friends

March 24, 2025

1 import libraries

[7]:
```
import numpy as np
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
import random

import warnings
warnings.filterwarnings("ignore")
```

2 perform spectral clustering

[8]:
```
A = np.zeros((1495, 1495),dtype=int)

file_name = 'friends.txt'
with open(file_name,'r') as file:
    for line in file:
        node1, node2 = map(int, line.strip().split(','))
        A[node1 - 1][node2 - 1] = 1

L = np.diag(np.sum(A, axis=1)) - A

eigenvalues, eigenvectors = np.linalg.eig(L)
sorted_indices = np.argsort(eigenvalues)
sorted_eigenvalues = eigenvalues[sorted_indices]
sorted_eigenvectors = eigenvectors[:, sorted_indices]

print(np.round(sorted_eigenvalues[:12],3))

for i in range(6):
    v = sorted_eigenvectors[:,i]
    threshold = np.max(np.abs(v)) * 0.9 # dynamically set threshold base on the␣
 ↪values of the eigenvector
    nodes = np.where(np.abs(v) > threshold)
    print("number of nodes in cluster", i+1, ":", len(nodes[0]))
```

```
[-0.    -0.     0.     0.     0.     0.     0.014  0.054  0.074  0.081
```

```
  0.12   0.133]
number of nodes in cluster 1 : 2
number of nodes in cluster 2 : 3
number of nodes in cluster 3 : 2
number of nodes in cluster 4 : 2
number of nodes in cluster 5 : 2
number of nodes in cluster 6 : 1484
```

2.1 interpret the output

- There are 6 zero eigenvalues, which means the graph has 6 connected components
- For each cluster, we can find the number of nodes within that cluster

3 cluster nodes with certain constrains

1. $150 <$ node_size < 750
2. cluster the data into at least 3 groups
3. the conductance between groups must smaller than 0.1
4. The conductance of $S \subset V$ is defined as

$$conductance = \frac{E}{D}$$

where E = number of boundary edges of S, D = total degree of S

[9]:
```
def compute_conductance(A, S):
    n = A.shape[0]
    S_complement = np.setdiff1d(np.arange(n), S) # find the complement of S
    boundary_edges = np.sum(A[np.ix_(S, S_complement)]) # find the number of␣
 ↪boundary edges
    degree_S = np.sum(np.sum(A[S, :], axis=1)) # find the degree of S

    if degree_S == 0:
        return 1
    return boundary_edges / degree_S


def find_low_conductance_sets(A, min_size=150, max_size=750, max_conductance=0.
 ↪1):

    # we need to try the proper combination of eigenvectors to find the best␣
 ↪partition
    # try different number of clusters are also important to avoid some␣
 ↪clusters are too small or too large

    selected_eigenvectors = sorted_eigenvectors[:, 6:15]
    kmeans = KMeans(n_clusters=50, random_state=0)
    labels = kmeans.fit_predict(selected_eigenvectors)


    sets = []
    for i in range(np.max(labels) + 1):
        S = np.where(labels == i)[0]
        if min_size <= len(S) <= max_size:
            conductance = compute_conductance(A, S)
            if conductance <= max_conductance:
                sets.append(S)

    return sets

sets = find_low_conductance_sets(A)
```

```python
for i, S in enumerate(sets):
    print(f"S{i + 1}:")
    print(f"Size: {len(S)}")
    print(f"first 10 members: {S[:10]}")
    print(f"conductance: {compute_conductance(A, S)}")
    print()
```

```
S1:
Size: 198
first 10 members: [ 1  5  7 12 16 20 54 59 66 70]
conductance: 0.008555133079847909

S2:
Size: 550
first 10 members: [ 4  6 11 14 22 23 26 29 33 41]
conductance: 0.012405699916177704

S3:
Size: 506
first 10 members: [ 0  2  8  9 13 15 18 19 21 25]
conductance: 0.042727618220315976
```