Chapter 17

Spectral and Graph Clustering

Given a dataset $\mathbf{D} = \{\mathbf{x}_i\}_{i=1}^n$ consisting of n points in $\mathbb{R}^d$, let $\mathbf{A}$ denote the $n \times n$ symmetric *similarity matrix* between the points, given as

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \cdots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix} \tag{17.1}$$

where $\mathbf{A}(i,j) = a_{ij} \geq 0$ denotes the similarity or affinity between point $\mathbf{x}_i$ and $\mathbf{x}_j$. Notice how we require the similarity to be symmetric and non-negative.

The matrix $\mathbf{A}$ may be considered to be a *weighted adjacency matrix* of the weighted (undirected) graph $G = (V, E)$, where each vertex is a point, i.e., $V = \{\mathbf{x}_i : i = 1, \cdots, n\}$, and each edge joins a pair of points, i.e., $E = \{(\mathbf{x}_i, \mathbf{x}_j) : 1 \leq i, j \leq n\}$. Further, the similarity matrix $\mathbf{A}$ gives the weight on each edge, i.e., a_{ij} denotes the weight of the edge $(\mathbf{x}_i, \mathbf{x}_j)$. If all affinities are zero or one, then $\mathbf{A}$ represents the regular adjacency relationship between the vertices.

For a vertex $\mathbf{x}_i$, let d_i denote the *degree* of the vertex, defined as

$$d_i = \sum_{j=1}^{n} a_{ij} \tag{17.2}$$

Define the degree matrix $\mathbf{\Delta}$ of the graph G, as the $n \times n$ diagonal matrix

$$\mathbf{\Delta} = \begin{pmatrix} d_1 & 0 & \cdots & 0 \\ 0 & d_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & d_n \end{pmatrix} = \begin{pmatrix} \sum_{j=1}^n a_{1j} & 0 & \cdots & 0 \\ 0 & \sum_{j=1}^n a_{2j} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sum_{j=1}^n a_{nj} \end{pmatrix} \tag{17.3}$$

$\mathbf{\Delta}$ can be compactly written as $\mathbf{\Delta}(i,i) = d_i$ for all $1 \leq i \leq n$.

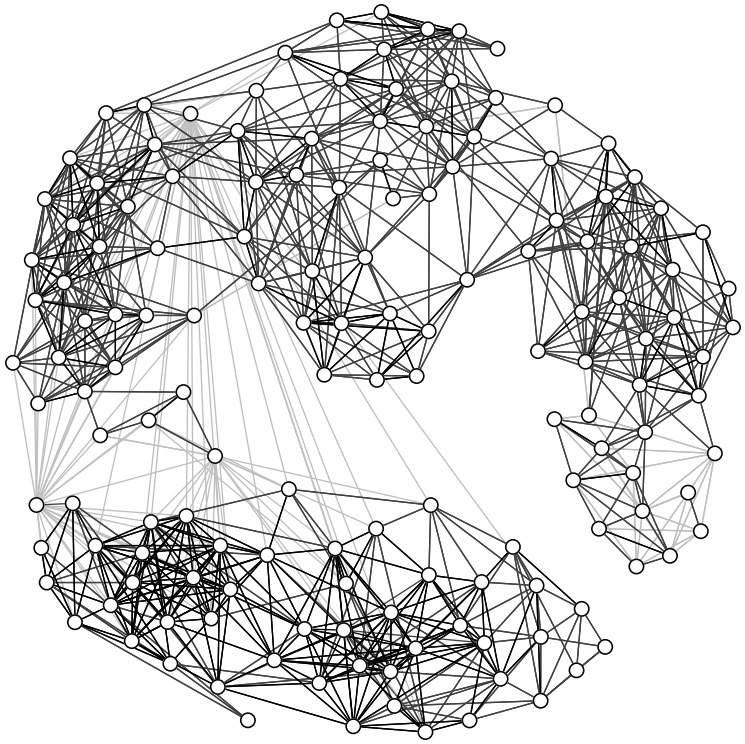


Figure 17.1: Iris Similarity Graph

Example 17.1: Figure 17.1 shows the similarity graph $G = (V, E)$ for the Iris dataset, obtained as follows. Each of the $n = 150$ points $\mathbf{x}_i \in \mathbb{R}^4$ in the Iris dataset is represented by a node in G. To create the edges, we first compute the pair-wise similarity between the points

$$a_{ij} = \exp\left\{-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{2\sigma^2}\right\}$$

where $\sigma = 1$. Next, for each node i we compute the top q nearest neighbors in terms of the similarity value, given as

$$N_q(i) = \{j \in V : \ a_{ij} \leq a_{iq}\}$$

where a_{iq} represents the similarity value between i and its q-th nearest neighbor. We used a value of $q = 16$, since in this case each node records its 15 other neighbors (not including the node itself), which corresponds to 10% of the nodes. An edge is added between nodes i and j only if both i and j are *mutual nearest neighbors*, i.e., if $j \in N_q(i)$ and $i \in N_q(j)$. Finally, if the resulting graph is disconnected, we

add the top q most similar edges between any two connected components. Each edge (i, j) has the weight a_{ij}.

The resulting Iris similarity graph is shown in Figure 17.1. It has $|V| = n = 150$ nodes and $|E| = m = 1730$ edges. The most similar edges (with $a_{ij} \geq 0.95$) are shown in black, the least similar edges (with $a_{ij} < 0.9$) in light gray, and the remaining edges (with $a_{ij} \in [0.9, 0.95)$) in dark gray. Although $a_{ii} = 1.0$ for all nodes, we do not show the self-edges or loops.

Normalized Adjacency Matrix The normalized adjacency matrix is obtained by dividing each row of the adjacency matrix by the degree of the corresponding node. Given the weighted adjacency matrix $\mathbf{A}$ for a graph $G = (V, E)$, its normalized adjacency matrix is defined as

$$\mathbf{M} = \boldsymbol{\Delta}^{-1}\mathbf{A} = \begin{pmatrix} \frac{a_{11}}{d_1} & \frac{a_{12}}{d_1} & \cdots & \frac{a_{1n}}{d_1} \\ \frac{a_{21}}{d_2} & \frac{a_{22}}{d_2} & \cdots & \frac{a_{2n}}{d_2} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{a_{n1}}{d_n} & \frac{a_{n2}}{d_n} & \cdots & \frac{a_{nn}}{d_n} \end{pmatrix} \tag{17.4}$$

Since $\mathbf{A}$ is assumed to have non-negative elements, this implies that each element of $\mathbf{M}$, namely m_{ij} is also non-negative (since $m_{ij} = \frac{a_{ij}}{d_i} \geq 0$). Consider the sum of the i-th row in $\mathbf{M}$

$$\sum_{j=1}^{n} m_{ij} = \sum_{j=1}^{n} \frac{a_{ij}}{d_i} = \frac{d_i}{d_i} = 1 \tag{17.5}$$

Thus, each row in $\mathbf{M}$ sums to 1. This implies that 1 is an eigenvalue of $\mathbf{M}$. In fact, $\lambda_1 = 1$ is the largest eigenvalue of $\mathbf{M}$, and the other eigenvalues satisfy the property that $|\lambda_i| \leq 1$. Also, if G is connected then the eigenvector corresponding to λ_1 is $\mathbf{u}_1 = \frac{1}{\sqrt{n}}(1, 1, \ldots, 1)^T = \frac{1}{\sqrt{n}}\mathbf{1}$.

Example 17.2: Consider the graph in Figure 17.2. Its adjacency and degree matrices are given as

$$\mathbf{A} = \begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \end{pmatrix} \qquad \boldsymbol{\Delta} = \begin{pmatrix} 3 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 4 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 3 \end{pmatrix}$$

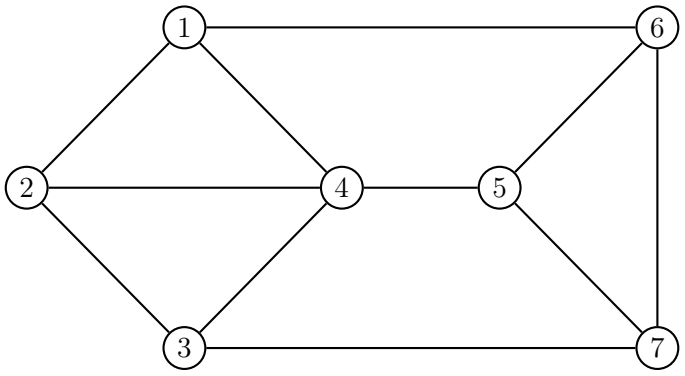


Figure 17.2: Example Graph

The normalized adjacency matrix is as follows

$$\mathbf{M} = \boldsymbol{\Delta}^{-1}\mathbf{A} = \begin{pmatrix} 0 & 0.33 & 0 & 0.33 & 0 & 0.33 & 0 \\ 0.33 & 0 & 0.33 & 0.33 & 0 & 0 & 0 \\ 0 & 0.33 & 0 & 0.33 & 0 & 0 & 0.33 \\ 0.25 & 0.25 & 0.25 & 0 & 0.25 & 0 & 0 \\ 0 & 0 & 0 & 0.33 & 0 & 0.33 & 0.33 \\ 0.33 & 0 & 0 & 0 & 0.33 & 0 & 0.33 \\ 0 & 0 & 0.33 & 0 & 0.33 & 0.33 & 0 \end{pmatrix}$$

The eigenvalues of $\mathbf{M}$ sorted in decreasing are as follows: $\lambda_1 = 1$, $\lambda_2 = 0.483$, $\lambda_3 = 0.206$, $\lambda_4 = -0.045$, $\lambda_5 = -0.405$, $\lambda_6 = -0.539$, $\lambda_7 = -0.7$. The eigenvector corresponding to $\lambda_1 = 1$ is

$$\mathbf{u}_1 = \frac{1}{\sqrt{7}}(1,1,1,1,1,1,1)^T = (0.38, 0.38, 0.38, 0.38, 0.38, 0.38, 0.38)^T$$

Graph Laplacian Matrices The *Laplacian matrix* of the graph is defined as

$$\begin{aligned} \mathbf{L} &= \boldsymbol{\Delta} - \mathbf{A} \qquad (17.6) \\ &= \begin{pmatrix} \sum_{j=1}^n a_{1j} & 0 & \cdots & 0 \\ 0 & \sum_{j=1}^n a_{2j} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sum_{j=1}^n a_{nj} \end{pmatrix} - \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \cdots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix} \end{aligned}$$

$$= \begin{pmatrix} \sum_{j \neq 1} a_{1j} & -a_{12} & \cdots & -a_{1n} \\ -a_{21} & \sum_{j \neq 2} a_{2j} & \cdots & -a_{2n} \\ \vdots & \vdots & \cdots & \vdots \\ -a_{n1} & -a_{n2} & \cdots & \sum_{j \neq n} a_{nj} \end{pmatrix} \tag{17.7}$$

It is interesting to note that $\mathbf{L}$ is a symmetric, positive semi-definite matrix, since for any $\mathbf{c} \in \mathbb{R}^n$, we have

$$\begin{aligned} \mathbf{c}^T \mathbf{L} \mathbf{c} &= \mathbf{c}^T (\mathbf{\Delta} - \mathbf{A}) \mathbf{c} = \mathbf{c}^T \mathbf{\Delta} \mathbf{c} - \mathbf{c}^T \mathbf{A} \mathbf{c} \\ &= \sum_{i=1}^{n} d_i c_i^2 - \sum_{i=1}^{n} \sum_{j=1}^{n} c_i c_j a_{ij} \\ &= \frac{1}{2} \left(\sum_{i=1}^{n} d_i c_i^2 - 2 \sum_{i=1}^{n} \sum_{j=1}^{n} c_i c_j a_{ij} + \sum_{j=1}^{n} d_j c_j^2 \right) \\ &= \frac{1}{2} \left(\sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij} c_i^2 - 2 \sum_{i=1}^{n} \sum_{j=1}^{n} c_i c_j a_{ij} + \sum_{i=j}^{n} \sum_{i=1}^{n} a_{ij} c_j^2 \right) \\ &= \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij} (c_i - c_j)^2 \\ &\geq 0 \qquad \text{since } a_{ij} \geq 0 \text{ and } (c_i - c_j)^2 \geq 0 \end{aligned} \tag{17.8}$$

This means that $\mathbf{L}$ has n real, non-negative eigenvalues, which can be arranged in decreasing order as follows: $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n \geq 0$. Furthermore, from (17.7) we can clearly see that the first column (and the first row) is a linear combination of the remaining columns (rows). That is, if L_i denotes the i-th column of $\mathbf{L}$, then we can observe that $L_1 + L_2 + L_3 + \cdots + L_n = \mathbf{0}$. This implies that the rank of $\mathbf{L}$ is at most $n - 1$, with the smallest eigenvalue $\lambda_n = 0$, and the corresponding eigenvector $\mathbf{u}_n = \frac{1}{\sqrt{n}}(1, 1, \ldots, 1)^T = \frac{1}{\sqrt{n}}\mathbf{1}$, provided the graph is connected. If the graph is disconnected, then the number of eigenvalues equal to zero specifies the number of connected components in the graph.

Example 17.3: Consider the graph in Figure 17.2, whose adjacency and degree matrices are shown in Example 17.2. The graph Laplacian is given as

$$\mathbf{L} = \mathbf{\Delta} - \mathbf{A} = \begin{pmatrix} 3 & -1 & 0 & -1 & 0 & -1 & 0 \\ -1 & 3 & -1 & -1 & 0 & 0 & 0 \\ 0 & -1 & 3 & -1 & 0 & 0 & -1 \\ -1 & -1 & -1 & 4 & -1 & 0 & 0 \\ 0 & 0 & 0 & -1 & 3 & -1 & -1 \\ -1 & 0 & 0 & 0 & -1 & 3 & -1 \\ 0 & 0 & -1 & 0 & -1 & -1 & 3 \end{pmatrix}$$

The eigenvalues of $\mathbf{L}$ are as follows: $\lambda_1 = 5.618$, $\lambda_2 = 4.618$, $\lambda_3 = 4.414$, $\lambda_4 = 3.382$, $\lambda_5 = 2.382$, $\lambda_6 = 1.586$, $\lambda_7 = 0$. The eigenvector corresponding to $\lambda_7 = 0$ is

$$\mathbf{u}_7 = \frac{1}{\sqrt{7}}(1,1,1,1,1,1,1)^T = (0.38, 0.38, 0.38, 0.38, 0.38, 0.38, 0.38)^T$$

The *normalized symmetric Laplacian matrix* of the graph is defined as

$$\begin{aligned} \mathbf{L}^s &= \boldsymbol{\Delta}^{-1/2}\mathbf{L}\boldsymbol{\Delta}^{-1/2} \\ &= \boldsymbol{\Delta}^{-1/2}(\boldsymbol{\Delta} - \mathbf{A})\boldsymbol{\Delta}^{-1/2} = \boldsymbol{\Delta}^{-1/2}\boldsymbol{\Delta}\boldsymbol{\Delta}^{-1/2} - \boldsymbol{\Delta}^{-1/2}\mathbf{A}\boldsymbol{\Delta}^{-1/2} \\ &= \mathbf{I} - \boldsymbol{\Delta}^{-1/2}\mathbf{A}\boldsymbol{\Delta}^{-1/2} \end{aligned} \tag{17.9}$$

where $\boldsymbol{\Delta}^{1/2}$ is the diagonal matrix given as $\boldsymbol{\Delta}^{1/2}(i,i) = \sqrt{d_i}$, and $\boldsymbol{\Delta}^{-1/2}$ is the diagonal matrix given as $\boldsymbol{\Delta}^{-1/2}(i,i) = \frac{1}{\sqrt{d_i}}$ (assuming that $d_i \neq 0$), for $1 \leq i \leq n$. In other words, the normalized Laplacian is given as

$$\begin{aligned} \mathbf{L}^s &= \boldsymbol{\Delta}^{-1/2}\mathbf{L}\boldsymbol{\Delta}^{-1/2} \\ &= \begin{pmatrix} \frac{\sum_{j\neq 1} a_{1j}}{\sqrt{d_1 d_1}} & -\frac{a_{12}}{\sqrt{d_1 d_2}} & \cdots & -\frac{a_{1n}}{\sqrt{d_1 d_n}} \\ -\frac{a_{21}}{\sqrt{d_2 d_1}} & \frac{\sum_{j\neq 2} a_{2j}}{\sqrt{d_2 d_2}} & \cdots & -\frac{a_{2n}}{\sqrt{d_2 d_n}} \\ \vdots & \vdots & \ddots & \vdots \\ -\frac{a_{n1}}{\sqrt{d_n d_1}} & -\frac{a_{n2}}{\sqrt{d_n d_2}} & \cdots & \frac{\sum_{j\neq n} a_{nj}}{\sqrt{d_n d_n}} \end{pmatrix} \end{aligned} \tag{17.10}$$

Like the derivation in (17.8), we can show that $\mathbf{L}^s$ is also positive semi-definite, since for any $\mathbf{c} \in \mathbb{R}^d$, we get

$$\mathbf{c}^T\mathbf{L}^s\mathbf{c} = \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij}\left(\frac{c_i}{\sqrt{d_i}} - \frac{c_j}{\sqrt{d_j}}\right)^2 \geq 0 \tag{17.11}$$

Furthermore, if L_i^s denotes the i-th column of $\mathbf{L}^s$, then from (17.10) we can see that

$$\sqrt{d_1}L_1^s + \sqrt{d_2}L_2^s + \sqrt{d_3}L_3^s + \cdots + \sqrt{d_n}L_n^s = \mathbf{0}$$

That is, the first column is a linear combination of the other columns, which means that $\mathbf{L}^s$ has rank at most $n-1$, with the smallest eigenvalue $\lambda_n = 0$, and the corresponding eigenvector $\frac{1}{\sqrt{\sum_i d_i}}(\sqrt{d_1}, \sqrt{d_2}, \ldots, \sqrt{d_n})^T = \frac{1}{\sqrt{\sum_i d_i}}\boldsymbol{\Delta}^{-1/2}\mathbf{1}$. Combined with the fact that $\mathbf{L}^s$ is positive semi-definite, we conclude that $\mathbf{L}^s$ has n (not necessarily distinct) real, positive eigenvalues $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n = 0$.

Example 17.4: We continue with Example 17.3. For the graph in Figure 17.2, its normalized symmetric Laplacian is given as

$$\mathbf{L}^s = \begin{pmatrix} 1 & -0.33 & 0 & -0.29 & 0 & -0.33 & 0 \\ -0.33 & 1 & -0.33 & -0.29 & 0 & 0 & 0 \\ 0 & -0.33 & 1 & -0.29 & 0 & 0 & -0.33 \\ -0.29 & -0.29 & -0.29 & 1 & -0.29 & 0 & 0 \\ 0 & 0 & 0 & -0.29 & 1 & -0.33 & -0.33 \\ -0.33 & 0 & 0 & 0 & -0.33 & 1 & -0.33 \\ 0 & 0 & -0.33 & 0 & -0.33 & -0.33 & 1 \end{pmatrix}$$

The eigenvalues of $\mathbf{L}^s$ are as follows: $\lambda_1 = 1.7$, $\lambda_2 = 1.539$, $\lambda_3 = 1.405$, $\lambda_4 = 1.045$, $\lambda_5 = 0.794$, $\lambda_6 = 0.517$, $\lambda_7 = 0$. The eigenvector corresponding to $\lambda_7 = 0$ is

$$\begin{aligned} \mathbf{u}_7 &= \frac{1}{\sqrt{22}}(\sqrt{3}, \sqrt{3}, \sqrt{3}, \sqrt{4}, \sqrt{3}, \sqrt{3}, \sqrt{3})^T \\ &= (0.37, 0.37, 0.37, 0.43, 0.37, 0.37, 0.37)^T \end{aligned}$$

The *normalized asymmetric Laplacian* matrix is defined as

$$\begin{aligned} \mathbf{L}^a &= \boldsymbol{\Delta}^{-1}\mathbf{L} \\ &= \boldsymbol{\Delta}^{-1}(\boldsymbol{\Delta} - \mathbf{A}) = \mathbf{I} - \boldsymbol{\Delta}^{-1}\mathbf{A} \\ &= \begin{pmatrix} \frac{\sum_{j\neq 1} a_{1j}}{d_1} & -\frac{a_{12}}{d_1} & \cdots & -\frac{a_{1n}}{d_1} \\ -\frac{a_{21}}{d_2} & \frac{\sum_{j\neq 2} a_{2j}}{d_2} & \cdots & -\frac{a_{2n}}{d_2} \\ \vdots & \vdots & \ddots & \vdots \\ -\frac{a_{n1}}{d_n} & -\frac{a_{n2}}{d_n} & \cdots & \frac{\sum_{j\neq n} a_{nj}}{d_n} \end{pmatrix} \end{aligned} \tag{17.12}$$

Consider the eigenvalue equation for the symmetric Laplacian $\mathbf{L}^s$

$$\mathbf{L}^s\mathbf{u} = \lambda\mathbf{u}$$

Left multiplying by $\boldsymbol{\Delta}^{-1/2}$ on both sides, we get

$$\begin{aligned} \boldsymbol{\Delta}^{-1/2}\mathbf{L}^s\mathbf{u} &= \lambda\boldsymbol{\Delta}^{-1/2}\mathbf{u} \\ \boldsymbol{\Delta}^{-1/2}\boldsymbol{\Delta}^{-1/2}\mathbf{L}\boldsymbol{\Delta}^{-1/2}\mathbf{u} &= \lambda\boldsymbol{\Delta}^{-1/2}\mathbf{u} \\ \boldsymbol{\Delta}^{-1}\mathbf{L}\left(\boldsymbol{\Delta}^{-1/2}\mathbf{u}\right) &= \lambda\left(\boldsymbol{\Delta}^{-1/2}\mathbf{u}\right) \\ \mathbf{L}^a\mathbf{v} &= \lambda\mathbf{v} \end{aligned}$$

This means that $\mathbf{v} = \mathbf{\Delta}^{-1/2}\mathbf{u}$ is an eigenvector of $\mathbf{L}^a$, where $\mathbf{u}$ is an eigenvector of $\mathbf{L}^s$. Furthermore, $\mathbf{L}^a$ has the same set of eigenvalues as $\mathbf{L}^s$, which means that $\mathbf{L}^a$ is a positive semi-definite matrix with n real eigenvalues $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n = 0$. From (17.12) we can see that if L_i^a denotes the i-th column of $\mathbf{L}^a$, then $L_1^a + L_2^a + \cdots + L_n^a = 0$, which implies that $\mathbf{v}_n = \frac{1}{\sqrt{n}}\mathbf{1}$ is the eigenvector corresponding to the smallest eigenvalue $\lambda_n = 0$.

Example 17.5: For the graph in Figure 17.2, its normalized asymmetric Laplacian matrix is given as

$$\mathbf{L}^a = \mathbf{\Delta}^{-1}\mathbf{L} = \begin{pmatrix} 1 & -0.33 & 0 & -0.33 & 0 & -0.33 & 0 \\ -0.33 & 1 & -0.33 & -0.33 & 0 & 0 & 0 \\ 0 & -0.33 & 1 & -0.33 & 0 & 0 & -0.33 \\ -0.25 & -0.25 & -0.25 & 1 & -0.25 & 0 & 0 \\ 0 & 0 & 0 & -0.33 & 1 & -0.33 & -0.33 \\ -0.33 & 0 & 0 & 0 & -0.33 & 1 & -0.33 \\ 0 & 0 & -0.33 & 0 & -0.33 & -0.33 & 1 \end{pmatrix}$$

The eigenvalues of $\mathbf{L}^a$ are identical to those for $\mathbf{L}^s$: $\lambda_1 = 1.7$, $\lambda_2 = 1.539$, $\lambda_3 = 1.405$, $\lambda_4 = 1.045$, $\lambda_5 = 0.794$, $\lambda_6 = 0.517$, $\lambda_7 = 0$. The eigenvector corresponding to $\lambda_7 = 0$ is

$$\mathbf{u}_7 = \frac{1}{\sqrt{7}}(1, 1, 1, 1, 1, 1, 1)^T = (0.38, 0.38, 0.38, 0.38, 0.38, 0.38, 0.38)^T$$

17.1 Clustering as Graph Cuts

A *k-way cut* in a graph is a partitioning or clustering of the vertex set, given as $\mathcal{C} = \{C_1, \cdots, C_k\}$, such that $C_i \neq \emptyset$ for all i, $C_i \cap C_j = \emptyset$ for all i, j, and $V = \bigcup_i C_i$. We require $\mathcal{C}$ to satisfy some objective function, to capture the intuition that nodes within a cluster should have high similarity, and nodes from different clusters should have low similarity.

Given a weighted graph G defined by its similarity matrix (17.1), let $S, T \subseteq V$ be any two subsets of the vertices. We denote by $W(S, T)$ the sum of the weights on all edges with one vertex in S and the other in T, given as

$$W(S, T) = \sum_{v_i \in S} \sum_{v_j \in T} a_{ij} \tag{17.13}$$

Given $S \subseteq V$, we denote by $\overline{S}$ the complementary set of vertices, i.e., $\overline{S} = V - S$. A *(vertex) cut* in a graph is defined as a partitioning of V into $S \subset V$ and $\overline{S}$. The

weight of the cut or *cut-weight* is defined as the sum of all the weights on edges between vertices in S and $\overline{S}$, given as $W(S, \overline{S})$.

Given a clustering $\mathcal{C} = \{C_1, \ldots, C_k\}$ into k clusters, the *size* of a cluster C_i is the number of nodes in the cluster, given as $|C_i|$. The *volume* of a cluster C_i is defined as the sum of all the weights on edges with one end in cluster C_i, given as

$$vol(C_i) = \sum_{v_j \in C_i} d_j = \sum_{v_j \in C_i} \sum_{v_r \in V} a_{jr} = W(C_i, V) \tag{17.14}$$

Let $\mathbf{c}_i \in \mathbb{R}^n$ be the *cluster indicator vector* that records the cluster membership for cluster C_i, given as

$$c_{ij} = \begin{cases} 1 & \text{if } v_j \in C_i \\ 0 & \text{if } v_j \notin C_i \end{cases} \tag{17.15}$$

Since a clustering creates pair-wise disjoint clusters, we immediately have

$$\mathbf{c}_i^T \mathbf{c}_j = 0$$

Furthermore, the cluster size is given as follows

$$|C_i| = \mathbf{c}_i^T \mathbf{c}_i = \|\mathbf{c}_i\|^2 \tag{17.16}$$

The following identities allow us to express the weight of a cut in terms of matrix operations. Let us derive an expression for the sum of the weights for all edges with one end in C_i. These edges include internal cluster edges (with both ends in C_i), as well as external cluster edge (with the other end in another cluster $C_{j \neq i}$).

$$\begin{aligned} vol(C_i) = W(C_i, V) &= \sum_{v_r \in C_i} d_r = \sum_{v_r \in C_i} c_{ir} d_r c_{ir} \\ &= \sum_{r=1}^{n} \sum_{s=1}^{n} c_{ir} \mathbf{\Delta}_{rs} c_{is} = \mathbf{c}_i^T \mathbf{\Delta} \mathbf{c}_i \end{aligned} \tag{17.17}$$

Consider the sum of weights of all internal edges

$$\begin{aligned} W(C_i, C_i) &= \sum_{v_r \in C_i} \sum_{v_s \in C_i} a_{rs} \\ &= \sum_{r=1}^{n} \sum_{s=1}^{n} c_{ir} a_{rs} c_{is} = \mathbf{c}_i^T \mathbf{A} \mathbf{c}_i \end{aligned} \tag{17.18}$$

We can get the sum of weights for all the external edges, or the cut-weight by subtracting (17.18) from (17.17), as follows

$$\begin{aligned} W(C_i, \overline{C_i}) &= \sum_{v_r \in C_i} \sum_{v_s \in V - C_i} a_{rs} = W(C_i, V) - W(C_i, C_i) \\ &= \mathbf{c}_i (\mathbf{\Delta} - \mathbf{A}) \mathbf{c}_i = \mathbf{c}_i^T \mathbf{L} \mathbf{c}_i \end{aligned} \tag{17.19}$$

Example 17.6: Consider the graph in Figure 17.2. Assume that $C_1 = \{1, 2, 3, 4\}$ and $C_2 = \{5, 6, 7\}$ are two clusters. Their cluster indicator vectors are given as

$$\mathbf{c}_1 = (1, 1, 1, 1, 0, 0, 0)^T \qquad \mathbf{c}_2 = (0, 0, 0, 0, 1, 1, 1)^T$$

As required, we have $\mathbf{c}_1^T\mathbf{c}_2 = 0$, and $\mathbf{c}_1^T\mathbf{c}_1 = \|\mathbf{c}_1\|^2 = 4$ and $\mathbf{c}_2^T\mathbf{c}_2 = 3$ give the cluster sizes. Consider the cut-weight between C_1 and C_2. Since there are three edges between the two clusters, we have $W(C_1, \overline{C_1}) = W(C_1, C_2) = 3$. From (17.19), we have

$$\begin{aligned} W(C_1, \overline{C_1}) &= \mathbf{c}_1^T\mathbf{L}\mathbf{c}_1 \\ &= (1,1,1,1,0,0,0) \begin{pmatrix} 3 & -1 & 0 & -1 & 0 & -1 & 0 \\ -1 & 3 & -1 & -1 & 0 & 0 & 0 \\ 0 & -1 & 3 & -1 & 0 & 0 & -1 \\ -1 & -1 & -1 & 4 & -1 & 0 & 0 \\ 0 & 0 & 0 & -1 & 3 & -1 & -1 \\ -1 & 0 & 0 & 0 & -1 & 3 & -1 \\ 0 & 0 & -1 & 0 & -1 & -1 & 3 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} \\ &= (1,0,1,1,-1,-1,-1)(1,1,1,1,0,0,0)^T = 3 \end{aligned}$$

17.1.1 Clustering Objective Functions

The clustering objective function can be formulated as an optimization problem over the k-way cut $\mathcal{C} = \{C_1, \ldots, C_k\}$. There are several commonly used objective functions, such as Average Cut, Ratio Cut and Normalized Cut. We look at these formulations below, incuding another based on the notion of Modularity.

Ratio Cut

The *ratio cut* objective is defined over a k-way cut as follows

$$\min_{\mathcal{C}} \; J_{rc}(\mathcal{C}) = \sum_{i=1}^{k} \frac{W(C_i, \overline{C_i})}{|C_i|} = \sum_{i=1}^{k} \frac{\mathbf{c}_i^T\mathbf{L}\mathbf{c}_i}{\mathbf{c}_i^T\mathbf{c}_i} = \sum_{i=1}^{k} \frac{\mathbf{c}_i^T\mathbf{L}\mathbf{c}_i}{\|\mathbf{c}_i\|^2} \tag{17.20}$$

Ratio cut tries to minimize the sum of the similarities from a cluster C_i to other points not in the cluster $\overline{C_i}$, taking into account the size of each cluster. One can observe that the objective function has a lower value when the cut-weight is minimized, and when the cluster size is large.

Unfortunately, for binary indicator vectors $\mathbf{c}_i$, the ratio cut objective is NP-hard. An obvious relaxation is to allow $\mathbf{c}_i$ to take on any real value. In this case, we can

re-write the objective as

$$\min_{\mathcal{C}} \; J_{rc}(\mathcal{C}) = \sum_{i=1}^{k} \frac{\mathbf{c}_i^T \mathbf{L} \mathbf{c}_i}{\|\mathbf{c}_i\|^2} = \sum_{i=1}^{k} \left(\frac{\mathbf{c}_i}{\|\mathbf{c}_i\|}\right)^T \mathbf{L} \left(\frac{\mathbf{c}_i}{\|\mathbf{c}_i\|}\right) = \sum_{i=1}^{k} \mathbf{u}_i^T \mathbf{L} \mathbf{u}_i \tag{17.21}$$

where $\mathbf{u}_i = \frac{\mathbf{c}_i}{\|\mathbf{c}_i\|}$ is the unit vector in the direction of $\mathbf{c}_i$.

To minimize J_{rc} we take its derivative with respect to $\mathbf{u}_i$ and set it to zero. To incorporate the constraint that $\mathbf{u}_i^T \mathbf{u}_i = 1$, we introduce the Lagrange multiplier λ_i for each cluster C_i. We have

$$\begin{aligned} \frac{\partial}{\partial \mathbf{u}_i} \left(\sum_{i=1}^{k} \mathbf{u}_i^T \mathbf{L} \mathbf{u}_i + \sum_{i=1}^{n} \lambda_i (1 - \mathbf{u}_i^T \mathbf{u}_i) \right) &= 0 \\ \mathbf{L}\mathbf{u}_i - \lambda_i \mathbf{u}_i &= 0 \\ \mathbf{L}\mathbf{u}_i &= \lambda_i \mathbf{u}_i \end{aligned} \tag{17.22}$$

This implies that $\mathbf{u}_i$ is one of the eigenvectors of the Laplacian matrix $\mathbf{L}$, corresponding to the eigenvalue λ_i. Using (17.22), we can see that

$$\mathbf{u}_i^T \mathbf{L} \mathbf{u}_i = \mathbf{u}_i^T \lambda_i \mathbf{u}_i = \lambda_i$$

which in turn implies that to minimize the ratio cut objective (17.21), we should choose the k smallest eigenvalues, and the corresponding eigenvectors, so that

$$\begin{aligned} \min_{\mathcal{C}} \; J_{rc}(\mathcal{C}) &= \mathbf{u}_n^T \mathbf{L} \mathbf{u}_n \; + \; \cdots \; + \; \mathbf{u}_{n-k+1}^T \mathbf{L} \mathbf{u}_{n-k+1} \\ &= \lambda_n \; + \; \cdots \; + \; \lambda_{n-k+1} \end{aligned} \tag{17.23}$$

where we assume that the eigenvalues have been sorted so that $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n$. Thus $\lambda_n \leq \lambda_{n-1} \leq \lambda_{n-k+1}$ are the k smallest eigenvalues with the corresponding eigenvectors $\mathbf{u}_n, \mathbf{u}_{n-1}, \ldots, \mathbf{u}_{n-k+1}$, which represent the relaxed cluster indicator vectors.

Normalized Cut

Normalized cut is similar to ratio cut, except that it divides the cut-weight of each cluster by the volume of the cluster, instead of the size. The objective function is given as

$$\min_{\mathcal{C}} \; J_{nc}(\mathcal{C}) = \sum_{i=1}^{k} \frac{W(C_i, \overline{C_i})}{vol(C_i)} = \sum_{i=1}^{k} \frac{\mathbf{c}_i^T \mathbf{L} \mathbf{c}_i}{\mathbf{c}_i^T \mathbf{\Delta} \mathbf{c}_i} \tag{17.24}$$

The objective function has lower values when the cut-weight is low and when the cluster volume is high.

As in the case of ratio cut, we can obtain an optimal solution to the normalized cut objective if we relax the condition that $\mathbf{c}_i$ be a binary cluster indicator vector. Instead we all $\mathbf{c}_i$ to be an arbitrary real vector. We can re-write the normalized cut objective in terms of the normalized symmetric Laplacian, as follows

$$\begin{aligned}
\min_{\mathcal{C}} \; J_{nc}(\mathcal{C}) &= \sum_{i=1}^{k} \frac{\mathbf{c}_i^T \mathbf{L} \mathbf{c}_i}{\mathbf{c}_i^T \boldsymbol{\Delta} \mathbf{c}_i} \\
&= \sum_{i=1}^{k} \frac{\mathbf{c}_i^T \boldsymbol{\Delta}^{1/2} \boldsymbol{\Delta}^{-1/2} \mathbf{L} \boldsymbol{\Delta}^{-1/2} \boldsymbol{\Delta}^{1/2} \mathbf{c}_i}{\mathbf{c}_i^T \boldsymbol{\Delta}^{1/2} \boldsymbol{\Delta}^{1/2} \mathbf{c}_i} \\
&= \sum_{i=1}^{k} \frac{(\boldsymbol{\Delta}^{1/2} \mathbf{c}_i)^T (\boldsymbol{\Delta}^{-1/2} \mathbf{L} \boldsymbol{\Delta}^{-1/2})(\boldsymbol{\Delta}^{1/2} \mathbf{c}_i)}{(\boldsymbol{\Delta}^{1/2} \mathbf{c}_i)^T (\boldsymbol{\Delta}^{1/2} \mathbf{c}_i)} \\
&= \sum_{i=1}^{k} \left(\frac{\boldsymbol{\Delta}^{1/2} \mathbf{c}_i}{\left\| \boldsymbol{\Delta}^{1/2} \mathbf{c}_i \right\|} \right)^T \mathbf{L}^s \left(\frac{\boldsymbol{\Delta}^{1/2} \mathbf{c}_i}{\left\| \boldsymbol{\Delta}^{1/2} \mathbf{c}_i \right\|} \right) \\
&= \sum_{i=1}^{k} \mathbf{u}_i^T \mathbf{L}^s \mathbf{u}_i
\end{aligned} \tag{17.25}$$

where $\mathbf{u}_i = \frac{\boldsymbol{\Delta}^{1/2} \mathbf{c}_i}{\left\| \boldsymbol{\Delta}^{1/2} \mathbf{c}_i \right\|}$ is the unit vector in the direction of $\boldsymbol{\Delta}^{1/2} \mathbf{c}_i$. In the derivation above we made use of the fact that since $\boldsymbol{\Delta}$ is a diagonal matrix, we have $\boldsymbol{\Delta}^T = \boldsymbol{\Delta}$. We also used the observation that $\mathbf{I} = \boldsymbol{\Delta}^{1/2} \boldsymbol{\Delta}^{-1/2}$ and $\boldsymbol{\Delta} = \boldsymbol{\Delta}^{1/2} \boldsymbol{\Delta}^{1/2}$. Following the same approach as in (17.22), we conclude that the normalized cut objective is optimized by selecting the k smallest eigenvalues of the normalized Laplacian matrix $\mathbf{L}^s$, namely $\lambda_n \leq \cdots \leq \lambda_{n-k+1}$. The corresponding eigenvectors $\mathbf{u}_n, \cdots, \mathbf{u}_{n-k+1}$, with $\mathbf{c}_i = \boldsymbol{\Delta}^{-1/2} \mathbf{u}_i$, represent the real-valued cluster indicator vectors.

The normalized cut objective (17.24), can also be expressed in terms of the normalized asymmetric Laplacian, by differentiating (17.24) with respect to $\mathbf{c}_i$ and setting the result to zero.

$$\begin{aligned}
&\frac{\partial}{\partial \mathbf{c}_i} \left(\frac{\mathbf{c}_i^T \mathbf{L} \mathbf{c}_i}{\mathbf{c}_i^T \boldsymbol{\Delta} \mathbf{c}_i} \right) = 0 \\
&\frac{\mathbf{L}\mathbf{c}_i (\mathbf{c}_i^T \boldsymbol{\Delta} \mathbf{c}_i) - \boldsymbol{\Delta} \mathbf{c}_i (\mathbf{c}_i^T \mathbf{L} \mathbf{c}_i)}{(\mathbf{c}_i^T \boldsymbol{\Delta} \mathbf{c}_i)^2} = 0 \\
&\mathbf{L}\mathbf{c}_i = \left(\frac{\mathbf{c}_i^T \mathbf{L} \mathbf{c}_i}{\mathbf{c}_i^T \boldsymbol{\Delta} \mathbf{c}_i} \right) \boldsymbol{\Delta} \mathbf{c}_i \\
&\boldsymbol{\Delta}^{-1} \mathbf{L} \mathbf{c}_i = \lambda_i \mathbf{c}_i \\
&\mathbf{L}^a \mathbf{c}_i = \lambda_i \mathbf{c}_i
\end{aligned} \tag{17.26}$$

where $\lambda_i = \frac{\mathbf{c}_i^T \mathbf{L} \mathbf{c}_i}{\mathbf{c}_i^T \boldsymbol{\Delta} \mathbf{c}_i}$ is the eigenvalue corresponding to the eigenvector $\mathbf{c}_i$ of $\mathbf{L}^a$. To minimize the normalized cut objective we therefore choose the k smallest eigenvalues

of the normalized asymmetric Laplacian $\mathbf{L}^a$: $\lambda_n \leq \cdots \leq \lambda_{n-k+1}$. The corresponding eigenvectors $\mathbf{u}_n, \cdots, \mathbf{u}_{n-k+1}$, with $\mathbf{c}_i = \mathbf{u}_i$, represent the real-valued cluster indicator vectors.

Average Weight

The *average weight* objective is defined as

$$\max_{\mathcal{C}} \; J_{aw}(\mathcal{C}) = \sum_{i=1}^{k} \frac{W(C_i, C_i)}{|C_i|} = \sum_{i=1}^{k} \frac{\mathbf{c}_i^T \mathbf{A} \mathbf{c}_i}{\mathbf{c}_i^T \mathbf{c}_i} \tag{17.27}$$

where we used the equivalence $W(C_i, C_i) = \mathbf{c}_i^T \mathbf{A} \mathbf{c}_i$ established in (17.18). Instead of trying to minimize the weights on edges between clusters as in ratio cut, average weight tries to maximize the within cluster weights. The problem of maximizing J_{aw} for binary cluster indicator vectors is NP-hard. As in the other objectives, we can obtain a solution by relaxing the constraint on $\mathbf{c}_i$, by assuming that it can take on any real values for its elements. This leads to the relaxed objective

$$\max_{\mathcal{C}} \; J_{aw}(\mathcal{C}) = \sum_{i=1}^{k} \mathbf{u}_i^T \mathbf{A} \mathbf{u}_i \tag{17.28}$$

where $\mathbf{u}_i = \frac{\mathbf{c}_i}{\|\mathbf{c}_i\|}$. Following the same approach as in (17.22), we can maximize the objective by selecting the k largest eigenvalues of $\mathbf{A}$, and the corresponding eigenvectors

$$\begin{aligned} \max_{\mathcal{C}} \; J_{aw}(\mathcal{C}) &= \mathbf{u}_1^T \mathbf{A} \mathbf{u}_1 + \cdots + \mathbf{u}_k^T \mathbf{A} \mathbf{u}_k \\ &= \lambda_1 + \cdots + \lambda_k \end{aligned}$$

Assuming that $\mathbf{A}$ is the weighted adjacency matrix obtained from a symmetric and positive semi-definite kernel, i.e., with $a_{ij} = K(\mathbf{x}_i, \mathbf{x}_j)$, then $\mathbf{A}$ will be positive semi-definite and will have non-negative real eigenvalues. In general, if we threshold $\mathbf{A}$ or if $\mathbf{A}$ is unweighted adjacency matrix for an undirected graph, then even though $\mathbf{A}$ is symmetric, it is not positive semi-definite. This means that in general $\mathbf{A}$ can have negative eigenvalues, though they are all real.

Example 17.7: For the graph in Figure 17.2, with the adjacency matrix shown in Example 17.3, its eigenvalues are as follows: $\lambda_1 = 3.18$, $\lambda_2 = 1.49$, $\lambda_3 = 0.62$, $\lambda_4 = -0.15$, $\lambda_5 = -1.27$, $\lambda_6 = -1.62$, $\lambda_7 = -2.25$. We can see that the eigenvalues can be negative, since $\mathbf{A}$ is the adjacency graph and is not positive semi-definite.

Average Weight and Kernel K-means The average weight objective leads to an interesting connection between kernel K-means and graph cuts. If the weighted adjacency matrix $\mathbf{A}$ represents the kernel value between a pair of points, so that $a_{ij} = K(\mathbf{x}_i, \mathbf{x}_j)$, then we may use the sum of squared errors objective (13.10) of kernel K-means for graph clustering. The SSE objective is given as

$$\begin{aligned}
\min_{\mathcal{C}} \; J_{sse}(\mathcal{C}) &= \sum_{i=1}^{k} \sum_{\mathbf{x}_j \in C_i} \left(K(\mathbf{x}_j, \mathbf{x}_j) - \frac{1}{|C_i|^2} \sum_{\mathbf{x}_r \in C_i} \sum_{\mathbf{x}_s \in C_i} K(\mathbf{x}_r, \mathbf{x}_s) \right) \\
&= \sum_{i=1}^{k} \sum_{v_j \in C_i} \left(a_{jj} - \frac{1}{|C_i|^2} \sum_{v_r \in C_i} \sum_{v_s \in C_i} a_{rs} \right) \\
&= \sum_{j=1}^{n} a_{jj} - \sum_{i=1}^{k} \frac{1}{|C_i|} \sum_{v_r \in C_i} \sum_{v_s \in C_i} a_{rs} \\
&= \sum_{j=1}^{n} a_{jj} - \sum_{i=1}^{k} \frac{\mathbf{c}_i^T \mathbf{A} \mathbf{c}_i}{\mathbf{c}_i^T \mathbf{c}_i} \qquad (17.29)
\end{aligned}$$

We can observe that since $\sum_{j=1}^{n} a_{jj}$ is independent of the clustering, minimizing the SSE objective is the same as maximizing the average weight. In particular, if a_{ij} represents the linear kernel $\mathbf{x}_i^T \mathbf{x}_j$ between the nodes, then the average weight objective (17.27) is equivalent to the regular K-means method (13.1). Thus spectral clustering using J_{aw} and kernel k-means represent two approaches to solve the same problem. Kernel K-means tries to solve the NP-hard problem by using a greedy iterative approach to directly maximize the SSE objective, whereas the graph cut formulation tries to solve the same NP-hard problem by optimally solving a relaxed problem.

Modularity

Informally, modularity is defined as the difference between the observed and expected fraction of edges within a cluster; it measures the extent to which nodes of the same type (in our case, the same cluster) are linked to each other.

Let us assume that the graph G is unweighted, and that $\mathbf{A}$ is its binary adjacency matrix. The number of edges within a cluster C_i is given as

$$\frac{1}{2} \sum_{v_r \in C_i} \sum_{v_s \in C_i} a_{rs}$$

where we divide by $\frac{1}{2}$ since each edge is counted twice in the summation. Over all the clusters, the observed number of edges within the same cluster is given as

$$\frac{1}{2} \sum_{i=1}^{k} \sum_{v_r \in C_i} \sum_{v_s \in C_i} a_{rs} \qquad (17.30)$$

Let us compute the expected number of edges between any two vertices v_r and v_s, assuming that edges are placed at random, and allowing multiple edges between the same pair of vertices. Let $|E| = m$ be the total number of edges in the graph. The probability that one end of an edge is v_r is given as $\frac{d_r}{2m}$. The probability that one end is v_r and the other v_s is then given as

$$p_{rs} = \frac{d_r d_s}{4m^2}$$

The number of edges between v_r and v_s follows a Binomial distribution with success probability p_{rs} over $2m$ trials (since we are selecting the two ends of m edges). The expected number of edges between v_r and v_s is given as

$$2m \cdot p_{rs} = \frac{d_r d_s}{2m}$$

The expected number of edges within a cluster C_i is then

$$\sum_{v_r \in C_i} \sum_{v_s \in C_i} \frac{d_r d_s}{2m}$$

and the expected number of edges within the same cluster is given as

$$\frac{1}{2} \sum_{i=1}^{k} \sum_{v_r \in C_i} \sum_{v_s \in C_i} \frac{d_r d_s}{2m} \tag{17.31}$$

The *modularity* of the clustering $\mathcal{C}$ is defined as the difference between the observed and expected fraction of edges within the same cluster, obtained by subtracting (17.31) from (17.30), and dividing by the number of edges

$$Q = \frac{1}{2m} \sum_{i=1}^{k} \sum_{v_r \in C_i} \sum_{v_s \in C_i} \left(a_{rs} - \frac{d_r d_s}{2m} \right)$$

Since $2m = \sum_{i=1}^{n} d_i$, we can rewrite modularity as follows

$$Q = \sum_{i=1}^{k} \sum_{v_r \in C_i} \sum_{v_s \in C_i} \left(\frac{a_{rs}}{\sum_{j=1}^{n} d_j} - \frac{d_r d_s}{\left(\sum_{j=1}^{n} d_j \right)^2} \right) \tag{17.32}$$

One advantage of this formulation is that it directly generalizes to weighted graphs. We may thus assume that $\mathbf{A}$ is the weighted adjacency matrix, and we interpret the modularity value of a clustering as the difference between the observed and expected fraction of weights on edges within the clusters.

Note that from (17.18) we have

$$\sum_{v_r \in C_i} \sum_{v_s \in C_i} a_{rs} = W(C_i, C_i)$$

and from (17.17), we have

$$\sum_{v_r \in C_i} \sum_{v_s \in C_i} d_r d_s = \left(\sum_{v_r \in C_i} d_r \right) \left(\sum_{v_s \in C_i} d_s \right) = W(C_i, V)^2$$

Further, note that

$$\sum_{j=1}^{n} d_j = W(V, V)$$

Using the above equivalences, can write the modularity objective in terms of the weight function W as follows

$$\max_{\mathcal{C}} \; J_Q(\mathcal{C}) = \sum_{i=1}^{k} \left(\frac{W(C_i, C_i)}{W(V, V)} - \left(\frac{W(C_i, V)}{W(V, V)} \right)^2 \right) \tag{17.33}$$

From (17.18), we have

$$W(C_i, C_i) = \mathbf{c}_i^T \mathbf{A} \mathbf{c}_i$$

Note that

$$W(C_i, V) = \sum_{v_r \in C_i} d_r = \sum_{v_r \in C_i} d_r c_{ir} = \sum_{j=1}^{n} d_j c_{ij} = \sum_{j=1}^{n} \mathbf{d}^T \mathbf{c}_i$$

where $\mathbf{d} = (d_1, d_2, \ldots, d_n)^T$ is the vector of vertex degrees. Further, we have

$$W(V, V) = \sum_{j=1}^{n} d_j = tr(\mathbf{\Delta})$$

The clustering objective based on modularity can then be written as

$$\begin{aligned} \max_{\mathcal{C}} \; J_Q(\mathcal{C}) &= \sum_{i=1}^{k} \left(\frac{\mathbf{c}_i^T \mathbf{A} \mathbf{c}_i}{tr(\mathbf{\Delta})} - \frac{(\mathbf{d}_i^T \mathbf{c}_i)^2}{tr(\mathbf{\Delta})^2} \right) \\ &= \sum_{i=1}^{k} \left(\mathbf{c}_i^T \left(\frac{\mathbf{A}}{tr(\mathbf{\Delta})} \right) \mathbf{c}_i - \mathbf{c}_i^T \left(\frac{\mathbf{d} \cdot \mathbf{d}^T}{tr(\mathbf{\Delta})^2} \right) \mathbf{c}_i \right) \\ &= \sum_{i=1}^{k} \mathbf{c}_i^T \mathbf{Q} \mathbf{c}_i \end{aligned} \tag{17.34}$$

where $\mathbf{Q}$ is the *modularity matrix*

$$\mathbf{Q} = \frac{1}{tr(\mathbf{\Delta})} \left(\mathbf{A} - \frac{\mathbf{d} \cdot \mathbf{d}^T}{tr(\mathbf{\Delta})} \right) \tag{17.35}$$

Directly maximizing the objective for binary cluster vectors $\mathbf{c}_i$ is hard. We resort to the approximation that elements of $\mathbf{c}_i$ can take on real values. Further, we require that $\mathbf{c}_i^T\mathbf{c}_i = \|\mathbf{c}_i\|^2 = 1$ to ensure that J_Q does not increase without bound. Following the approach in (17.22), we conclude that $\mathbf{c}_i$ is an eigenvector of $\mathbf{Q}$. However, since this a maximization problem, instead of selecting the k smallest eigenvalues, we select the k largest eigenvalues and the corresponding eigenvectors to obtain

$$\begin{aligned}\max_{\mathcal{C}}\ J_Q(\mathcal{C}) &= \mathbf{u}_1^T\mathbf{Q}\mathbf{u}_1 + \cdots + \mathbf{u}_k^T\mathbf{Q}\mathbf{u}_k \\ &= \lambda_1 + \cdots + \lambda_k\end{aligned}$$

where $\mathbf{u}_i$ is the eigenvector corresponding to λ_i, and the eigenvalues are sorted so that $\lambda_1 \geq \cdots \geq \lambda_n$. The relaxed cluster indicator vectors are given as $\mathbf{c}_i = \mathbf{u}_i$. Note that the modularity matrix $\mathbf{Q}$ is symmetric, but it is not positive semi-definite. This means that while it has real eigenvalues, they may be negative too. Also note that if Q_i denotes the i-th column of $\mathbf{Q}$, then we have $Q_1 + Q_2 + \cdots + Q_n = \mathbf{0}$, which implies that 0 is an eigenvalue of $\mathbf{Q}$ with the corresponding eigenvector $\frac{1}{\sqrt{n}}\mathbf{1}$. Thus, for maximizing the modularity one should use only the positive eigenvalues.

Modularity and Normalized Adjacency Matrix Consider what happens to the modularity matrix $\mathbf{Q}$ if we use the normalized adjacency matrix $\mathbf{M} = \mathbf{\Delta}^{-1}\mathbf{A}$. In this case, we know by (17.5) that each row of $\mathbf{M}$ sums to 1, i.e.,

$$\sum_{j=1}^{n} m_{ij} = d_i = 1, \text{ for all } i = 1, \ldots, n$$

We immediately have $tr(\mathbf{\Delta}) = \sum_{i=1}^{n} d_i = n$, and further $\mathbf{d} \cdot \mathbf{d}^T = \mathbf{1}_{n\times n}$, where $\mathbf{1}_{n\times n}$ is the $n \times n$ matrix of all ones. The modularity matrix can then be written as

$$\mathbf{Q} = \frac{1}{n}\mathbf{M} - \frac{1}{n^2}\mathbf{1}_{n\times n}$$

However, note that for large graphs, n is large, and the second term practically vanishes since $\frac{1}{n^2}$ will be very small. Thus, the modularity matrix can be reasonably approximated as

$$\mathbf{Q} \simeq \frac{1}{n}\mathbf{M} \tag{17.36}$$

Substituting the above in the modularity objective (17.34), we get

$$\max_{\mathcal{C}} J_Q(\mathcal{C}) = \sum_{i=1}^{k} \mathbf{c}_i^T\mathbf{Q}\mathbf{c}_i = \sum_{i=1}^{k} \mathbf{c}_i^T\mathbf{M}\mathbf{c}_i \tag{17.37}$$

where we have dropped the $\frac{1}{n}$ factor, since it is a constant for a given graph. In other words, if we start with the normalized adjacency matrix, maximizing the modularity

is equivalent to selecting the k largest eigenvalues and the corresponding eigenvectors of the normalized adjacency matrix $\mathbf{M}$.

Example 17.8: Consider the graph in Figure 17.2. The degree vector is $\mathbf{d} = (3, 3, 3, 4, 3, 3, 3)$, and the sum of degrees is $tr(\mathbf{\Delta}) = 22$. The modularity matrix is given as

$$\begin{aligned}
\mathbf{Q} &= \frac{1}{tr(\mathbf{\Delta})}\mathbf{A} - \frac{1}{tr(\mathbf{\Delta})^2}\mathbf{d}\cdot\mathbf{d}^T \\
&= \frac{1}{22}\begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \end{pmatrix} - \frac{1}{484}\begin{pmatrix} 9 & 9 & 9 & 12 & 9 & 9 & 9 \\ 9 & 9 & 9 & 12 & 9 & 9 & 9 \\ 9 & 9 & 9 & 12 & 9 & 9 & 9 \\ 12 & 12 & 12 & 16 & 12 & 12 & 12 \\ 9 & 9 & 9 & 12 & 9 & 9 & 9 \\ 9 & 9 & 9 & 12 & 9 & 9 & 9 \\ 9 & 9 & 9 & 12 & 9 & 9 & 9 \end{pmatrix} \\
&= \begin{pmatrix} -0.019 & 0.027 & -0.019 & 0.021 & -0.019 & 0.027 & -0.019 \\ 0.027 & -0.019 & 0.027 & 0.021 & -0.019 & -0.019 & -0.019 \\ -0.019 & 0.027 & -0.019 & 0.021 & -0.019 & -0.019 & 0.027 \\ 0.021 & 0.021 & 0.021 & -0.033 & 0.021 & -0.025 & -0.025 \\ -0.019 & -0.019 & -0.019 & 0.021 & -0.019 & 0.027 & 0.027 \\ 0.027 & -0.019 & -0.019 & -0.025 & 0.027 & -0.019 & 0.027 \\ -0.019 & -0.019 & 0.027 & -0.025 & 0.027 & 0.027 & -0.019 \end{pmatrix}
\end{aligned}$$

The eigenvalues of $\mathbf{Q}$ are as follows: $\lambda_1 = 0.0678$, $\lambda_2 = 0.0281$, $\lambda_3 = 0$, $\lambda_4 = -0.0068$, $\lambda_5 = -0.0579$, $\lambda_6 = -0.0736$, $\lambda_7 = -0.1024$. The eigenvector corresponding to $\lambda_3 = 0$ is

$$\mathbf{u}_3 = \frac{1}{\sqrt{7}}(1, 1, 1, 1, 1, 1, 1)^T = (0.38, 0.38, 0.38, 0.38, 0.38, 0.38, 0.38)^T$$

17.1.2 Spectral Clustering

For each of the clustering objectives, once we have found the cluster indicator vectors $\mathbf{u}_i$, the main problem we face is that they are not binary, and thus it is not immediately clear how we can assign points to clusters. One solution to this problem is to

treat the $n \times k$ matrix of eigenvectors as a new data matrix

$$\mathbf{U} = \begin{pmatrix} | & | & & | \\ \mathbf{u}_1 & \mathbf{u}_2 & \cdots & \mathbf{u}_k \\ | & | & & | \end{pmatrix} = \begin{pmatrix} u_{11} & u_{21} & \cdots & u_{k1} \\ u_{12} & u_{22} & \cdots & u_{k2} \\ | & | & \cdots & | \\ u_{1n} & u_{2n} & \cdots & u_{kn} \end{pmatrix} = \begin{pmatrix} — & \mathbf{y}_1^T & — \\ — & \mathbf{y}_2^T & — \\ & \vdots & \\ — & \mathbf{y}_n^T & — \end{pmatrix} = \mathbf{Y} \tag{17.38}$$

where each row vector $\mathbf{y}_i$ is normalized to have unit length

$$\mathbf{y}_i = \frac{1}{\sqrt{\sum_{j=1}^k u_{ji}^2}} (u_{1i}, u_{2i}, \ldots, u_{ki})^T$$

We can now cluster the $n \times k$ normalized data matrix $\mathbf{Y}$ into k clusters via the regular K-Means algorithm, since it is expected that the clusters are well-separated in the k-dimensional eigenspace. For this reason, any other fast clustering method can also be used to extract the final clusters. Algorithm 17.1 gives the pseudo-code for the general spectral clustering approach. Depending on the objective function, we choose the corresponding matrix $\mathbf{B}$ (lines 2-5). For instance, for the normalized cut objective, $\mathbf{B}$ is chosen to be either $\mathbf{L}^s$ or $\mathbf{L}^a$. Next we compute the k largest or k smallest eigenvalues and eigenvectors of $\mathbf{B}$, depending on the objective. The final step is to cluster the matrix $\mathbf{Y}$ consisting of the normalized points $\mathbf{y}_i$.

Algorithm 17.1: Spectral Clustering Algorithm

SPECTRAL CLUSTERING $(\mathcal{D} = \{\mathbf{x}_i\}_{i=1}^n, k, J)$:
1 Compute the similarity matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$
`// Choose` $\mathbf{B}$ `according to the objective` J
2 **if** $J = J_{rc}$ **then** $\mathbf{B} = \mathbf{L}$
3 **else if** $J = J_{nc}$ **then** $\mathbf{B} = \mathbf{L}^s$ or $\mathbf{L}^a$
4 **else if** $J = J_Q$ **then** $\mathbf{B} = \mathbf{Q}$ or $\mathbf{M}$
5 **else if** $J = J_{aw}$ **then** $\mathbf{B} = \mathbf{A}$
`// Solve eigenvalue problem and cluster using K-means`
6 Solve $\mathbf{B}\mathbf{u}_i = \lambda_i \mathbf{u}_i$ for $i = 1, \ldots, n$
7 Sort eigenvalues (and corresponding eigenvectors) so that $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n$
8 **if** $J = J_{rc}$ *or* J_{nc} **then** $\mathbf{U} = \begin{pmatrix} \mathbf{u}_n & \mathbf{u}_{n-1} & \cdots & \mathbf{u}_{n-k+1} \end{pmatrix}$
9 **else if** $J = J_Q$ *or* J_{aw} **then** $\mathbf{U} = \begin{pmatrix} \mathbf{u}_1 & \mathbf{u}_2 & \cdots & \mathbf{u}_k \end{pmatrix}$
10 Create normalized dataset $\mathbf{Y} = \{\mathbf{y}_i\}_{i=1,\cdots,n}$ from $\mathbf{U}$
11 Cluster $\mathbf{Y}$ into k clusters $\mathcal{C} = \{C_1, \ldots, C_k\}$, via (weighted) K-Means

Strictly speaking, the normalization step is recommended only for the normalized symmetric Laplacian $\mathbf{L}^s$. This is because the eigenvectors of $\mathbf{L}^s$ and the cluster

indicator vectors are related as $\mathbf{\Delta}^{1/2}\mathbf{c}_i = \mathbf{u}_i$. The j-th entry of $\mathbf{u}_i$, corresponding to vertex v_j is given as

$$u_{ij} = \frac{\sqrt{d_j}c_{ij}}{\sqrt{\sum_{r=1}^{n} d_r c_{ir}^2}}$$

Even in the ideal case when $\mathbf{c}_i$ is a binary vector, we have $u_{ij} = \sqrt{\frac{d_j}{\sum_{r=1}^{n} d_r}}$ if $v_j \in C_i$ (in this case $c_{ij} = 1$). If vertex degrees vary a lot, the vertices with small degrees would have very small values u_{ij}. This can cause problems for the K-means step in correctly clustering these vertices. The normalization step helps alleviate this problem for $\mathbf{L}^s$, though it can also help other objectives. Also note that for the modularity objective, it may help to use weighted K-means, where the dimensions in $\mathbf{Y}$ are weighted from high to low, so that $\mathbf{u}_1$ is weighted the highest, with decreasing contributions from $\mathbf{u}_2, \ldots, \mathbf{u}_k$.

Computational Complexity The computational complexity of the spectral clustering algorithm is $O(n^3)$, since computing all the eigenvectors takes that much time. If the matrix is sparse, the complexity is $O(n^2)$. Running the K-means method on $\mathbf{Y}$ takes $O(tnk^2)$ time, where t is the number of iterations until convergence.

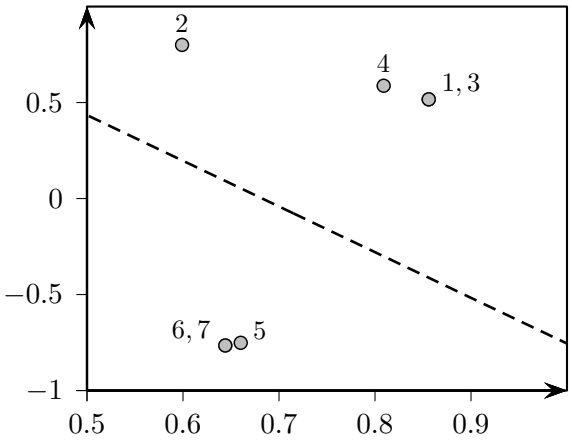


Figure 17.3: K-means on Spectral Dataset $\mathbf{Y}$

Example 17.9: Consider the normalized cut approach applied to the graph in Figure 17.2. Assuming we want to find $k = 2$ clusters, for the symmetric Laplacian shown in Example 17.4, we compute the eigenvectors, $\mathbf{v}_7$ and $\mathbf{v}_6$, corresponding to the two smallest eigenvalues, $\lambda_7 = 0$ and $\lambda_6 = 0.517$. The matrix composed of

both the eigenvectors is given as

$$\mathbf{U} = \begin{pmatrix} \mathbf{u}_1 & \mathbf{u}_2 \\ \hline 0.369 & 0.223 \\ 0.369 & 0.494 \\ 0.369 & 0.223 \\ 0.426 & 0.310 \\ 0.369 & -0.420 \\ 0.369 & -0.439 \\ 0.369 & -0.439 \end{pmatrix}$$

We treat the i-th component of $\mathbf{u}_1$ and $\mathbf{u}_2$ as the i-th point $(u_{1i}, u_{2i}) \in \mathbb{R}^2$, and after normalizing all points to have unit length we obtain the new dataset

$$\mathbf{Y} = \begin{pmatrix} 0.856 & 0.517 \\ 0.599 & 0.800 \\ 0.856 & 0.517 \\ 0.809 & 0.588 \\ 0.660 & -0.751 \\ 0.644 & -0.765 \\ 0.644 & -0.765 \end{pmatrix}$$

For instance the first point is computed as

$$\mathbf{y}_1 = \frac{1}{\sqrt{(0.369)^2 + (0.223^2)}}(0.369, 0.223)^T = (0.856, 0.517)^T$$

Clustering $\mathbf{Y}$ into $k = 2$ clusters yields $C_1 = \{1, 2, 3, 4\}$ and $C_2 = \{5, 6, 7\}$, as shown in Figure 17.3

	`iris-setosa`	`iris-virginica`	`iris-versicolor`
C_1 (square)	50	0	4
C_2 (circle)	0	36	0
C_3 (triangle)	0	14	46

Table 17.1: Contingency Table: Clusters versus Iris Types

Example 17.10: Figure 17.4 shows the $k = 3$ clusters obtained for the iris graph in Figure 17.1, using the normalized cut objective. Comparing the clusters with the iris class attribute (not used in clustering), we obtain the contingency table

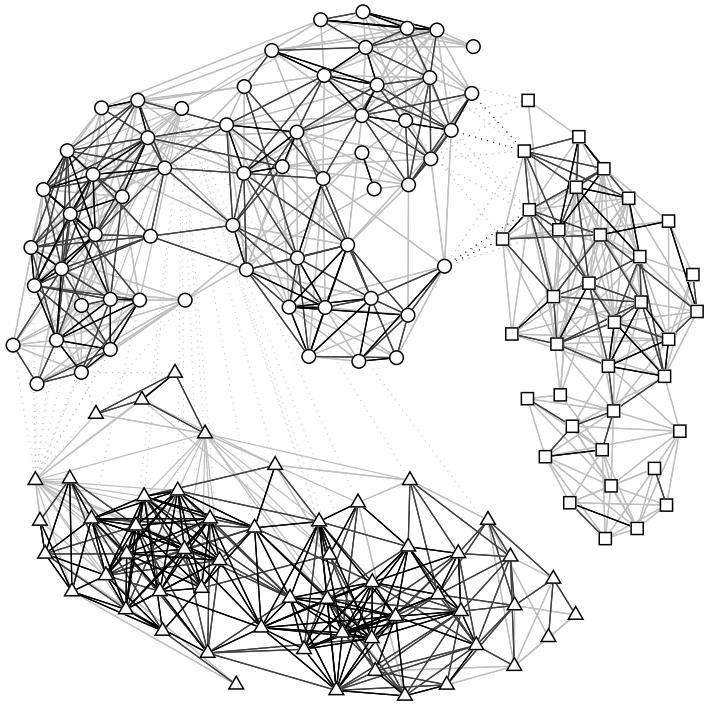


Figure 17.4: Normalized Cut on Iris Graph

shown in Table 17.1, indicating the number of points clustered correctly (on the main diagonal) and incorrectly (off-diagonal). We can see that clusters C_1 is mainly corresponds to `iris-setosa`, C_2 to `iris-virginica` and C_3 to `iris-versicolor`. The latter two are more difficult to separate.

17.2 Markov Clustering

We now consider a graph clustering method based on simulating a random walk on a weighted graph. The basic intuition is that if transitions reflect the weights on the edges, then transitions from one node to another within a cluster are much more likely than transitions between nodes from different clusters. This is because nodes within a cluster have higher similarities or weights, and nodes across clusters have lower similarities.

Given the weighted adjacency matrix $\mathbf{A}$ for a graph $G = (V, E)$, the normalized adjacency matrix (17.4) is given as $\mathbf{M} = \mathbf{\Delta}^{-1}\mathbf{A}$. $\mathbf{M}$ can be interpreted as the $n \times n$ *transition matrix* where the entry $m_{ij} = \frac{a_{ij}}{d_i}$, given as the ratio of the weight a_{ij} on the edge (v_i, v_j) to the degree d_i of v_i, can be interpreted as the probability of transitioning or jumping from node i to node j in the graph G. This is because $\mathbf{M}$

is a *row stochastic* or *Markov* matrix, that is, a matrix which satisfies the condition that each matrix element $m_{ij} \geq 0$, and each row adds up to 1. A vector with non-negative elements that sum up to 1 is also called a *probability vector*. Thus the rows of $\mathbf{M}$ are all probability vectors. Since $\mathbf{A}$ is assumed to have non-negative elements, we also have $m_{ij} \geq 0$. Consider the sum of the i-th row in $\mathbf{M}$

$$\sum_{j=1}^{n} m_{ij} = \sum_{j=1}^{n} \frac{a_{ij}}{d_i} = 1$$

Thus, $\mathbf{M}$ is indeed a Markov matrix.

$\mathbf{M}$ is thus the transition matrix for a *Markov chain* or a Markov random walk on $G = (V, E)$. A Markov chain is a discrete-time stochastic process over a set of states, in our case the set of vertices V. The Markov chain makes a transition from one node to another at discrete time-steps $t = 1, 2, \ldots$, with the probability of making a transition from node i to node j given as m_{ij}. Let the random variable X_t denote the state at time t. The Markov property means that the probability distribution of X_t, over the states at time t, depends only on the probability distribution of X_{t-1}, i.e.,

$$P(X_t = i | X_0, X_1, \ldots, X_{t-1}) = P(X_t = i | X_{t-1})$$

Further, we assume that the Markov chain is *homogeneous*, i.e., the transition probability

$$P(X_t = j | X_{t-1} = i) = m_{ij}$$

is independent of the time-step t.

Given a node i, the transition matrix $\mathbf{M}$ specifies the probabilities of reaching any other node j in one time-step. Starting from node i at $t = 0$, let us consider the probability of being at node j at $t = 2$. We denote by $m_{ij}(2)$ the probability of reaching j from i in two time-steps. We can compute this as follows

$$\begin{aligned} m_{ij}(2) = P(X_2 = j | X_0 = i) &= \sum_{a=1}^{n} P(X_1 = a | X_0 = i) P(X_2 = j | X_1 = a) \\ &= \sum_{a=1}^{n} m_{ia} m_{aj} = \mathbf{m}_i^T M_j \end{aligned} \tag{17.39}$$

where $\mathbf{m}_i = (m_{i1}, m_{i2}, \ldots, m_{in})^T$ denotes the column vector corresponding to the i-th row of $\mathbf{M}$ and $M_j = (m_{1j}, m_{2j}, \ldots, m_{nj})^T$ denotes the column vector corresponding to the j-th column of $\mathbf{M}$.

Consider the product of $\mathbf{M}$ with itself

$$\mathbf{M}^2 = \mathbf{M} \cdot \mathbf{M} = \begin{pmatrix} — & \mathbf{m}_1^T & — \\ — & \mathbf{m}_2^T & — \\ & \vdots & \\ — & \mathbf{m}_n^T & — \end{pmatrix} \begin{pmatrix} | & | & & | \\ M_1 & M_2 & \cdots & M_n \\ | & | & & | \end{pmatrix}$$

$$= \left\{ \mathbf{m}_i^T M_j \right\}_{i,j=1}^{n} = \left\{ m_{ij}(2) \right\}_{i,j=1}^{n} \tag{17.40}$$

We conclude that the matrix $\mathbf{M}^2$ is precisely the transition probability matrix for the Markov chain over two time-steps. Using (17.40), based on the Markov property, the transition probability matrix of transitioning from any node i to any node j in t time-steps is given as

$$\mathbf{M}^t = \mathbf{M} \cdot \mathbf{M}^{t-1} \tag{17.41}$$

A basic random walk on G corresponds to taking successive powers of the transition matrix $\mathbf{M}$. Let $\boldsymbol{\pi}_0$ specify the state probability vector at time t_0, that is, $\boldsymbol{\pi}_{0i} = P(X_0 = i)$, specifies the probability of starting at node i for all $i = 1, \ldots, n$. Starting from $\boldsymbol{\pi}_0$, we can obtain the state probability vector for X_t, which specifies the probability of being at node i at time-step t, as follows

$$\boldsymbol{\pi}_t^T = \boldsymbol{\pi}_{t-1}^T \mathbf{M} = \boldsymbol{\pi}_{t-2}^T \mathbf{M}^2 = \cdots = \boldsymbol{\pi}_0^T \mathbf{M}^t$$

Equivalently, taking transpose on both sides, we get

$$\boldsymbol{\pi}_t = (M^t)^T \boldsymbol{\pi}_0 = (\mathbf{M}^T)^t \boldsymbol{\pi}_0$$

The state probability vector thus eventually converges to the dominant eigenvector of $\mathbf{M}^T$, reflecting the steady-state probability of reaching any node in the graph, regardless of the starting node. Note that if the graph is directed, then the steady-state vector is equivalent to the normalized prestige vector (4.24).

Transition Probability Inflation We now consider a variation of the random walk, where the probability of transitioning from node i to j is inflated by taking each element m_{ij} to the power $r \geq 1$. Given a transition matrix $\mathbf{M}$, define the inflation operator Υ, given as follows

$$\Upsilon(\mathbf{M}, r) = \left\{ \frac{(m_{ij})^r}{\sum_{a=1}^{n} (m_{ia})^r} \right\}_{i,j=1}^{n} \tag{17.42}$$

The inflation operation results in a transformed or inflated transition probability matrix, since the elements remain non-negative, and each row is normalized to sum to 1. The net effect of the inflation operator is to increase the higher probability transitions, and decrease the lower probability transitions.

Algorithm 17.2: Markov Clustering Algorithm

```
MARKOV CLUSTERING (A, r, ε):
t = 0
Add self-edges to A if they do not exist
M_t = Δ^{-1} A
repeat
    t = t + 1
    M_t = M_{t-1} · M_{t-1}
    M_t = Υ(M_t, r)
until ‖M_t − M_{t-1}‖_F ≤ ε
G_t = directed graph induced by M_t
C = {weakly connected components in G_t}
```

Markov Clustering The Markov clustering (MCL) method is an iterative method that interleaves matrix expansion and inflation steps. Matrix expansion corresponds to taking successive powers of the transition matrix, leading to random walks of longer lengths. On the other hand, matrix inflation makes the higher probability transitions even more likely, and reduces the lower probability transitions. Since nodes in the same cluster are expected to have higher weights, and consequently higher transitions probabilities between them, the inflation operator makes it more likely to stay within the cluster. It thus limits the extent of the random walk.

The pseudo-code for MCL is given in Algorithm 17.2. The method works on the weighted adjacency matrix for a graph. Unlike some of the other graph clustering methods, MCL does not use the parameter k for the number of output clusters. Instead, it uses only the inflation parameter $r > 1$. Higher values lead to more, smaller clusters, whereas smaller values lead to fewer, larger clusters. However, the exact number of clusters cannot be guaranteed. Given the adjacency matrix $\mathbf{A}$, MCL first adds *loops* or self-edges to $\mathbf{A}$ if they do not exist. If $\mathbf{A}$ is a similarity matrix, then this is not required, since a node is most similar to itself, and thus $\mathbf{A}$ should have high values on the diagonals. For simple, undirected graphs, if $\mathbf{A}$ is the adjacency matrix, then adding self-edges associates return probabilities with each node.

The iterative MCL expansion and inflation process stops when the transition matrix converges, i.e., when the differences between the transition matrix from two successive iterations falls below some threshold $\epsilon \geq 0$. The matrix difference is given in terms of the *Frobenius norm*, given as

$$\|\mathbf{M}_t - \mathbf{M}_{t-1}\|_F = \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{n}(m_{ij_t} - m_{ij_{t-1}})^2}$$

where $\mathbf{M}_t(i,j) = m_{ij_t}$ denotes the entry in the i-th row and j-th column in $\mathbf{M}_t$. The

MCL process stops when $\|\mathbf{M}_t - \mathbf{M}_{t-1}\|_F \leq \epsilon$.

The clusters are found by enumerating the weakly connected components in the directed graph induced by the converged transition matrix $\mathbf{M}_t$. The directed graph induced by $\mathbf{M}_t$ is denoted as $G_t = (V_t, E_t)$. The vertex set is the same as the set of nodes in the original graph, i.e., $V_t = V$, and the edge set is given as

$$E_t = \{(i, j) : m_{ij_t} > 0\}$$

In other words, a directed edge (i, j) exists only if node i can transition to node j at step t of the expansion and inflation process. A node i is called an *attractor* if $m_{ii_t} > 0$. The MCL process yields a set of attractor nodes, $V_a \subseteq V$, such that other nodes are attracted to at least one attractor in V_a, i.e., for all nodes i, there exists a node $j \in V_a$, such that $(i, j) \in E_t$. A strongly connected component in a directed graph is defined a maximal subgraph such that there exists a directed path between all pairs of vertices in the subgraph. To extract the clusters from G_t, MCL first finds the strongly connected components $S_1, S_2, \ldots, S_q$ over the set of attractors V_a. Next, for each strongly connected set of attractors S_i, MCL finds the weakly connected components consisting of all nodes $j \in V_t - V_a$ attracted to a node in S_i. If a node j is attracted to multiple strongly connected components, it is added to each such cluster, resulting in possibly overlapping clusters.

Computational Complexity The computational complexity of the MCL algorithm is clearly $O(tn^3)$, where t is the number of iterations until convergence. This follows from the fact that whereas the inflation operation takes $O(n^2)$ time, the expansion operation requires matrix multiplication, which takes $O(n^3)$ time. However, the matrices become sparse very quickly, and it is possible to use sparse matrix multiplication to obtain $O(n^2)$ complexity for expansion in later iterations. Upon convergence, the weakly connected components in G_t can be found in $O(n + m)$, where m is the number of edges. Since G_t is very sparse, with $m = O(n)$, the final clustering step takes $O(n)$ time.

Example 17.11: We apply the MCL method to find $k = 2$ clusters in Figure 17.2. We add the self-loops to the graph to obtain the adjacency matrix

$$\mathbf{A} = \begin{pmatrix} 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{pmatrix}$$

The corresponding Markov matrix is given as

$$\mathbf{M}_0 = \mathbf{\Delta}^{-1}\mathbf{A} = \begin{pmatrix} 0.25 & 0.25 & 0 & 0.25 & 0 & 0.25 & 0 \\ 0.25 & 0.25 & 0.25 & 0.25 & 0 & 0 & 0 \\ 0 & 0.25 & 0.25 & 0.25 & 0 & 0 & 0.25 \\ 0.20 & 0.20 & 0.20 & 0.20 & 0.20 & 0 & 0 \\ 0 & 0 & 0 & 0.25 & 0.25 & 0.25 & 0.25 \\ 0.25 & 0 & 0 & 0 & 0.25 & 0.25 & 0.25 \\ 0 & 0 & 0.25 & 0 & 0.25 & 0.25 & 0.25 \end{pmatrix}$$

In the first iteration, we apply expansion and then inflation (with $r = 2.5$) to obtain

$$\mathbf{M}_1 = \mathbf{M}_0 \cdot \mathbf{M}_0 = \begin{pmatrix} 0.237 & 0.175 & 0.113 & 0.175 & 0.113 & 0.125 & 0.062 \\ 0.175 & 0.237 & 0.175 & 0.237 & 0.050 & 0.062 & 0.062 \\ 0.113 & 0.175 & 0.237 & 0.175 & 0.113 & 0.062 & 0.125 \\ 0.140 & 0.190 & 0.140 & 0.240 & 0.090 & 0.100 & 0.100 \\ 0.113 & 0.050 & 0.113 & 0.113 & 0.237 & 0.188 & 0.188 \\ 0.125 & 0.062 & 0.062 & 0.125 & 0.188 & 0.250 & 0.188 \\ 0.062 & 0.062 & 0.125 & 0.125 & 0.188 & 0.188 & 0.250 \end{pmatrix}$$

$$\mathbf{M}_1 = \Upsilon(\mathbf{M}_1, 2.5) = \begin{pmatrix} 0.404 & 0.188 & 0.062 & 0.188 & 0.062 & 0.081 & 0.014 \\ 0.154 & 0.331 & 0.154 & 0.331 & 0.007 & 0.012 & 0.012 \\ 0.062 & 0.188 & 0.404 & 0.188 & 0.062 & 0.014 & 0.081 \\ 0.109 & 0.234 & 0.109 & 0.419 & 0.036 & 0.047 & 0.047 \\ 0.060 & 0.008 & 0.060 & 0.060 & 0.386 & 0.214 & 0.214 \\ 0.074 & 0.013 & 0.013 & 0.074 & 0.204 & 0.418 & 0.204 \\ 0.013 & 0.013 & 0.074 & 0.074 & 0.204 & 0.204 & 0.418 \end{pmatrix}$$

MCL converges in 10 iterations (using $\epsilon = 0.001$), with the final transition matrix

$$\mathbf{M} = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.5 & 0.5 \\ 0 & 0 & 0 & 0 & 0 & 0.5 & 0.5 \\ 0 & 0 & 0 & 0 & 0 & 0.5 & 0.5 \end{pmatrix}$$

We can observe that m_{44}, m_{66}, and m_{77} are all greater than zero, making nodes 4, 6 and 7 the three attractors. Since both 6 and 7 can reach each other, the equivalence classes of attractors are $\{4\}$ and $\{6, 7\}$. Nodes $1, 2, 3$ are attracted to 4, and node 5 is attracted to both 6 and 7. Figure 17.5 illustrates the attractors (self-loops are not shown). One can clearly observe the two weakly connected components, which make up the two clusters $C_1 = \{1, 2, 3, 4\}$ and $C_2 = \{5, 6, 7\}$.

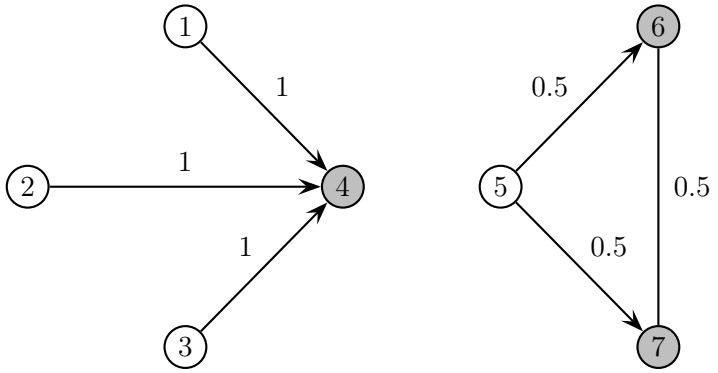
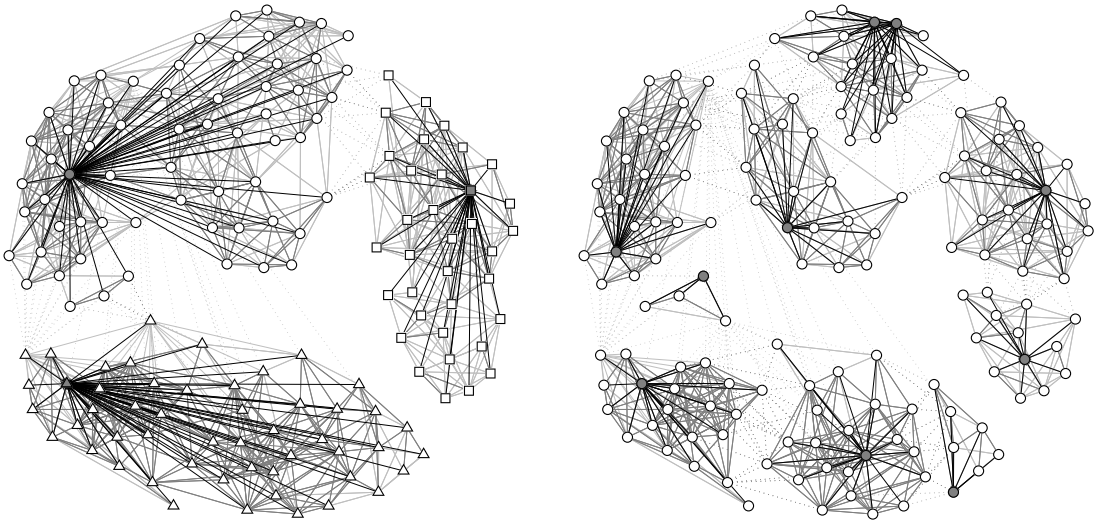


Figure 17.5: MCL Attractors and Clusters



(a) $r = 1.3$ (b) $r = 2$

Figure 17.6: MCL on Iris Graph

	`iris-setosa`	`iris-virginica`	`iris-versicolor`
C_1 (triangle)	50	0	1
C_2 (square)	0	36	0
C_3 (circle)	0	14	49

Table 17.2: Contingency Table: MCL Clusters versus Iris Types

Example 17.12: Figure 17.6a shows the clusters obtained via the MCL algorithm on the Iris graph, using $r = 1.3$ in the inflation step. MCL yields three attractors (shown as gray nodes), which separate the graph into three clusters. The

contingency table for the discovered clusters versus the true iris types is given in Table 17.2. Compared to the results from the normalized cut, the only difference is that only one point with class `iris-versicolor`, is (wrongly) grouped with `iris-setosa` in C_1.

Notice that the only parameter for MCL is r the exponent for the inflation step. The number of clusters is not explicitly specified, but higher values of r result in more clusters. The value of $r = 1.3$ was used above, since it resulted in three clusters. Figure 17.6b shows what happens if $r = 2$. MCL yields 9 clusters, where one of the clusters has two attractors.

17.3 Annotated References

17.4 Exercises

1. Show that if Q_i denotes the i-th column of the modularity matrix $\mathbf{Q}$, then $\sum_{i=1}^{n} Q_i = \mathbf{0}$.