(1.0/np.power(D_entry, 0.5))
            L = np.dot(np.dot(D_pow, W), D_pow)
            L = (L+L.T)/2.0

            # Find cluster_num largest eigenvectors of L

            evalues, evectors = np.linalg.eigh(L) # evalues sorted from small to large

            # evalues_topk = np.real(evalues[:-cluster_num-1:-1])
            evectors_topk = np.real(evectors[:, :-cluster_num-1:-1]) # Pick last k largest evectors, (1102,8)

            Y = normalize(evectors_topk, axis = 1, norm = 'l2')

            kmean = KMeans(n_clusters = cluster_num)
            kmean.fit(Y)
            label = kmean.labels_

            return label
```

III. Calculate Misclassification Error

Use the following values $r = 0.1, 0.2, \ldots, 0.9, 1, 2, \ldots, 100$ and $k = 2, 3, 4, \ldots, 50$. For each k, find the minimum misclassification error calculated among all different parameter r's.

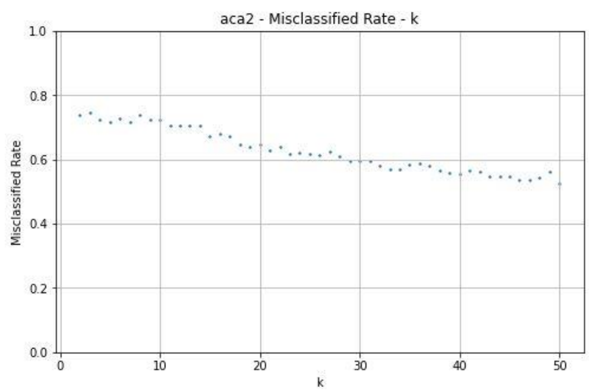
```
In [10]: error

Out[10]: array([[ 0.73956443,  0.74500907,  0.72504537,  0.71597096,  0.72595281,
                  0.71778584,  0.73956443,  0.72323049,  0.72323049,  0.70689655,
                  0.70508167,  0.70689655,  0.70689655,  0.67422868,  0.68058076,
                  0.67422868,  0.64519056,  0.64065336,  0.64519056,  0.62794918,
                  0.63793103,  0.61705989,  0.6215971 ,  0.61615245,  0.61433757,
                  0.62431942,  0.60980036,  0.59618875,  0.59437387,  0.59528131,
                  0.58166969,  0.56896552,  0.5707804 ,  0.58348457,  0.5862069 ,
                  0.57894737,  0.56442831,  0.55716878,  0.55626134,  0.56715064,
                  0.56261343,  0.54809437,  0.54809437,  0.54627949,  0.53811252,
                  0.53629764,  0.54264973,  0.56170599,  0.52631579],
                [83.        , 54.        , 25.        ,  4.        ,  2.        ,
                  1.        ,  0.2       ,  0.8       ,  4.        ,  1.        ,
                  0.4       ,  2.        ,  0.6       ,  0.1       ,  0.9       ,
                  0.4       ,  0.7       ,  2.        ,  0.3       ,  3.        ,
                  3.        ,  7.        ,  5.        ,  9.        , 12.        ,
                  2.        ,  0.2       , 24.        , 21.        , 19.        ,
                  2.        ,  0.9       ,  0.1       ,  4.        ,  5.        ,
                  0.3       ,  9.        ,  8.        ,  9.        , 10.        ,
                 20.        , 20.        , 16.        , 19.        , 13.        ,
                 12.        , 23.        , 19.        , 13.        ]])
```

```
In [11]: time = time.time()-start_time
         print("The run time is %.2f" %time)

         The run time is 11844.72
```

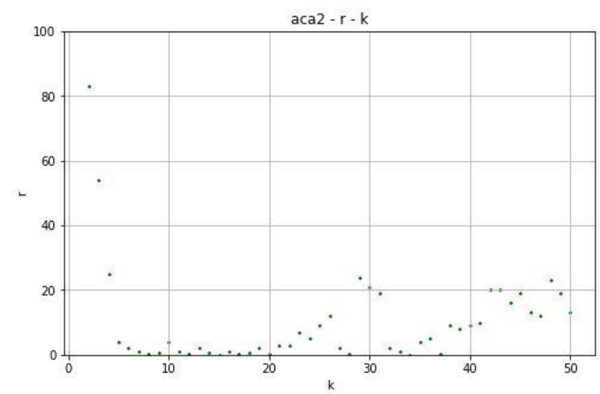
*results of data 'aca2', with **minimum error rate** and corresponding parameter **r***

```
In [10]: error

Out[10]: array([[ 0.74370709,  0.75286041,  0.72768879,  0.72540046,  0.6819222 ,
                  0.66132723,  0.65446224,  0.64530892,  0.64302059,  0.64302059,
                  0.64874142,  0.63501144,  0.62242563,  0.6201373 ,  0.61441648,
                  0.60640732,  0.59954233,  0.59038902,  0.5812357 ,  0.58466819,
                  0.58466819,  0.5812357 ,  0.56750572,  0.55949657,  0.55606407,
                  0.55377574,  0.55034325,  0.54462243,  0.54576659,  0.54462243,
                  0.54576659,  0.54576659,  0.54347826,  0.54347826,  0.54462243,
                  0.5423341 ,  0.54347826,  0.52974828,  0.51830664,  0.51716247,
                  0.50915332,  0.51372998,  0.50228833,  0.5       ,  0.50114416,
                  0.5       ,  0.49771167,  0.49542334,  0.49313501],
                [ 8.        , 70.        , 16.        , 16.        , 11.        ,
                 19.        , 17.        , 14.        ,  8.        ,  9.        ,
                  0.4       ,  5.        , 11.        , 15.        , 13.        ,
                 13.        , 18.        , 12.        , 17.        , 17.        ,
                 16.        , 12.        , 20.        , 24.        , 20.        ,
                 25.        , 17.        , 19.        , 16.        , 16.        ,
                 18.        , 21.        , 32.        , 33.        , 31.        ,
                 30.        ,  8.        ,  9.        , 10.        , 10.        ,
                 11.        , 13.        , 10.        , 13.        , 12.        ,
                 15.        , 14.        , 11.        , 13.        ]])

In [11]: time = time.time()-start_time
         print("The run time is %.2f" %time)

         The run time is 7414.43
```

*results of data 'aca5', with **minimum error rate** and corresponding parameter **r***

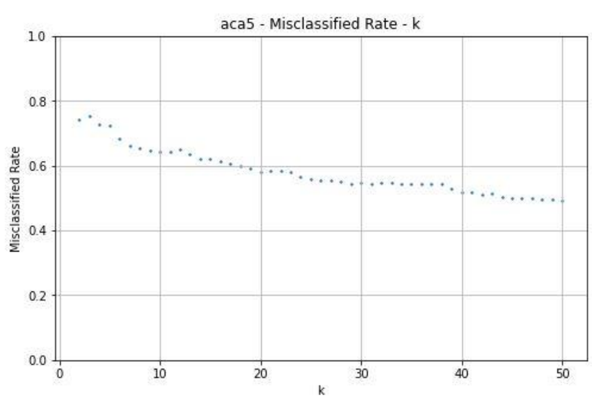


minimum misclassification error

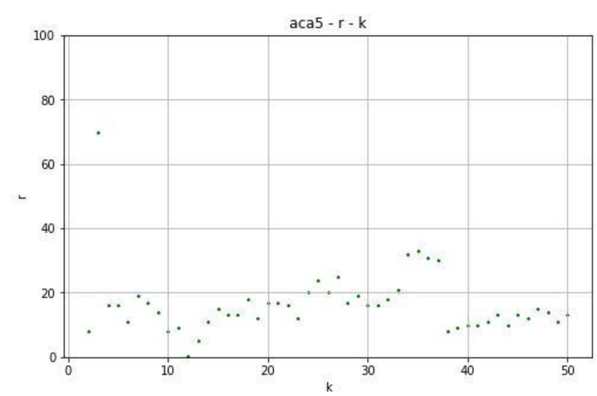


corresponding parameter r's

The minimum misclassification error on 'aca2' is **0.526**, corresponding *r* is **13**, *k* is 50.



minimum misclassification error



corresponding parameter r's

The minimum misclassification error on 'aca5' is **0.493**, corresponding *r* is also **13**, *k* is 50.

Other Code parts:

```
In [6]: # Project: MvDSCN   Author: huybery   File: metric.py   License: MIT License

def best_map(L1, L2):
    #L1 should be the groundtruth labels and L2 should be the clustering labels we got
    Label1 = np.unique(L1)
    nClass1 = len(Label1)
    Label2 = np.unique(L2)
    nClass2 = len(Label2)
    nClass = np.maximum(nClass1,nClass2)
    G = np.zeros((nClass,nClass))
    for i in range(nClass1):
        ind_cla1 = L1 == Label1[i]
        ind_cla1 = ind_cla1.astype(float)
        for j in range(nClass2):
            ind_cla2 = L2 == Label2[j]
            ind_cla2 = ind_cla2.astype(float)
            G[i,j] = np.sum(ind_cla2 * ind_cla1)
    m = Munkres()
    index = m.compute(-G.T)
    index = np.array(index)
    c = index[:,1]
    newL2 = np.zeros(L2.shape)
    for i in range(nClass2):
        newL2[L2 == Label2[i]] = Label1[c[i]]
    return newL2
```

```
In [7]: # Calculate Misclassification error

def Misclassification_error(true_label, cluster_label):
    cluster_label_new = best_map(true_label, cluster_label)
    error_points_num = np.sum(true_label[:] != cluster_label_new[:])
    misclassified_rate = error_points_num / cluster_label_new.shape[0]
    return misclassified_rate
```

This Misclassification Function is modified from net resource

```
In [8]: # 1st row is min error, 2nd row is corresponding r

error = np.zeros((2,49))
```

```
In [9]: for k in range (2,51):
    min_error = 1
    min_r = 0

    for r in range (1,10):
        label = Spectral_Cluster(l2_dist, 0.1*r, k, cluster_num)
        temp = Misclassification_error(s, label)
        if min_error > temp:
            min_error = temp
            min_r = 0.1*r

    for r in range (1,101):
        label = Spectral_Cluster(l2_dist, r, k, cluster_num)
        temp = Misclassification_error(s, label)
        if min_error > temp:
            min_error = temp
            min_r = r

    print(min_error, min_r)
    error[0,k-2] = min_error
    error[1,k-2] = min_r
```

Recording minimum misclassification error

end.