Data Mining and Machine Learning: Fundamental Concepts and Algorithms

dataminingbook.info

Mohammed J. Zaki[1] Wagner Meira Jr.[2]

[1]Department of Computer Science
Rensselaer Polytechnic Institute, Troy, NY, USA

[2]Department of Computer Science
Universidade Federal de Minas Gerais, Belo Horizonte, Brazil

Chapter 16: Spectral & Graph Clustering

Graphs and Matrices: Adjacency Matrix

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

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

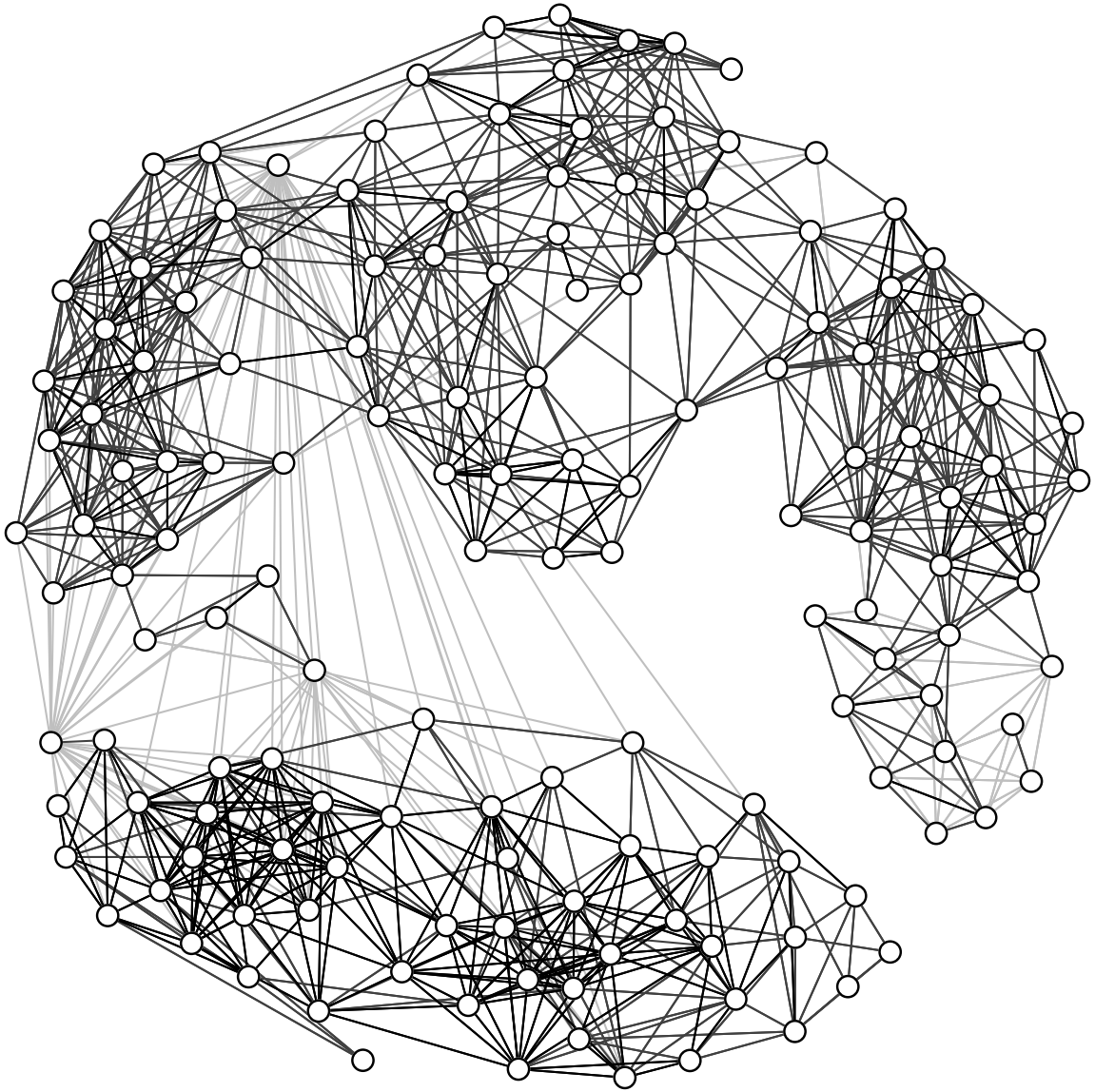
where $\boldsymbol{A}(i,j) = a_{ij}$ denotes the similarity or affinity between points $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$.

We require the similarity to be symmetric and non-negative, that is, $a_{ij} = a_{ji}$ and $a_{ij} \geq 0$, respectively.

The matrix $\boldsymbol{A}$ is the *weighted adjacency matrix* for the data graph. If all affinities are 0 or 1, then $\boldsymbol{A}$ represents the regular adjacency relationship between the vertices.

Iris Similarity Graph: Mutual Nearest Neighbors

$|V| = n = 150$, $|E| = m = 1730$



Graphs and Matrices: Degree Matrix

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

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

We define the *degree matrix* Δ of graph G as the $n \times n$ diagonal matrix:

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

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

Graphs and Matrices: 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 $\boldsymbol{A}$ for a graph G, its normalized adjacency matrix is defined as

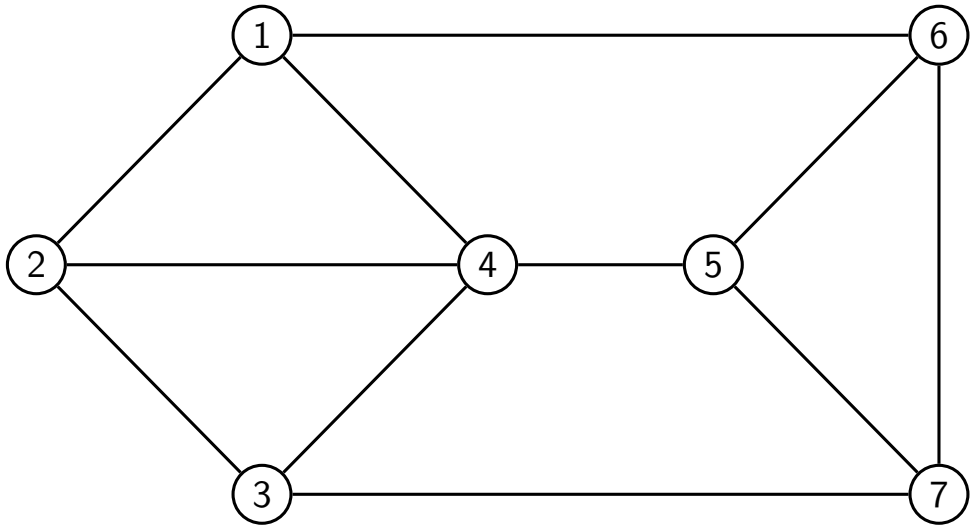
$$\boldsymbol{M} = \Delta^{-1}\boldsymbol{A} = \begin{pmatrix} \frac{a_{11}}{d_1} & \frac{a_{12}}{d_1} & \cdots & \frac{a_{1n}}{d_1} \\ \frac{a_{21}}{d_2} & \frac{a_{22}}{d_2} & \cdots & \frac{a_{2n}}{d_2} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{a_{n1}}{d_n} & \frac{a_{n2}}{d_n} & \cdots & \frac{a_{nn}}{d_n}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

Because $\boldsymbol{A}$ is assumed to have non-negative elements, this implies that each element of $\boldsymbol{M}$, namely m_{ij} is also non-negative, as $m_{ij} = \frac{a_{ij}}{d_i} \geq 0$.

Each row in $\boldsymbol{M}$ sums to 1, which implies that 1 is an eigenvalue of $\boldsymbol{M}$. In fact, $\lambda_1 = 1$ is the largest eigenvalue of $\boldsymbol{M}$, and the other eigenvalues satisfy the property that $|\lambda_i| \leq 1$. Because $\boldsymbol{M}$ is not symmetric, its eigenvectors are not necessarily orthogonal.

Example Graph

Adjacency and Degree Matrices

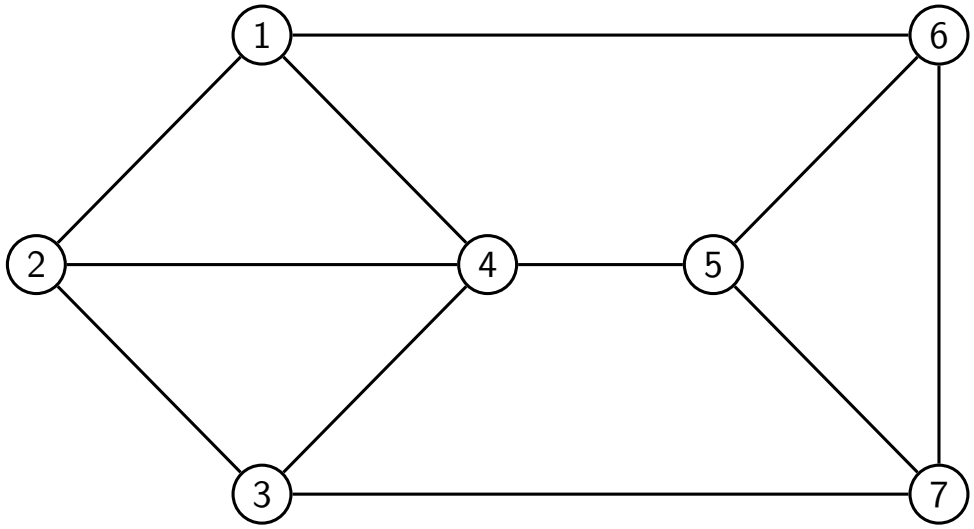


Its adjacency and degree matrices are given as

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

Example Graph

Normalized Adjacency Matrix



The normalized adjacency matrix is as follows:

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

The eigenvalues of $\boldsymbol{M}$ are: $\lambda_1 = 1$, $\lambda_2 = 0.483$, $\lambda_3 = 0.206$, $\lambda_4 = -0.045$, $\lambda_5 = -0.405$, $\lambda_6 = -0.539$, $\lambda_7 = -0.7$

Graph Laplacian Matrix

The *Laplacian matrix* of a graph is defined as

$$\boldsymbol{L} = \Delta - \boldsymbol{A}$$

$$= \begin{pmatrix} \sum_{j=1}^{n} a_{1j} & 0 & \cdots & 0 \\ 0 & \sum_{j=1}^{n} a_{2j} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sum_{j=1}^{n} a_{nj} \end{pmatrix} - \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \cdots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix}$$

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

$\boldsymbol{L}$ is a symmetric, positive semidefinite matrix. This means that $\boldsymbol{L}$ has n real, non-negative eigenvalues, which can be arranged in decreasing order as follows: $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n \geq 0$. Because $\boldsymbol{L}$ is symmetric, its eigenvectors are orthonormal. The rank of $\boldsymbol{L}$ is at most $n-1$, and the smallest eigenvalue is $\lambda_n = 0$.

Graph Laplacian Matrix

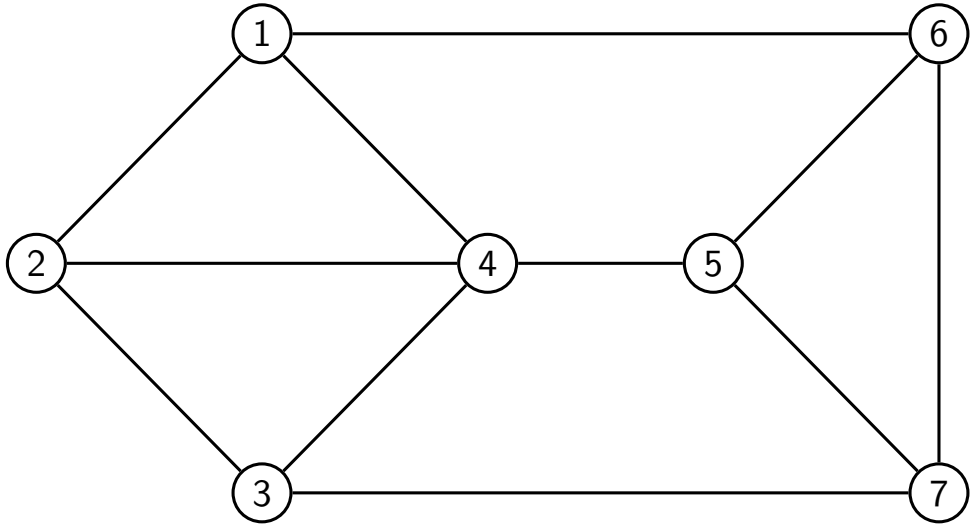
The Laplacian matrix is used to compute some properties of its corresponding graph. For example:

1. The number of connected components of graph G is equal to the multiplicity of the Laplacian eigenvalue 0.
2. The cofactor of any element of L(G) is equal to the number of spanning trees of G.

Laplacian matrix is usually used when the graph is irregular. Its main advantage is that it enables us to study certain properties of irregular graphs (unlike adjacent matrix).

Given a graph function, the Laplacian measures how much the function on a graph differs at a vertex from the average of the values of the same function over the neighbors of the vertex.

Example Graph: Laplacian Matrix



The graph Laplacian is given as

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

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

Normalized Laplacian Matrices

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

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

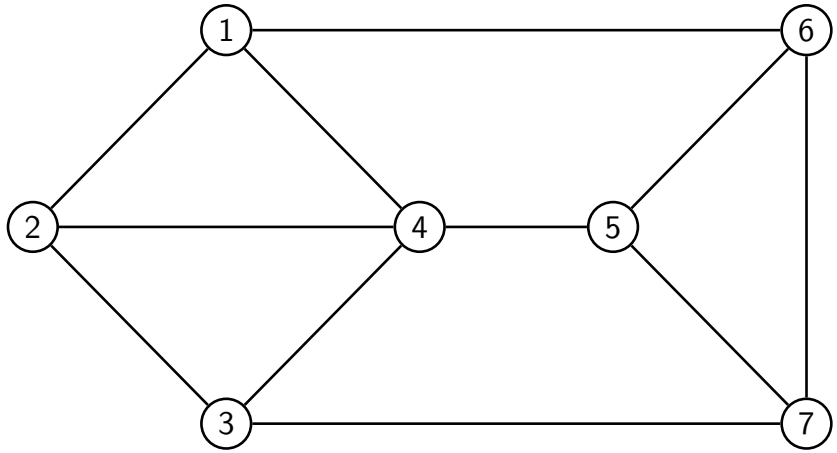
$\boldsymbol{L}^s$ is a symmetric, positive semidefinite matrix, with rank at most $n-1$. The smallest eigenvalue $\lambda_n = 0$. The *normalized asymmetric Laplacian* matrix is defined as

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

$\boldsymbol{L}^a$ is also a positive semi-definite matrix with n real eigenvalues $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n = 0$.

Example Graph

Normalized Symmetric Laplacian Matrix



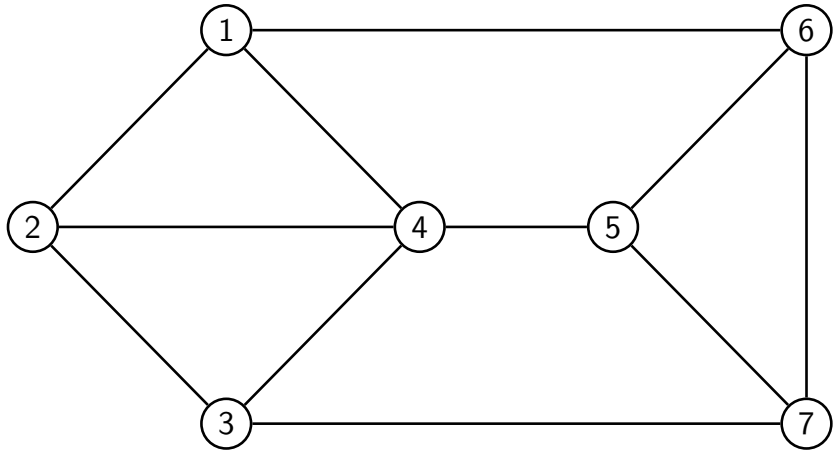
The normalized symmetric Laplacian is given as

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

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

Example Graph

Normalized Asymmetric Laplacian Matrix



The normalized asymmetric Laplacian matrix is given as

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

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

Clustering as Graph Cuts

A *k-way cut* in a graph is a partitioning or clustering of the vertex set, given as $\mathcal{C} = \{C_1, \ldots, C_k\}$. We require $\mathcal{C}$ to optimize some objective function that captures 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 $\boldsymbol{A}$, 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

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

Given $S \subseteq V$, we denote by $\overline{S}$ the complementary set of vertices, that is, $\overline{S} = V - S$. 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 $W(S, \overline{S})$.

Cuts and Matrix Operations

Given a clustering $\mathcal{C} = \{C_1, \ldots, C_k\}$ comprising k clusters. Let $\boldsymbol{c}_i \in \{0,1\}^n$ be the *cluster indicator vector* that records the cluster membership for cluster C_i, defined as

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

The cluster size can be written as

$$|C_i| = \boldsymbol{c}_i^T \boldsymbol{c}_i = \|\boldsymbol{c}_i\|^2$$

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:

$$vol(C_i) = W(C_i, V) = \sum_{v_r \in C_i} d_r = \sum_{v_r \in C_i} c_{ir} d_r c_{ir} = \sum_{r=1}^{n} \sum_{s=1}^{n} c_{ir} \Delta_{rs} c_{is} = \boldsymbol{c}_i^T \Delta \boldsymbol{c}_i$$

Cuts and Matrix Operations

The sum of weights of all internal edges is:

$$W(C_i, C_i) = \sum_{v_r \in C_i} \sum_{v_s \in C_i} a_{rs} = \sum_{r=1}^{n} \sum_{s=1}^{n} c_{ir} a_{rs} c_{is} = \boldsymbol{c}_i^T \boldsymbol{A} \boldsymbol{c}_i$$

We can get the sum of weights for all the external edges as follows:

$$W(C_i, \overline{C_i}) = \sum_{v_r \in C_i} \sum_{v_s \in V - C_i} a_{rs} = W(C_i, V) - W(C_i, C_i) = \boldsymbol{c}_i(\Delta - \boldsymbol{A})\boldsymbol{c}_i = \boldsymbol{c}_i^T \boldsymbol{L} \boldsymbol{c}_i$$

Clustering Objective Functions: Ratio Cut

The clustering objective function can be formulated as an optimization problem over the k-way cut $\mathcal{C} = \{C_1, \ldots, C_k\}$.
The *ratio cut* objective is defined over a k-way cut as follows:

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

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

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

The optimal solution comprises the eigenvectors corresponding to the k smallest eigenvalues of $\boldsymbol{L}$, i.e., the eigenvectors $\boldsymbol{u}_n, \boldsymbol{u}_{n-1}, \ldots, \boldsymbol{u}_{n-k+1}$ represent the relaxed cluster indicator vectors.

Clustering Objective Functions: Normalized Cut

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

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

We can obtain an optimal solution by allowing $\boldsymbol{c}_i$ to be an arbitrary real vector.

The optimal solution comprise the eigenvectors corresponding to the k smallest eigenvalues of either the normalized symmetric or asymmetric Laplacian matrices, $\boldsymbol{L}^s$ and $\boldsymbol{L}^a$.

Spectral Clustering Algorithm

The spectral clustering algorithm takes a dataset $\boldsymbol{D}$ as input and computes the similarity matrix $\boldsymbol{A}$. For normalized cut we chose either $\boldsymbol{L}^s$ or $\boldsymbol{L}^a$, whereas for ratio cut we choose $\boldsymbol{L}$. Next, we compute the k smallest eigenvalues and corresponding eigenvectors of the chosen matrix.

The main problem is that the eigenvectors $\boldsymbol{u}_i$ 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:

$$\boldsymbol{U} = \begin{pmatrix} | & | & & | \\ \boldsymbol{u}_n & \boldsymbol{u}_{n-1} & \cdots & \boldsymbol{u}_{n-k+1} \\ | & | & & | \end{pmatrix} \rightarrow \text{ normalize rows } \rightarrow \begin{pmatrix} — & \boldsymbol{y}_1^T & — \\ — & \boldsymbol{y}_2^T & — \\ & \vdots & \\ — & \boldsymbol{y}_n^T & — \end{pmatrix} = \boldsymbol{Y}$$

We then cluster the new points in $\boldsymbol{Y}$ into k clusters via the K-means algorithm or any other fast clustering method to obtain binary cluster indicator vectors $\boldsymbol{c}_i$.

Spectral Clustering Algorithm

Spectral Clustering $(\boldsymbol{D}, k)$**:**

1 Compute the similarity matrix $\boldsymbol{A} \in \mathbb{R}^{n \times n}$

2 **if** *ratio cut* **then** $\boldsymbol{B} \leftarrow \boldsymbol{L}$

3 **else if** *normalized cut* **then** $\boldsymbol{B} \leftarrow \boldsymbol{L}^s$ or $\boldsymbol{L}^a$

4 Solve $\boldsymbol{B}\boldsymbol{u}_i = \lambda_i \boldsymbol{u}_i$ for $i = n, \ldots, n-k+1$, where $\lambda_n \leq \lambda_{n-1} \leq \cdots \leq \lambda_{n-k+1}$

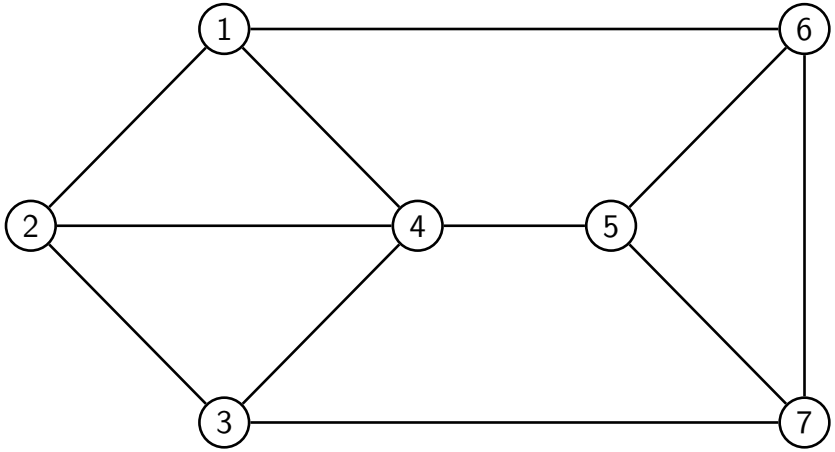
5 $\boldsymbol{U} \leftarrow \begin{pmatrix} \boldsymbol{u}_n & \boldsymbol{u}_{n-1} & \cdots & \boldsymbol{u}_{n-k+1} \end{pmatrix}$

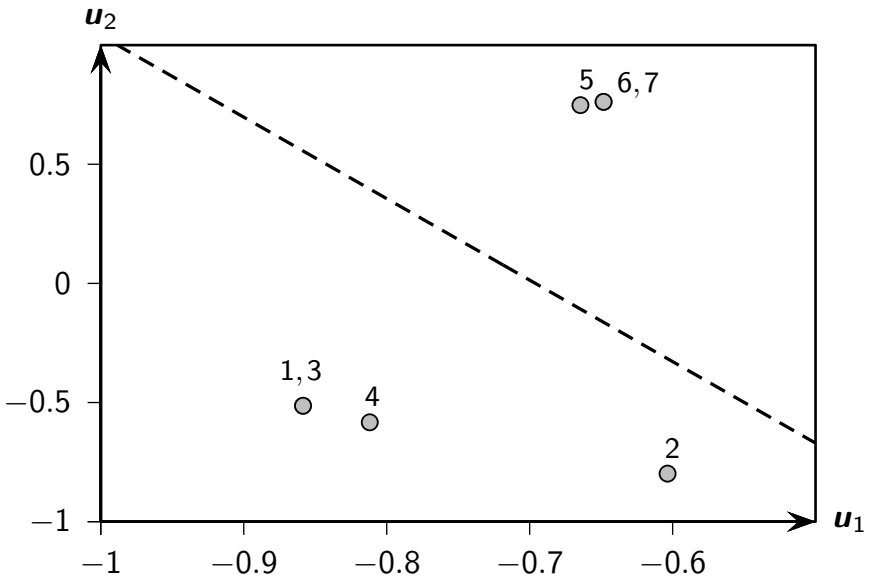
6 $\boldsymbol{Y} \leftarrow$ normalize rows of $\boldsymbol{U}$

7 $\mathcal{C} \leftarrow \{C_1, \ldots, C_k\}$ via K-means on $\boldsymbol{Y}$

Spectral Clustering on Example Graph

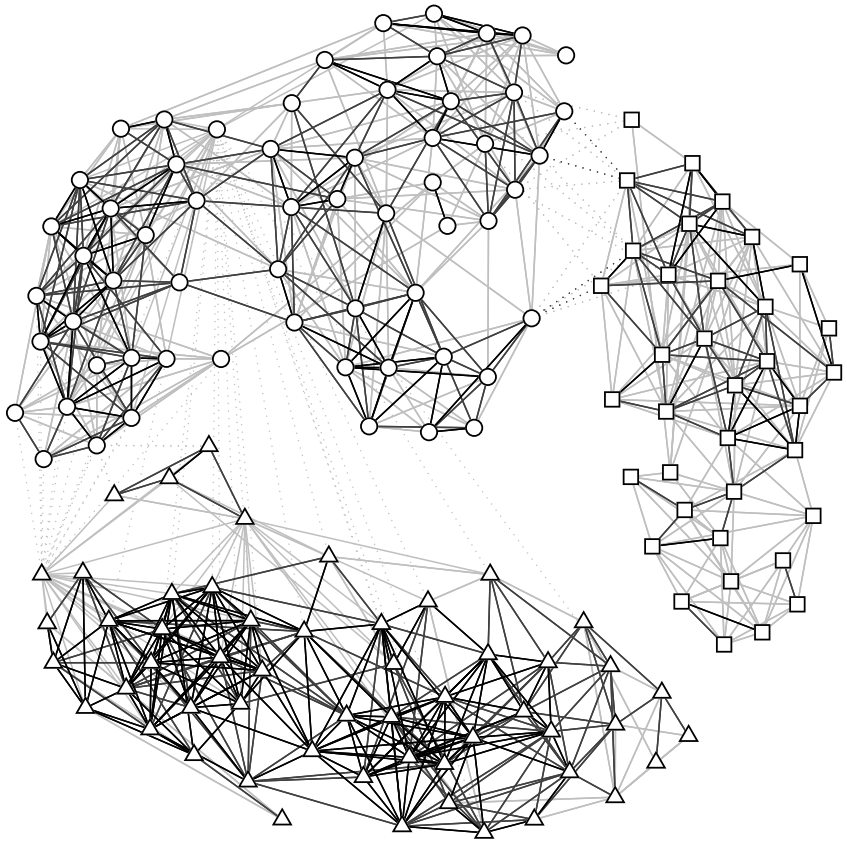
$k = 2$, normalized cut (normalized asymmetric Laplacian)





Normalized Cut on Iris Graph

$k = 3$, normalized asymmetric Laplacian



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

Maximization Objectives: Average Cut

The *average weight* objective is defined as

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

where $\boldsymbol{u}_i$ is an arbitrary real vector, which is a relaxation of the binary cluster indicator vectors $\boldsymbol{c}_i$.

We can maximize the objective by selecting the k largest eigenvalues of $\boldsymbol{A}$, and the corresponding eigenvectors.

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

where $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n$. In general, while $\boldsymbol{A}$ is symmetric, it may not be positive semidefinite. This means that $\boldsymbol{A}$ can have negative eigenvalues, and to maximize the objective we must consider only the positive eigenvalues and the corresponding eigenvectors.

Maximization Objectives: Modularity

Given $\boldsymbol{A}$, the weighted adjacency matrix, the modularity of a clustering is the difference between the observed and expected fraction of weights on edges within the clusters. The clustering objective is given as

$$\max_{\mathcal{C}} J_Q(\mathcal{C}) = \sum_{i=1}^{k} \left(\frac{\boldsymbol{c}_i^T \boldsymbol{A} \boldsymbol{c}_i}{tr(\Delta)} - \frac{(\boldsymbol{d}_i^T \boldsymbol{c}_i)^2}{tr(\Delta)^2} \right) = \sum_{i=1}^{k} \boldsymbol{c}_i^T \boldsymbol{Q} \boldsymbol{c}_i$$

where $\boldsymbol{Q}$ is the *modularity matrix*:

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

The optimal solution comprises the eigenvectors corresponding to the k largest eigenvalues of $\boldsymbol{Q}$. Since $\boldsymbol{Q}$ is symmetric, but not positive semidefinite, we use only the positive eigenvalues.

Markov Chain Clustering

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 timesteps $t = 1, 2, \ldots$, 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}, that is,

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

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

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

is independent of the time step t.

Markov Chain Clustering: Markov Matrix

The normalized adjacency matrix $\boldsymbol{M} = \Delta^{-1}\boldsymbol{A}$ can be interpreted as the $n \times n$ *transition matrix* where the entry $m_{ij} = \frac{a_{ij}}{d_i}$ is the probability of transitioning or jumping from node i to node j in the graph G.

The matrix $\boldsymbol{M}$ is thus the transition matrix for a *Markov chain* or a Markov random walk on graph G. That is, given node i the transition matrix $\boldsymbol{M}$ specifies the probabilities of reaching any other node j in one time step.

In general, the transition probability matrix for t time steps is given as

$$\boldsymbol{M}^{t-1} \cdot \boldsymbol{M} = \boldsymbol{M}^t$$

Markov Chain Clustering: Random Walk

A random walk on G thus corresponds to taking successive powers of the transition matrix $\boldsymbol{M}$.

Let $\boldsymbol{\pi}_0$ specify the initial state probability vector at time $t = 0$. The state probability vector after t steps is

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

Equivalently, taking transpose on both sides, we get

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

The state probability vector thus converges to the dominant eigenvector of $\boldsymbol{M}^T$.

Markov Clustering Algorithm

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 $\boldsymbol{M}$, define the inflation operator Υ as follows:

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

The net effect of the inflation operator is to increase the higher probability transitions and decrease the lower probability transitions.

The Markov clustering algorithm (MCL) 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.

MCL takes as input the inflation parameter $r \geq 1$. Higher values lead to more, smaller clusters, whereas smaller values lead to fewer, but larger clusters.

Markov Clustering Algorithm: MCL

The final clusters are found by enumerating the weakly connected components in the directed graph induced by the converged transition matrix $\boldsymbol{M}_t$, where the edges are defined as:

$$E = \{(i,j) \mid \boldsymbol{M}_t(i,j) > 0\}$$

A directed edge (i,j) exists only if node i can transition to node j within t steps of the expansion and inflation process.

A node j is called an *attractor* if $\boldsymbol{M}_t(j,j) > 0$, and we say that node i is attracted to attractor j if $\boldsymbol{M}_t(i,j) > 0$. 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.

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_j, MCL finds the weakly connected components consisting of all nodes $i \in V_t - V_a$ attracted to an attractor in S_j. If a node i is attracted to multiple strongly connected components, it is added to each such cluster, resulting in possibly overlapping clusters.

Algorithm Markov Clustering

Markov Clustering $(\boldsymbol{A}, r, \epsilon)$**:**

$t \leftarrow 0$

Add self-edges to $\boldsymbol{A}$ if they do not exist

$\boldsymbol{M}_t \leftarrow \Delta^{-1}\boldsymbol{A}$

repeat

  $t \leftarrow t+1$

  $\boldsymbol{M}_t \leftarrow \boldsymbol{M}_{t-1} \cdot \boldsymbol{M}_{t-1}$

  $\boldsymbol{M}_t \leftarrow \Upsilon(\boldsymbol{M}_t, r)$

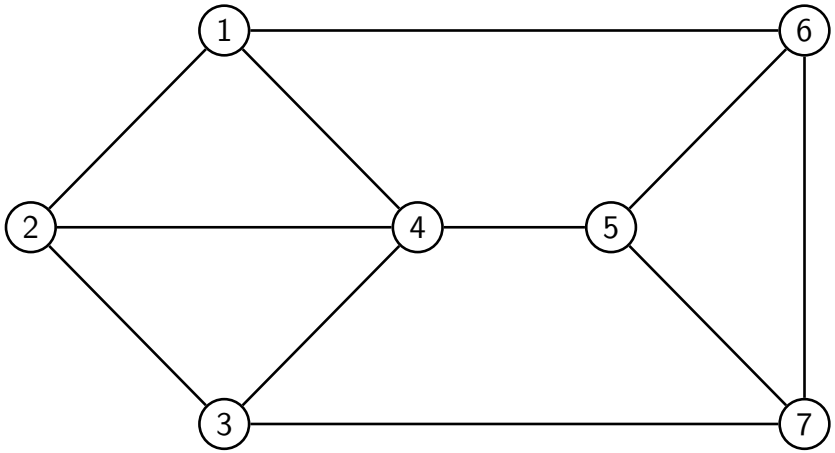
until $\|\boldsymbol{M}_t - \boldsymbol{M}_{t-1}\|_F \leq \epsilon$

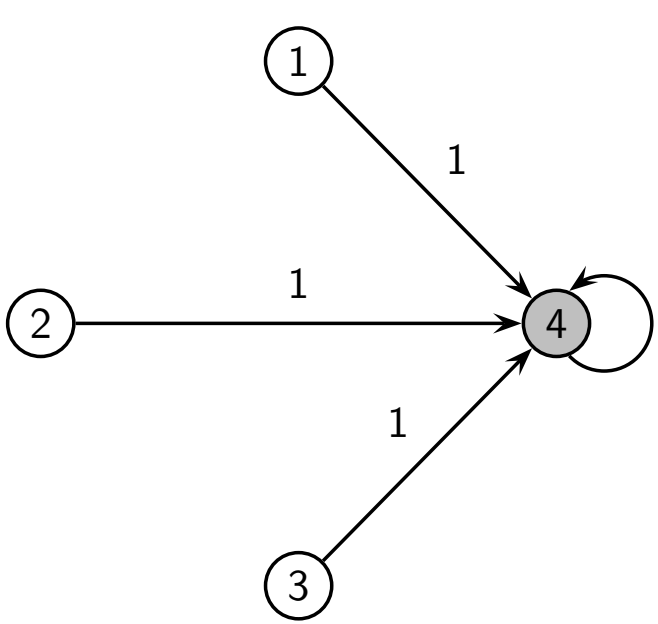
$G_t \leftarrow$ directed graph induced by $\boldsymbol{M}_t$

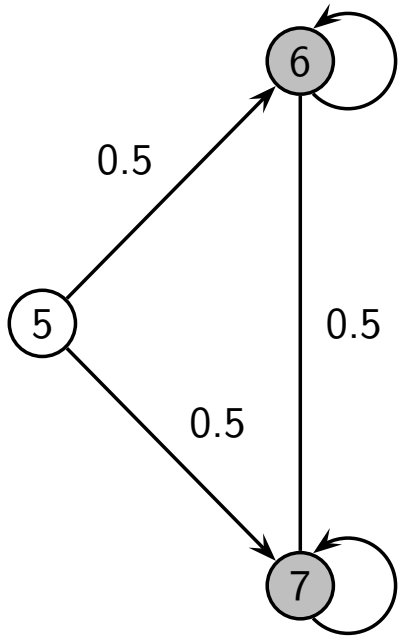
$\mathcal{C} \leftarrow \{$weakly connected components in $G_t\}$

MCL Attractors and Clusters

$r = 2.5$

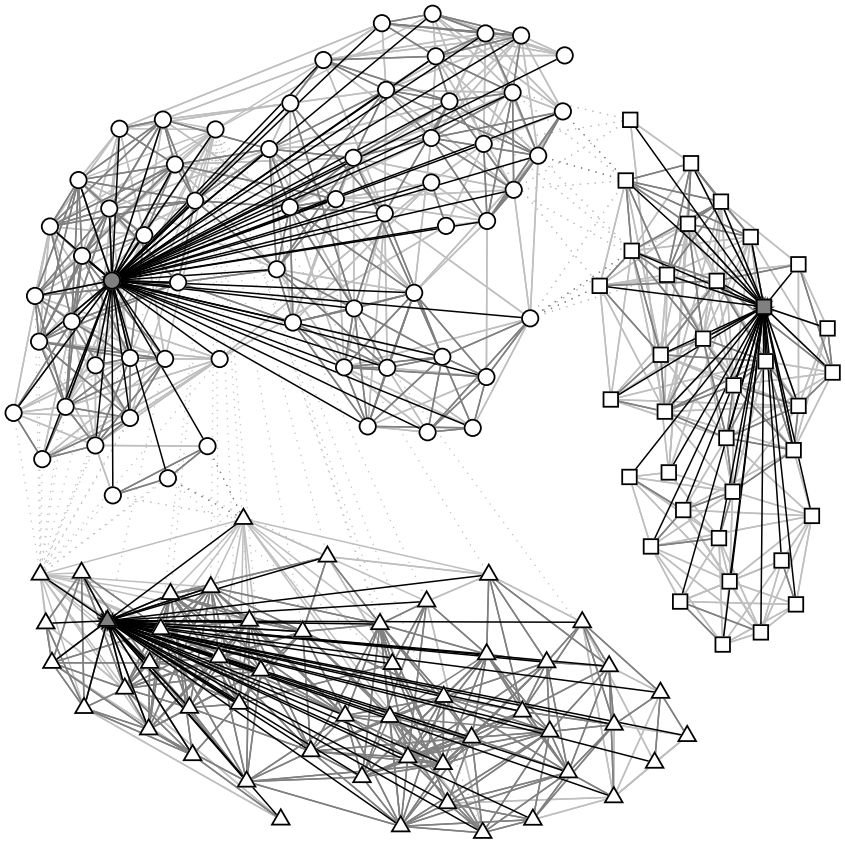




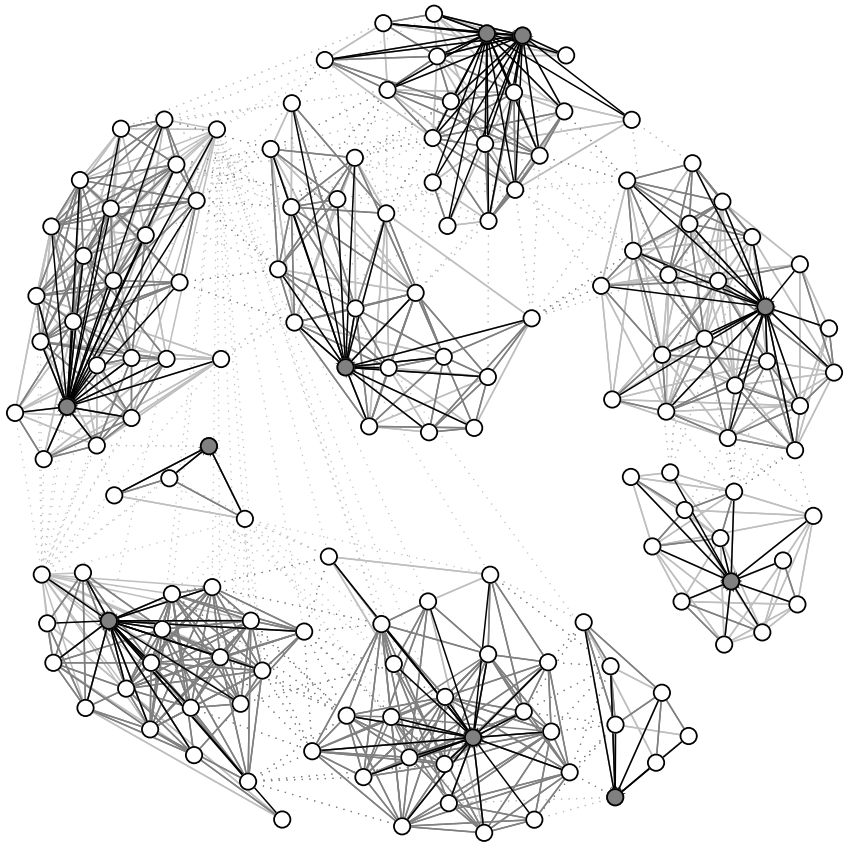


MCL on Iris Graph

Comparison between two values of r



(a) $r = 1.3$



(b) $r = 2$

Contingency Table: MCL Clusters versus Iris Types

$r = 1.3$

There is a quite significant number of `iris-virginica` misplaced.

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

Data Mining and Machine Learning: Fundamental Concepts and Algorithms

dataminingbook.info

Mohammed J. Zaki[1] Wagner Meira Jr.[2]

[1]Department of Computer Science
Rensselaer Polytechnic Institute, Troy, NY, USA

[2]Department of Computer Science
Universidade Federal de Minas Gerais, Belo Horizonte, Brazil

Chapter 16: Spectral & Graph Clustering