Statistical Data Analysis

Dr. Jana de Wiljes

14. Dezember 2022

Universität Potsdam

Relaxation of RatioCut for $k > 2$ (sketch)

- Define k indicator vectors $h_j = (h_{1,j}, h_{2,j}, \ldots, h_{n,j})^T$ via

$$h_{i,j} = \begin{cases} 1/\sqrt{|A_j|} & \text{if } v_i \in A_j, \\ 0 & \text{else} \end{cases}$$

- Matrix $H \in \mathbb{R}^{n \times k}$ with columns $h_1, h_2, \ldots, h_k$ is orthogonal
- We have $h_i^T L h_i = \frac{\text{cut}(A_i, \overline{A}_i)}{|A_i|}$ and $h_i^T L h_i = (H^T L H)_{ii}$
- Together this yields $\text{RatioCut}(A_1, A_2, \ldots, A_k) = \text{tr}(H^T L H)$
- Hence the minimum of $\text{RatioCut}(A_1, A_2, \ldots, A_k)$ can be approximated by solving (the relaxed version)

$$\min_{H \in \mathbb{R}^{n \times k}} \text{tr}(H^T L H) \text{ subject to } H^T H = I$$

General case of unnormalized spectral clustering

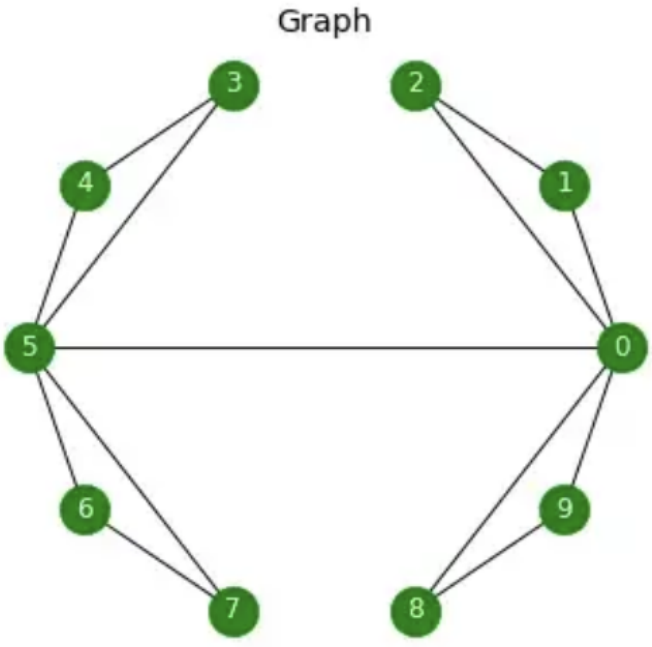
Algorithm: Unnormalized spectral clustering

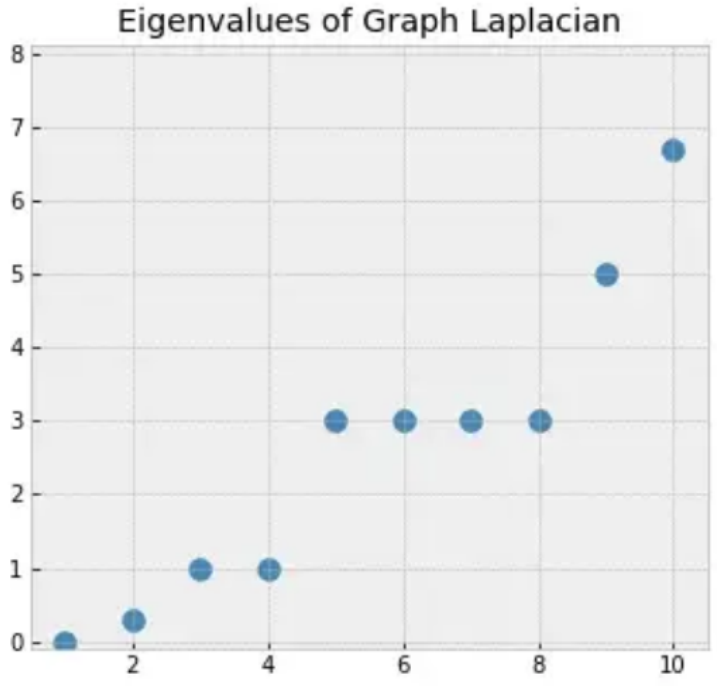
Input: Weight matrix (or any similarity matrix) $S \in \mathbb{R}^{n \times n}$, number k of clusters

1. Construct similarity graph G;
2. Compute $L(G)$;
3. Compute Eigenvectors $X_1, X_2, \ldots, X_k$ of $L(G)$;
4. Build $U \in \mathbb{R}^{n \times k}$ with $X_1, X_2, \ldots, X_k$ as columns;
5. Rows of U are $y_1, y_2, \ldots, y_n \in \mathbb{R}^k$;
6. Cluster $y_1, y_2, \ldots, y_n$ into Clusters $C_1, C_2, \ldots, C_k$ (k-means);

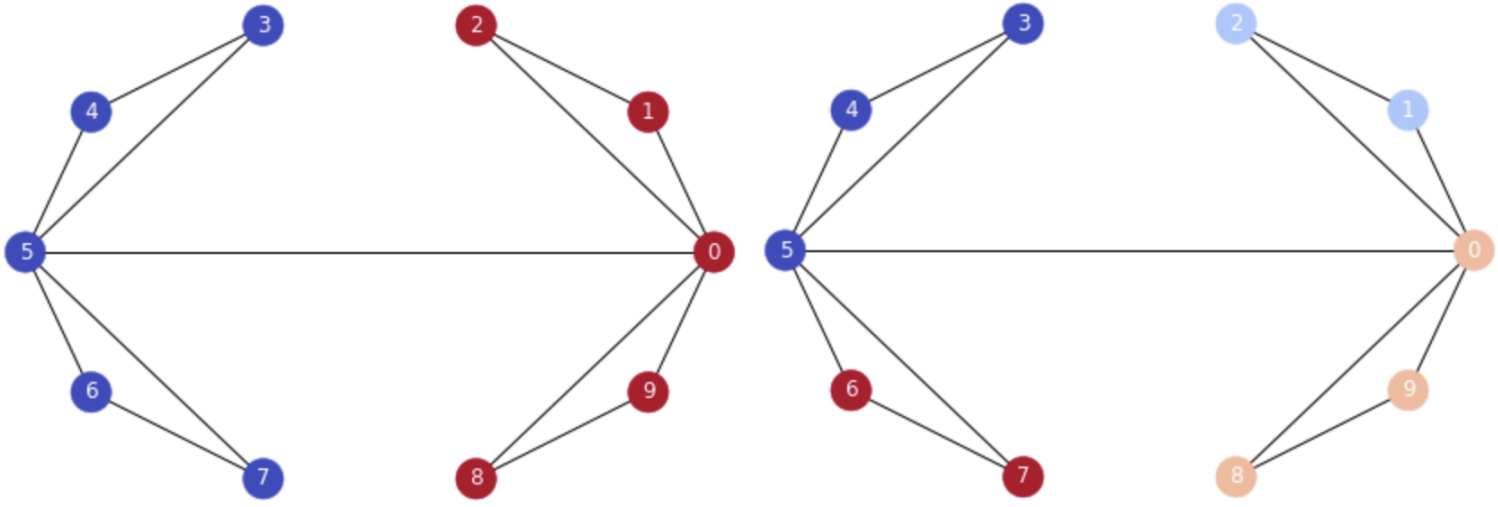
Output: Clusters $A_1, A_2, \ldots, A_k$ with $A_i = \{j : y_j \in C_i\}$

Relationship eigenvalues and graph structure



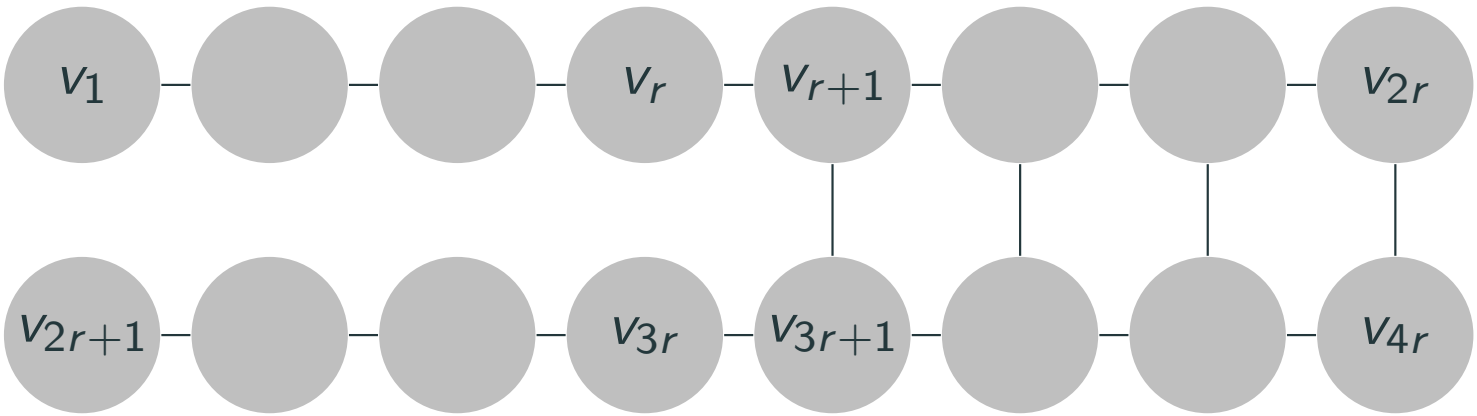


Results different number of regimes



Notes on spectral clustering

- faster ($\mathcal{O}(mn)$ or $\mathcal{O}(n^2)$ for the eigenvector calculation) than Kernighan-Lin
- quality of approximated solution can be arbitrarily far from exact solution (cockroach graphs for $k = 2$)



- other relaxations possible (and probably useful)
- any other clustering algorithm may be used instead of k-means

Normalised Version

Problem: In a weighted graph, a vertex may have a large degree because of a small number of connected edges but with large weights just as well as due to a large number of connected edges with unit weights.

Instead of using L one uses:

- symmetric normalized Laplacian

$$L^{\text{sym}} = I - D^{-1/2} A D^{-1/2} \tag{1}$$

- random walk normalized Laplacian

$$L^{\text{rw}} := D^{-1} L = I - D^{-1} A \tag{2}$$

Idea

Similarity can be determined in various ways.

Independent of graphs:

- dating services (common interests, backgrounds, age,. . .)
- movie recommendation (genre, actors, . . .)
- . . .
- typical measures: Euclidian distance, Gaussian, cosine. . .

Dependent on graph structure:

- *structural equivalence* (share many of the same neighbours)
- *regular equivalence* (have neighbours which are themselves similar)

We will only consider undirected graphs.

Common neighbours

Count the number of common neighbours:

$$cn_{ij} = \sum_k a_{ik} a_{jk}$$

Problems and remarks:

- related to cocitation measure
- What is a large value?
- need some sort of normalization
  - divide by total number of vertices in graph (penalizes vertices with low degree)
  - solution: take degree into account (such a measure is called *cosine similarity*)

Similarity and Dissimilarity Metrics

Minkowski distance:

$$D(x, y) = \left(\sum_{i=1}^{k} |x_i - y_i|^p \right)^{\frac{1}{p}}$$

Gaussian kernel function:

$$K(x, y) = \exp\left(-\frac{\|x - y\|^2}{2\sigma^2} \right) = \exp\left(-\frac{D(x, y)}{2\sigma^2} \right),$$

Cosine similarity

Similar to geometric cosine similarity:

$$\sigma_{ij} = \frac{\sum_k a_{ik} a_{jk}}{\sqrt{\sum_k a_{ik}^2} \sqrt{\sum_k a_{jk}^2}}$$

Graph unweighted and simple (no loops or multi-edges) then

$$\sum_k a_{ik}^2 = \sum_k a_{ik} = d_i$$

Hence

$$\sigma_{ij} = \frac{cn_{ij}}{\sqrt{d_i d_j}}$$

Note: we set $\sigma_{ij} = 0$ if one of i, j has degree 0.

Pearson coefficients

Alternative: normalize via expected values of the number of common neighbours.

- choose neighbours of i and j at random
- probability for choosing a neighbour of j which i chose is $\frac{d_i}{n}$ (per neighbour)
- $\rightsquigarrow$ expected number of common neighbours is $\frac{d_i d_j}{n}$ (ignore possibility of choosing same neighbour twice)

Pearson coefficient

Hence a reasonable measure of similarity would be

$$\begin{aligned}\sum_k a_{ik}a_{jk} - \frac{d_i d_j}{n} &= \sum_k a_{ik}a_{jk} - \frac{1}{n}\sum_k a_{ik}\sum_l a_{jl} \\ &= \sum_k a_{ik}a_{jk} - n\langle a_i\rangle\langle a_j\rangle \\ &= \sum_k (a_{ik} - \langle a_i\rangle)(a_{jk} - \langle a_j\rangle),\end{aligned}$$

where $\langle a_i\rangle = \frac{1}{n}\sum_k a_{ik}$.

Normalizing this (which is just the covariance $\mathrm{cov}(a_i, a_j)$ of the rows a_i and a_j) gives us the *standard Pearson correlation coefficient*:

$$r_{ij} = \frac{\mathrm{cov}(a_i, a_j)}{\sigma_i \sigma_j} = \frac{\sum_k (a_{ik} - \langle a_i\rangle)(a_{jk} - \langle a_j\rangle)}{\sqrt{\sum_k (a_{ik} - \langle a_i\rangle)^2}\sqrt{\sum_k (a_{jk} - \langle a_j\rangle)^2}}$$

Regular equivalence

Define similarity score ρ_{ij} such that i and j have high similarity if they have neighbours k and l with high similarity:

$$\rho_{ij} = \alpha \sum_{k,l} a_{ki} a_{lj} \rho_{kl}$$

which in matrix notation reads as

$$\rho = \alpha \mathbf{A} \rho \mathbf{A}$$

This is an eigenvector equation!

Problems:

- self-similarity is not necessarily high
- similarity of vertices with many common neighbours is not necessarily high

Regular equivalence

Fix this by introducing extra diagonal term:

$$\rho_{ij} = \alpha \sum_{k,l} a_{ki} a_{lj} \rho_{kl} + \delta_{ij}$$

which in matrix notation reads as

$$\rho = \alpha \mathbf{A} \rho \mathbf{A} + \mathbf{I}$$

Still problems:

- iterating this process with for example $\rho^{(0)} = \mathbf{0}$ will give us sums in the form of $\alpha^2 \mathbf{A}^4 + \alpha \mathbf{A}^2 + \mathbf{I}$, therefore similarity is weighted sum of number of paths of even lengths
- Why only paths of even length?

Regular equivalence

Better measure (i is similar to j if i has neighbour k which is similar to j):

$$\rho_{ij} = \alpha \sum_k a_{ki} \rho_{kj} + \delta_{ij}$$

which in matrix notation reads as

$$\rho = \alpha \mathbf{A} \rho + \mathbf{I}$$

Starting iteration with $\rho = \mathbf{0}$ results in

$$\rho = \sum_{h=0}^{\infty} (\alpha \mathbf{A})^h = (\mathbf{I} - \alpha \mathbf{A})^{-1}.$$

This basically counting paths of length h between i and j weighted by α^h (longer ones will have smaller weights if $\alpha < 1$).

Variations of regular equivalence

One can remove the bias towards high-degree vertices in the following way:

$$\frac{\alpha}{d_i} \sum_k a_{ki} \rho_{kl} + \delta_{ij}$$

which reads in matrix notation as

$$(\mathbf{I} - \alpha \mathbf{D}^{-1} \mathbf{A})^{-1} = (\mathbf{D} - \alpha \mathbf{A})^{-1} \mathbf{D},$$

where $\mathbf{D}$ is the degree matrix.

Different generalisation is to include more elements (except only diagonal terms given by δ_{ij})$\rightsquigarrow$ include other aspects of similarity between certain vertices (given by other features); also useful if vertices in different connected components are similar

Computing eigenvalues

- find roots of characteristic polynomial (computationally expensive)
- power method: iterate (with any starting vector $X_{(0)}$)

$$X_{(l)} = A^l X_{(0)}$$

- power method converges to eigenvector X corresponding to (w.r.t. absolute value) largest eigenvalue
- power method is fast but
  - method does not work if $X_{(0)}$ is orthogonal to X (can be avoided by choosing all entries of $X_{(0)}$ to be positive, since X has only entries of same sign)
  - entries of $X_{(l)}$ become large during the iterative process (renormalization helps)
  - When are we done? (option is to start with two different vectors)

Computational complexity of the power method

Two aspects:

- complexity of one multiplication
- required number of multiplications

First aspect:

- n^2 multiplications if stored in adjacency matrix
- less if matrix is stored in adjacency lists and matrix is sparse
  - compute
    $$\sum_{j \in N(i)} X_{(l)_j}$$
    which gives the i-th entry of $X_{(l+1)}$
  - therefore a total of
    $$\sum_i d(i) = 2m$$
    operations (m being the number of edges in the graph)

Computational complexity of the power method

Second aspect:

- it can be shown that this is $\mathcal{O}(n)$
- we will come back to this

Total computational complexity is $\mathcal{O}(mn)$, which means

- $\mathcal{O}(n^2)$ if graph is sparse
- $\mathcal{O}(n^3)$ if graph is dense

$\rightsquigarrow$ use adjacency list

Computing other eigenvalues

We have

$$(\lambda_n I - L)X_i = (\lambda_n - \lambda_i)X_i$$

hence eigenvalues are reversed for matrix $\lambda_n I - L$

$\rightsquigarrow$ smallest eigenvalue can be calculated with power method

Computing other eigenvalues

Trick to compute second largest eigenvalue:

- X_n normalised eigenvector corresponding to largest eigenvalue λ_n
- choose starting vector X and define

$$Y = X - (X_n^T X) X_n$$

- we have

$$X_i^T Y = \begin{cases} 0 & \text{if } i = n, \\ X_i^T X & \text{otherwise} \end{cases}$$

- therefore

$$Y = \sum_{i=1}^{n-1} c_i X_i$$

  where $c_i = X_i^T Y$

$\rightsquigarrow$ use Y as starting vector for power method

Compute all eigenvalues and eigenvectors

Combining methods for given (symmetric) matrix A with eigenvectors X_i and eigenvalues λ_i:

- Find orthogonal matrix Q with $B = Q^T A Q$ being a tridiagonal matrix
  - $Q^T X_i$ is eigenvector of B
  - B can be found efficiently, e.g. by Householder algorithm or Lancosz algorithm
- compute eigenvalues and eigenvectors of B
  - these give eigenvalues and eigenvectors of A
  - can be done using for example the QL algorithm