Implementation. We implement the spectral clustering algorithm with details as specified. Code is attached in the appendix. Note: the eigenvalue decomposition part (step 3) calls the `scipy.sparse.linalg` library to speed-up large matrix decomposition.

Stochastic block model.

(a) We generate random graphs from the stochastic block model with $n = 10000, b = 10$, and $a \in \{10, 15, 20, 25, 30\}$, one instance for each. Figure 1 shows the histogram of the eigenvalues of $A_G^{cen}$ along with the maximum eigenvalue.

As is seen, the histograms are smooth and roughly obey the semicircle law. In particular, the bulk of eigenvalues is supported on $[-2\sqrt{c}, 2\sqrt{c}]$, with $c = \frac{a+b^2}{2}$ being the average degree. The maximum eigenvalue is clearly separable from the bulk in the last two cases, $a \in \{25, 30\}$. In the other cases, the maximum eigenvalue is hidden in the bulk. As spectral clustering uses the leading eigenvector of $A_G^{cen}$, we expect better detection as we increase $a$.

![Figure 1: Eigenvalue histogram of stochastic block models. The red dot indicates the value of the maximum eigenvalue.](image)
(b) We generate 20 instances with each of the 5 configurations above and run spectral clustering. Table 1 reports the average clustering overlap. Note that the overlap is a linear rescaling of the accuracy from $[\frac{1}{k}, 1]$ to $[0, 1]$.

The mean overlap is higher when $a$ is larger, as we conjectured. When $a \in \{20, 25, 30\}$, spectral clustering captures a non-trivial proportion of the true signal. Note that the information theoretic detection threshold satisfies $a^*-10 = 1$, or $a^* = 17.4$, and we obtain non-trivial detection for all three values above this threshold. In particular, we get a nearly perfect recovery when $a = 30$.

<table>
<thead>
<tr>
<th>a</th>
<th>10</th>
<th>15</th>
<th>20</th>
<th>25</th>
<th>30</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overlap</td>
<td>0.00754</td>
<td>0.01206</td>
<td>0.55065</td>
<td>0.86125</td>
<td>0.95937</td>
</tr>
</tbody>
</table>

Table 1: Mean clustering overlap of spectral clustering on stochastic block models.

Real dataset. We preprocess the data as follows.

1. Read the graph data, remove all duplicate edges (in Python, we modify the networkx.read_gml function to ignore exceptions for duplicates).

2. Compute the adjacency matrix $\tilde{A}$. As the data is a directed graph and so $\tilde{A}$ is asymmetric, we symmetrize it via $A_{ij} = \max(\tilde{A}_{ij} + \tilde{A}_{ji}, 1)$.

3. Remove all isolated nodes, i.e. indices $i$ that satisfy $\sum_j A_{ij} = 0$. Such nodes contain no information at all. As a result, we are left with 1224 nodes, each of which is connected with at least one other node.

(c) Spectral clustering with $A^{cen}$ and the thresholding rule gives a clustering overlap 0.3023.

(d) Spectral clustering with the Laplacian. Note: when we pick large eigenvalues of the Laplacian, we discard all eigenvalues that equal one – those corresponding to connected components. In this problem, one such eigenvalue is discarded. The clustering overlap is 0.0098 – much lower than the simple centering transform. We will explore this more in the next part.

(e) In this part we set $k = 3$ and re-run the Laplacian algorithm. Figure 2 is the scatterplot of $v_1(L_G)$ and $v_2(L_G)$. As we introduce $v_2(L_G)$, we can clearly see the separation between the two classes (on the $y$-axis). It is the second largest non-one eigenvalue that displays the true signal.

We think that it might be worth relaxing the Laplacian algorithm: if $k$ clusters are desired, we can pull out the slightly more than $k$ largest non-one eigenvalues/eigenvectors, in order to capture the real signal. On this data, we perform KMEANS with $k = 2$ on the $(v_1, v_2)$ data. The resulting overlap equals 0.8971, verifying our observation.
Figure 2: Scatterplot of the polblog dataset. The x-axis is $v_1(L_G)$ and the y-axis is $v_2(L_G)$. Red points: true label=0; green points: true label=1.

A Implementation of spectral clustering

```python
import numpy as np
import scipy.sparse.linalg as splinalg
import KMeans as km
import matplotlib.pyplot as plt

""
The Graph transform function
Constructs the matrix M_G for spectral clustering
Inputs:
    A_G: the n*n adjacency matrix
    method: the method of generating M_G
        "cen": centered adjacency matrix
        "lap": normalized Laplacian
Output:
    M_G: the transformed graph matrix
""
def GraphTransform(A_G, method):
    n, _ = A_G.shape
    if method == "cen":
        return A_G - np.mean(A_G) * np.ones((n,n))
    if method == "lap":
        d_G = np.sum(A_G, 1)
        dhat = np.mean(d_G)
```

3
D_G_invhalf = np.diag(np.power(d_G, -0.5))
z_G = np.reshape(np.sqrt(d_G) / np.sqrt(n * dhat), (n,1))
M_G = np.dot(np.dot(D_G_invhalf, A_G), D_G_invhalf) - np.dot(z_G, z_G.T)
return M_G

### Spectral matrix generation
Constructs the matrix $V$ for spectral clustering

**Inputs:**
- $M_G$: the $n \times n$ transformed graph matrix
- $k$: number of groups
- method: method of generating $M_G$

**Output:**
- $V$: $n \times (k-1)$ clustering matrix

### The Spectral Clustering Algorithm
Performs spectral clustering from raw data

**Inputs:**
- $A_G$: the $n \times n$ adjacency matrix
- $k$: number of groups
- mat: method of generating $M_G$
- cluster: clustering method
  - "thres": simple thresholding rule, works only when $k=2$
  - "kmeans": KMeans clustering

**Output:**
- $\sigma$: vector of labels, ranging in $\{0, \ldots, k-1\}$

```python
def SpectralVMatrixGen(M_G, k, method="cen", max_comp=5):
    n, _ = M_G.shape
    if method == "cen":
        _, v = splinalg.eigsh(M_G, k-1)
        return v
    if method == "lap":
        tol = 1e-6
        w, v = splinalg.eigsh(M_G, min(k-1+max_comp, n-1), which="LA")
        print(w)
        for j in range(len(w)-1, -1, -1):
            if w[j] - 1 <= -tol:
                break
        return v[:,range(j,j-k+1,-1)]

def SpectralClustering(A_G, k, mat="cen", cluster="thres"):
    M_G = GraphTransform(A_G, mat)
    V = SpectralVMatrixGen(M_G, k, mat)
    if cluster == "thres":
```
assert(k == 2)
sigma = np.squeeze(V >= 0).astype(int)
if cluster == "kmeans":
    sigma, _, _ = km.KMeans(V, k, np.random.randint(k, size=n))
return sigma

""
The overlap function
Inputs:
    k: number of groups
    sigma_0, sigma_1: label vectors, ranging in {0,...,k-1}
Output:
    Q: the overlap value
""
import itertools
def ClusteringOverlap(sigma_0, sigma_1, k):
    max_accuracy = 0
    for p in itertools.permutations(range(k)):
        sigma_1_permuted = [p[j] for j in sigma_1]
        accuracy = np.mean(sigma_1_permuted == sigma_0)
        if accuracy > max_accuracy:
            max_accuracy = accuracy
    Q = 1.0*k/(k-1) * (max_accuracy - 1.0/k)
    return Q

B Code for part (a), (b)

""
Simulate a two-group stochastic block model
Inputs:
    n: number of vertices
    a, b: within/between group probabilities
Output:
    A_G: the n*n adjacency matrix
    sigma: the true labels
""
def StochasticBlockModel(n, a, b):
    sigma = np.random.randint(2, size=n)
    sigma_pm = np.reshape(2*sigma-1, (n,1))
    P = np.dot(sigma_pm, sigma_pm.T) * (a-b)/2.0/n + (a+b)/2.0/n
    A_G = (np.reshape(np.random.rand(n*n), (n,n)) <= P).astype(int)
    A_G = np.triu(A_G, 1)
    A_G = A_G + A_G.T
    return A_G, sigma

# Experiments
# Stochastic block model
n = 10000
b = 10
a_vec = np.array([10, 15, 20, 25, 30])
num_a = len(a_vec)

# Part (a): eigenvalues of A_G_cen
Eigvals = np.zeros((num_a, n))
for i in range(num_a):
    a = a_vec[i]
    A_G, sigma = StochasticBlockModel(n, a, b)
    A_G_cen = GraphTransform(A_G, "cen")
    Eigvals[i, :] = splinalg.eigvalsh(A_G_cen, overwrite_a=True)
    print("Part (a), a=%d finished." % a)

plt.subplots(2, 3, figsize=(12, 8))
for i in range(num_a):
    plt.subplot('23%d' % (i + 1))
    plt.hist(Eigvals[i, :], bins=50, normed=True, color='gray')
    plt.plot(np.max(Eigvals[i, :]), 0, 'or', markersize=10)
    plt.title('a=%d, b=%d' % (a_vec[i], b))
plt.savefig('sbm_eigvals.png')
plt.close()

# Part (b): repeat and compute empirical overlaps
n_rep = 20
Overlaps = np.zeros((num_a, n_rep))
k = 2
for i in range(num_a):
    a = a_vec[i]
    for j in range(n_rep):
        A_G, sigma = StochasticBlockModel(n, a, b)
        sigma_hat = SpectralClustering(A_G, k)
        Overlaps[i, j] = ClusteringOverlap(sigma, sigma_hat, k)
        print("Part (b), a=%d, rep %d finished with overlap %f" %
              (a, j, Overlaps[i, j]))

mean_overlap = np.mean(Overlaps, 1)

C Code for part (c) - (e)

# Real dataset
import networkx as nx
G = nx.read_gml('polblogs.gml', label='id')
A_G = nx.adjacency_matrix(G).toarray()
A_G = np.fmin(A_G + A_G.T, 1)
sigma = np.array([G.node[key]['value'] for key in G.node])
# Remove isolated nodes
inds = (np.sum(A_G, 1) != 0)
A_G = A_G[np.ix_(inds, inds)]
sigma = sigma[inds]

# Part (c): A_G_cen + thresholding rule
sigma_c = SpectralClustering(A_G, 2)
overlap_c = ClusteringOverlap(sigma, sigma_c, 2)
print(overlap_c)

# Part (d): Lap + thresholding rule
sigma_l = SpectralClustering(A_G, 2, mat="lap")
overlap_l = ClusteringOverlap(sigma, sigma_l, 2)
print(overlap_l)

# Part (e): Lap + k-means
M_G = GraphTransform(A_G, "lap")
V = SpectralVMatrixGen(M_G, 3, "lap")
plt.figure()
colors = [x * "red" + (1-x) * "green" for x in sigma]
plt.scatter(V[:,0], V[:,1], c=colors)
plt.savefig("polblog_spectral.png")
plt.close()

sigma_km, _, _ = km.KMeans(V, 2, np.random.randint(2, size=len(sigma)))
overlap_km = ClusteringOverlap(sigma, sigma_km, 2)
print(overlap_km)