Sheet#4 Clustering Evaluation

1. Perform clustering on the following data

In [2]:

```
#imports cell
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from copy import deepcopy
from sklearn.metrics.pairwise import rbf_kernel as rbf
from sklearn.neighbors import NearestNeighbors as nn
import pandas as pd
```

In [3]:

```
#load data on 2D graph onto 2D array for usage.
clustering_dataSet = np.array([[2,4],[3,3],[3,4],[5,4],[5,6],[5,8],[6,4],[6,5],[6,7],[7,3],[7,4],[8,2],[9,4]
,[10,6],[10,7],[10,8],[11,5],[11,8],[12,7],[13,6],[13,7],[14,6],[15,4],[15,5]])
```

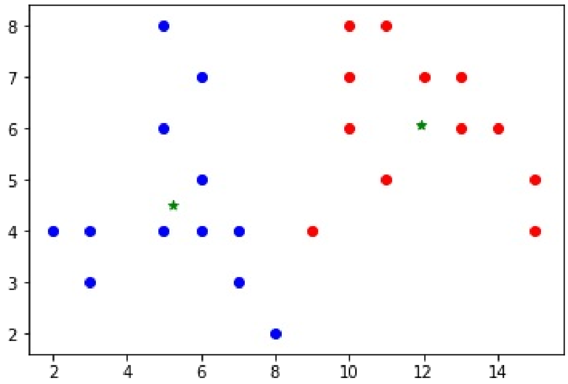
a. Using Kmeans: set K=2,3,4,5,6. Report different clustering results.

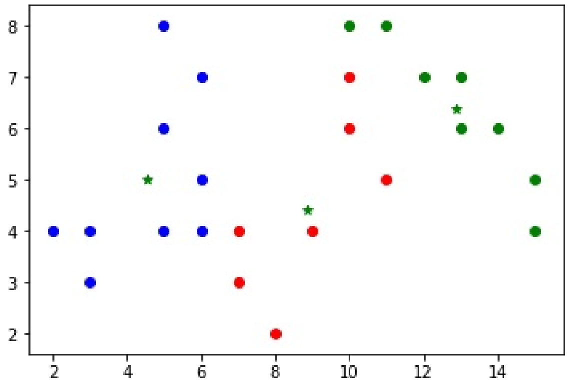
In [13]:

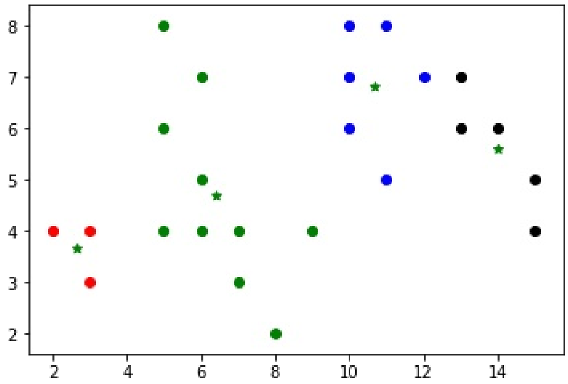
```
#my k-means implementation
def kmeans(data, num_clusters = 2, tolerance=0.0001, max_iter = 300, init_seed = None):
    iter_num = 0
    # Number of training data
    n = data.shape[0]
    # Number of features in the data
    c = data.shape[1]
    # Generate random centers, here i use standard devation
    #and mean to ensure it represents the whole data
    if(init_seed is None):
        mean = np.mean(data, axis = 0)
        std = np.std(data, axis = 0)
        centroids = np.random.randn(num_clusters,c)*std + mean
    else:
        centroids = init_seed

    # to store old centers
    old_centroids = np.zeros(centroids.shape)
    # Store new centers
    new_centroids = deepcopy(centroids)
    #generate error vector
    error = np.linalg.norm(new_centroids - old_centroids)
    #create clusters array
    clusters = np.zeros(n)
    #create distaces array
    distances = np.zeros((n,num_clusters))
    # When, after an update, the estimate of that center stays the same, exit loop
    while error > tolerance and iter_num < max_iter:
        iter_num +=1
        # Measure the distance to every center
        for i in range(num_clusters):
            distances[:,i] = np.linalg.norm(data - new_centroids[i], axis=1)
        # Assign all training data to closest center
        clusters = np.argmin(distances, axis = 1)
        old_centroids = deepcopy(new_centroids)
        # Calculate mean for every cluster and update the center
        for i in range(num_clusters):
            new_centroids[i] = np.mean(data[clusters == i], axis=0)
        error = np.linalg.norm(new_centroids - old_centroids)

    return new_centroids,clusters
```

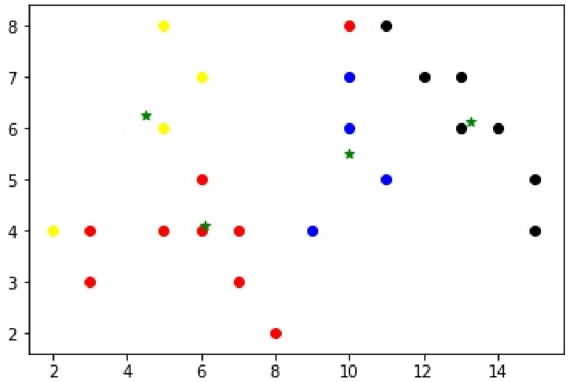
In [75]:

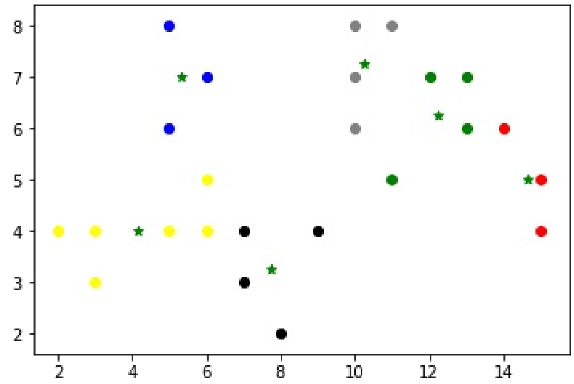
```
colors=['red', 'blue', 'green','black','yellow','gray']
kmeans_clusters = []
for k in range(2,7):
    centroids,clusters = kmeans(clustering_dataSet,num_clusters=k)
    kmeans_clusters.append(clusters)
    # Plot the data and the centers generated as random
    for i in range(clustering_dataSet.shape[0]):
        plt.scatter(clustering_dataSet[i, 0], clustering_dataSet[i,1],color = colors[clusters[i]]);
    plt.scatter(centroids[:,0], centroids[:,1], marker='*', c='g');
    plt.figure();
```











<Figure size 432x288 with 0 Axes>

b. K-ways normalized: cut k=2,3,4,5,6

In [23]:

```
def SpectralClustering(data, num_clusters=2, affinity='rbf', gamma=1.0, num_neighbors=1):
    if(affinity == 'rbf'):
        sim_matrix = rbf(data,data,gamma)
    elif(affinity == 'knn'):
        nearest_neigbhor = nn(n_neighbors=num_neighbors)
        nearest_neigbhor.fit(data)
        sim_matrix = nearest_neigbhor.kneighbors_graph(data, mode='connectivity').toarray()

    deg_matrix = np.diag(np.sum(sim_matrix, axis=1))
    laplace_matrix = deg_matrix - sim_matrix
    asym_laplace_matrix = np.dot(np.linalg.inv(deg_matrix),laplace_matrix)
    eig_values,eig_vectors = np.linalg.eig(asym_laplace_matrix)
    idx = np.real(eig_values).argsort()[:num_clusters]
    eig_vectors = np.real(eig_vectors[:,idx])
    rows_norm = np.linalg.norm(eig_vectors, axis=1)
    normalized_eig_vectors = (eig_vectors.T / rows_norm).T
    centroids,clusters = kmeans(normalized_eig_vectors, num_clusters=num_clusters)
    return normalized_eig_vectors,centroids,clusters
```

i. Use RBF kernel with gamma = {0.01,0.1}. Report the different clustering results.

In [105]:

```
spectral_rbf_clusters = []
for gamma in [0.01,0.1]:
    for k in range(2,7):
        new_data,centroids,clusters = SpectralClustering(clustering_dataSet,num_clusters=k, affinity='rbf',
gamma= gamma)
        spectral_rbf_clusters.append(clusters)
        print("For K = ",k,"and gamma = ",gamma,"\nClusters Vector is \n",clusters)
```

```
For K =  2 and gamma =  0.01
Clusters Vector is
 [0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1]
For K =  3 and gamma =  0.01
Clusters Vector is
 [2 2 2 2 2 0 2 2 0 2 2 1 1 0 0 0 1 0 0 1 0 1 1 1]
For K =  4 and gamma =  0.01
Clusters Vector is
 [1 1 1 1 1 0 3 1 0 3 3 3 3 0 0 0 3 0 0 2 2 2 2 2]
For K =  5 and gamma =  0.01
Clusters Vector is
 [2 2 2 3 1 1 3 1 1 3 3 3 3 0 0 0 3 0 0 0 0 4 4 4]
For K =  6 and gamma =  0.01
Clusters Vector is
 [2 2 2 2 3 3 1 1 3 0 1 0 1 1 5 5 1 5 5 4 5 4 4 4]
For K =  2 and gamma =  0.1
Clusters Vector is
 [1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0]
For K =  3 and gamma =  0.1
Clusters Vector is
 [0 0 0 0 0 0 0 0 0 0 0 0 2 2 2 2 2 2 2 1 1 1 1 1]
For K =  4 and gamma =  0.1
Clusters Vector is
 [3 3 3 0 0 0 0 0 0 0 0 0 0 2 2 2 2 2 2 1 1 1 1 1]
For K =  5 and gamma =  0.1
Clusters Vector is
 [3 3 3 4 4 4 0 4 4 0 0 0 0 1 1 1 1 1 1 2 2 2 2 2]
For K =  6 and gamma =  0.1
Clusters Vector is
 [0 0 0 1 2 2 1 1 2 1 1 1 5 5 5 5 5 5 4 4 4 4 3 3]
```

ii. Use Similarity graph as the {3,5}-NN graph. Where Sim(xi,xj)=1 iff xj is one of the nearest three points to xi (or vise versa). Report different clustering results.

In [142]:

```
spectral_knn_clusters = []
for j in [3,5]:
    for k in range(2,7):
        new_data,centroids,clusters = SpectralClustering(clustering_dataSet,num_clusters=k, affinity='knn',
num_neighbors= j)
        spectral_knn_clusters.append(clusters)
        print("For K = ",k,"and NN = ",j,"\nClusters Vector is \n",clusters)
```

```
For K =  2 and NN =  3
Clusters Vector is
 [1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
For K =  3 and NN =  3
Clusters Vector is
 [0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1]
For K =  4 and NN =  3
Clusters Vector is
 [2 2 2 3 3 3 3 3 3 3 3 3 3 0 0 0 3 0 1 1 1 1 1 1]
For K =  5 and NN =  3
Clusters Vector is
 [1 1 1 3 4 4 3 3 4 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3]
For K =  6 and NN =  3
Clusters Vector is
 [3 3 3 2 1 5 2 2 5 2 2 2 2 2 2 5 2 5 5 5 5 5 5 5]
For K =  2 and NN =  5
Clusters Vector is
 [0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1]
For K =  3 and NN =  5
Clusters Vector is
 [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
For K =  4 and NN =  5
Clusters Vector is
 [0 0 0 0 1 1 1 1 1 2 1 2 2 3 3 3 3 3 3 0 0 0 0 0]
For K =  5 and NN =  5
Clusters Vector is
 [3 3 3 0 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 4 4 4 4 4]
For K =  6 and NN =  5
Clusters Vector is
 [0 0 0 0 1 1 1 1 1 1 1 4 4 2 2 2 2 2 3 5 3 5 5 5]

/home/zawawy/Public/anaconda3/lib/python3.7/site-packages/numpy/core/fromnumeric.py:2920: Runti
meWarning: Mean of empty slice.
  out=out, **kwargs)
/home/zawawy/Public/anaconda3/lib/python3.7/site-packages/numpy/core/_methods.py:78: RuntimeWar
ning: invalid value encountered in true_divide
  ret, rcount, out=ret, casting='unsafe', subok=False)
/home/zawawy/Public/anaconda3/lib/python3.7/site-packages/numpy/core/fromnumeric.py:2920: Runti
meWarning: Mean of empty slice.
  out=out, **kwargs)
/home/zawawy/Public/anaconda3/lib/python3.7/site-packages/numpy/core/_methods.py:78: RuntimeWar
ning: invalid value encountered in true_divide
  ret, rcount, out=ret, casting='unsafe', subok=False)
/home/zawawy/Public/anaconda3/lib/python3.7/site-packages/numpy/core/fromnumeric.py:2920: Runti
meWarning: Mean of empty slice.
  out=out, **kwargs)
/home/zawawy/Public/anaconda3/lib/python3.7/site-packages/numpy/core/_methods.py:78: RuntimeWar
ning: invalid value encountered in true_divide
  ret, rcount, out=ret, casting='unsafe', subok=False)
/home/zawawy/Public/anaconda3/lib/python3.7/site-packages/numpy/core/fromnumeric.py:2920: Runti
meWarning: Mean of empty slice.
  out=out, **kwargs)
/home/zawawy/Public/anaconda3/lib/python3.7/site-packages/numpy/core/_methods.py:78: RuntimeWar
ning: invalid value encountered in true_divide
  ret, rcount, out=ret, casting='unsafe', subok=False)
```

c. Assume the ground truth clustering results is T1={p,q,v}, T2={a,d,h,k,r,s,t,l,w,x} and T3={b,c,e,i,m,f,g,j,n,a,u}.

In [143]:

```
ground_truth = np.array([0,0,0,1,1,1,1,1,1,1,1,1,1,2,2,2,2,2,2,2,2,2,2,2])
```

i. Compute the external measures we studied.

1. Purity

$$purity = \frac{1}{n}\sum_{i=1}^{r}\max_{j=1}^{k} n_{ij}$$

2. Max matching

$$match = argmax_M\{\frac{w(M)}{n}\}$$

3. F-measure

$$F_i = \frac{2n_{ij_i}}{n_i + m_{j_i}}$$

$$F = \frac{1}{r}\sum_{i=1}^{r} F_i$$

4. Conditional Entropy

$$H(T|C_i) = -\sum_{j=1}^{k}(\frac{n_{ij}}{n_i})\log(\frac{n_{ij}}{n_i})$$

$$H(T|C) = \sum_{i=1}^{r}\frac{n_i}{n}H(T|C_i)$$

5. Pairwise Measures

$$TP = \frac{1}{2}((\sum_{i=1}^{r}\sum_{j=1}^{k} n_{ij}^2) - n)$$

$$FN = \frac{1}{2}(\sum_{j=1}^{k} m_j^2 - \sum_{i=1}^{r}\sum_{j=1}^{k} n_{ij}^2)$$

$$FP = \frac{1}{2}(\sum_{i=1}^{r} n_i^2 - \sum_{i=1}^{r}\sum_{j=1}^{k} n_{ij}^2)$$

$$TN = \frac{1}{2}(n^2 - \sum_{i=1}^{r} n_i^2 - \sum_{j=1}^{k} m_j^2 - \sum_{i=1}^{r}\sum_{j=1}^{k} n_{ij}^2)$$

$$JaccardCoefficient = \frac{TP}{TP + FN + FP}$$

$$RandStatistic = \frac{TP + TN}{N}$$

For k means

In [58]:

```
contigencyTable = pd.crosstab(kmeans_clusters[1], ground_truth )
print("Contigency Table:\n",contigencyTable)
purity = 1/24 * (contigencyTable.max(0)[0] + contigencyTable.max(0)[1] + contigencyTable.max(0)[2])
print("Purity:\n",purity)
```

```
Contigency Table:
 col_0  0   1  2
row_0
1      3  10  4
2      0   0  7
Purity:
 0.8333333333333333
```

F-measure:
F0 = 2 *3 / 3 +3 = 1*
F1 = 2 10 / 10 + 10 = 1
F2 = 2 *11 / 11+11 = 1*
F = 1/3 (1+1+1) = 1.0

Conditional Entropy:
H(T|C1) = -(3/3)log(1)-0-0 = 0
H(T|C2) = -(10/10)log(1)-0-0 = 0
H(T|C3) = -(11/11)log(1)-0-0 = 0
H(T|C) = (0 *3/24)+(0* 10/24)+(0* 11/24) = 0

Pairwise Measures
TP = 0.5 *((3^2 + 10^2+ 11^2)- 24) = 103*
FN = 0.5 ((3^2 + 10^2+ 11^2) - (3^2 + 10^2+ 11^2)) = 0
FP = 0.5 * ((3^2 + 10^2+ 11^2) - (3^2 + 10^2+ 11^2)) = 0
TN = N - TP - FN - FP = 276 - 103 = 173
Jaccard = 103/ 103 +0+0 = 1
Rand = 103+173/276 = 1

For Spectral Clustering with RBF kernal and Gamma = 0.01

In [154]:

```
contigencyTable = pd.crosstab(spectral_rbf_clusters[1], ground_truth)
print("Contigency Table:\n",contigencyTable)
purity = 1/24 * (contigencyTable.max(0)[0] + contigencyTable.max(0)[1] + contigencyTable.max(0)[2])
print("Purity:\n",purity)
```

```
Contigency Table:
 col_0  0  1  2
row_0
0      3  6  0
1      0  2  6
2      0  2  5
Purity:
 0.625
```

F-measure:
F0 = 2 *6 / 9+10 = 0.63*
F1 = 2 6 / 8+11 = 0.63
F2 = 2 *5 / 7+11 = 0.59*
F = 1/3 (0.63 + 0.63 + 0.58) = 0.61

Conditional Entropy:
H(T|C1) = -(3/9)log(3/9)-(6/9)log(6/9)-0 = 0.918
H(T|C2) = -0-(2/8)log(2/8)-(6/8)log(6/8) = 0.811
H(T|C3) = -0-(2/7)log(2/7)-(5/7)log(5/7) = 0.863
H(T|C) = (0.918 *9/24)+(0.811* 8/24)+(0.863* 7/24) = 0.866

Pairwise Measures
TP = 0.5 *((3^2 + 6^2+ 2^2+ 2^2+6^2+5^2)- 24) = 45*
FN = 0.5 ((3^2 + 10^2+ 11^2) - (3^2 + 6^2+ 2^2+ 2^2+6^2+5^2)) = 58
FP = 0.5 * ((9^2 + 8^2+ 7^2) - (3^2 + 6^2+ 2^2+ 2^2+6^2+5^2)) = 40
TN = N - TP - FN - FP = 276 - 45- 58 - 40 = 133
Jaccard = 45/ 45 +58+40 = 0.315
Rand = 45+133/276 = 0.645

For Spectral Clustering with RBF kernal and Gamma = 0.1

In [133]:

```
contigencyTable = pd.crosstab(spectral_rbf_clusters[6], ground_truth)
print("Contigency Table:\n",contigencyTable)
purity = 1/24 * (contigencyTable.max(0)[0] + contigencyTable.max(0)[1] + contigencyTable.max(0)[2])
print("Purity:\n",purity)
```

```
Contigency Table:
 col_0  0  1  2
row_0
0      0  0  5
1      0  1  6
2      3  9  0
Purity:
 0.75
```

F-measure:

F0 = 2 *5 / 5+11 = 0.625*

F1 = 2 6 / 7+11 = 0.67

F2 = 2 *9 / 12+10 = 0.82*

F = 1/3 (0.625 + 0.67 + 0.82) = 0.705

Conditional Entropy:

H(T|C1) = -0-0-(5/5)log(5/5) = 0

H(T|C2) = -0-(1/7)log(1/7)-(6/7)log(6/7) = 0.592

H(T|C3) = -(3/12)log(3/12)-(9/12)log(9/12)-0 = 0.811

H(T|C) = (0 *9/24)+(0.592* 8/24)+(0.811* 7/24) = 0.434

Pairwise Measures

TP = 0.5 *((3^2 + 6^2+ 1^2+ 9^2+5^2)- 24) = 64*

FN = 0.5 ((3^2 + 10^2+ 11^2) - (3^2 + 6^2+ 1^2+ 9^2+5^2)) = 39

FP = 0.5 * ((5^2 + 12^2+ 7^2) - (3^2 + 6^2+ 1^2+ 9^2+5^2)) = 33

TN = N - TP - FN - FP = 276 - 64- 39 - 33 = 140

Jaccard = 64/ 64 +39+33 = 0.47

Rand = 64+140/276 = 0.739

For Spectral Clustering with 3-NN

In [141]:

```
contigencyTable = pd.crosstab(spectral_knn_clusters[1], ground_truth)
print("Contigency Table:\n",contigencyTable)
purity = 1/24 * (contigencyTable.max(0)[0] + contigencyTable.max(0)[1] + contigencyTable.max(0)[2])
print("Purity:\n",purity)
```

```
Contigency Table:
 col_0  0   1  2
row_0
0      3   0  0
1      0  10  2
2      0   0  9
Purity:
 0.9166666666666666
```

F-measure:
F0 = 2 *3 / 3+3 = 1*
F1 = 2 10 / 12+10 = 0.91
F2 = 2 *9 / 9+11 = 0.90*
F = 1/3 (1.00 + 0.91 + 0.90) = 0.94

Conditional Entropy:
H(T|C1) = -(3/3)log(3/3)-0-0 = 0
H(T|C2) = -0-(10/12)log(10/12)-(2/12)log(2/12) = 0.65
H(T|C3) = -0-0-(9/9)log(9/9) = 0
H(T|C) = 0+(0.65* 12/24)+0 = 0.325

Pairwise Measures
TP = 0.5 *((3^2 + 6^2+ 1^2+ 9^2+5^2)- 24) = 64*
FN = 0.5 ((3^2 + 10^2+ 11^2) - (3^2 + 6^2+ 1^2+ 9^2+5^2)) = 39
FP = 0.5 * ((5^2 + 12^2+ 7^2) - (3^2 + 6^2+ 1^2+ 9^2+5^2)) = 33
TN = N - TP - FN - FP = 276 - 64- 39 - 33 = 140
Jaccard = 64/ 64 +39+33 = 0.47
Rand = 64+140/276 = 0.739

For Spectral Clustering with 5-NN

In [142]:

```
contigencyTable = pd.crosstab(spectral_knn_clusters[6], ground_truth)
print("Contigency Table:\n",contigencyTable)
purity = 1/24 * (contigencyTable.max(0)[0] + contigencyTable.max(0)[1] + contigencyTable.max(0)[2])
print("Purity:\n",purity)
```

```
Contigency Table:
 col_0  0   1  2
row_0
0      0   0  6
1      0   0  5
2      3  10  0
Purity:
 0.7916666666666666
```

F-measure:
F0 = 2 *6 / 6+11 = 0.71*
F1 = 2 5 / 5+11 = 0.625
F2 = 2 *10 / 13+10 = 0.87*
F = 1/3 (0.71 + 0.625 + 0.87) = 0.735

Conditional Entropy:
H(T|C1) = -0-0-(6/6)log(6/6) = 0
H(T|C2) = -0-0-(5/5)log(5/5) = 0
H(T|C3) = -(3/13)log(3/13)-(10/13)log(10/13)-0 = 0.78
H(T|C) = (0 *6/24)+(0* 5/24)+(0.78* 13/24) = 0.4225

Pairwise Measures
TP = 0.5 *((3^2 + 6^2+ 1^2+ 9^2+5^2)- 24) = 64*
FN = 0.5 ((3^2 + 10^2+ 11^2) - (3^2 + 6^2+ 1^2+ 9^2+5^2)) = 39
FP = 0.5 * ((5^2 + 12^2+ 7^2) - (3^2 + 6^2+ 1^2+ 9^2+5^2)) = 33
TN = N - TP - FN - FP = 276 - 64- 39 - 33 = 140
Jaccard = 64/ 64 +39+33 = 0.47
Rand = 64+140/276 = 0.739

ii. Compute the internal measures we studied. You will need the proximity matrix before proceeding.

In [ ]:

```

```

For k-means :

In [144]:

```python
#compute promixity matrix
prox_matrix = np.arange(24*24).reshape(24,24)
for i in range(clustering_dataSet.shape[0]):
    for j in range(clustering_dataSet.shape[0]):
        prox_matrix[i][j] = np.linalg.norm(clustering_dataSet[i]-clustering_dataSet[j])

#compute cluster weight matrix
weight_matrix = np.arange(3*3).reshape(3,3)
C1,C2,C3 = [],[],[]
for i in range(0,24):
    if(kmeans_clusters[1][i] == 0): C1.append(i)
    elif(kmeans_clusters[1][i] == 1): C2.append(i)
    else: C3.append(i)

for i in range(len(C1)):
    for j in range(len(C1)):
        weight_matrix[0][0] += prox_matrix[i][j]

for i in range(len(C2)):
    for j in range(len(C2)):
        weight_matrix[1][1] += prox_matrix[i][j]


for i in range(len(C3)):
    for j in range(len(C3)):
        weight_matrix[2][2] += prox_matrix[i][j]

for i in range(len(C1)):
    for j in range(len(C2)):
        weight_matrix[0][1] += prox_matrix[i][j]

for i in range(len(C1)):
    for j in range(len(C3)):
        weight_matrix[0][2] += prox_matrix[i][j]

for i in range(len(C2)):
    for j in range(len(C1)):
        weight_matrix[1][0] += prox_matrix[i][j]

for i in range(len(C2)):
    for j in range(len(C3)):
        weight_matrix[1][2] += prox_matrix[i][j]

for i in range(len(C3)):
    for j in range(len(C1)):
        weight_matrix[2][0] += prox_matrix[i][j]

for i in range(len(C3)):
    for j in range(len(C2)):
        weight_matrix[2][1] += prox_matrix[i][j]
```

compute Nin and Nout

Nin = 0.5 *(3+10+11)*((3+10+11)-1) = 276

Nout = N - Nin = 276 - 276 = 0

Compute BetaCV

BetaCV = Nout/Nin * Win/Wout = 0

Compute Normalized Cut