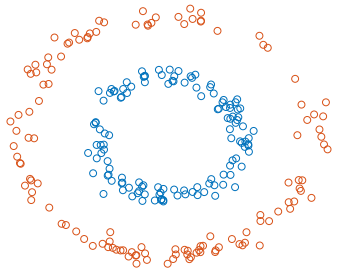
Clustering so far…

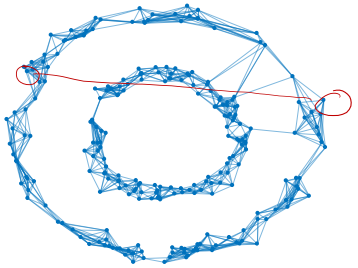
Clustering methods such as K-means and GMMs use a spherical or elliptical metric to form clusters

Does not work well when clusters are non-convex



Recall the approach of ISOMAP

- Conceptualize data as a ***graph***



- Redefine distance to how long it takes to "walk" from one point in the graph to another
- Means that other data points that have been observed play an important role in computing the distance

Spectral clustering

Given the data $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^d$, we begin by constructing a graph where:

- the $\mathbf{x}_i$ are the vertices of the graph
- the edges and their weights $w_{i,j}$ are determined by how close together $\mathbf{x}_i$ and $\mathbf{x}_j$ are in $\mathbb{R}^d$
- the weights are collected into an $n \times n$ matrix $\mathbf{W}$

The weights $w_{i,j}$ with $w_{i,j} > 0$ correspond to edges in the graph, in which case we say $\mathbf{x}_i$ and $\mathbf{x}_j$ are ***adjacent***

Note: we are considering an ***undirected graph***, in which case we assume that $\mathbf{W}$ is ***symmetric***

Similarity graphs

Some common choices for the graph structure of $\mathbf{W}$ include

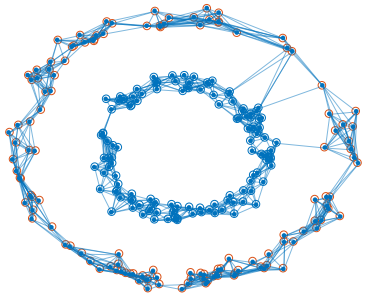
Adjacency structure

- k-nearest neighbors: connect $\mathbf{x}_i$ to its k-nearest neighbors
- ϵ-ball graph: connect $\mathbf{x}_i$ to any $\mathbf{x}_j$ within a radius of ϵ
- complete graph: connect $\mathbf{x}_i$ to every other $\mathbf{x}_j$

Edge weights

- constant: $$w_{i,j} = \begin{cases} 1 & \text{if } \mathbf{x}_i, \mathbf{x}_j \text{ connected} \\ 0 & \text{otherwise} \end{cases}$$
- Gaussian: $$w_{i,j} = \begin{cases} e^{-\gamma \|\mathbf{x}_i - \mathbf{x}_j\|_2^2} & \text{if } \mathbf{x}_i, \mathbf{x}_j \text{ connected} \\ 0 & \text{otherwise} \end{cases}$$

Example



We can see from the graph that it can naturally be separated into two components

This structure can be detected automatically by examining the ***graph Lapalacian matrix***

Graph Laplacian

To define the graph Laplacian matrix, we first need

- the ***weighted degree*** of a node $\mathbf{x}_i$is given by

$$d_i = \sum_{j=1}^{n} w_{i,j}$$

- the ***degree matrix*** is the $n \times n$matrix given by

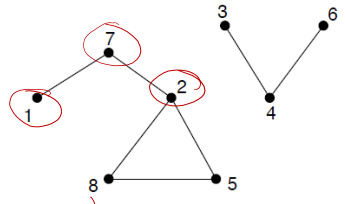
$$\mathbf{D} = \begin{bmatrix} d_1 & & \\ & \ddots & \\ & & d_n \end{bmatrix}$$

With these in hand, we can define the ***graph Laplacian***

$$\mathbf{L} = \mathbf{D} - \mathbf{W}$$

Example

Consider the graph below, where each edge has unit weight



$$\mathbf{W} = \begin{bmatrix} 0&0&0&0&0&0&1&0 \\ 0&0&0&0&1&0&1&1 \\ 0&0&0&1&0&0&0&0 \\ 0&0&1&0&0&1&0&0 \\ 0&1&0&0&0&0&0&1 \\ 0&0&0&1&0&0&0&0 \\ 1&1&0&0&0&0&0&0 \\ 0&1&0&0&1&0&0&0 \end{bmatrix} \quad \mathbf{D} = \begin{bmatrix} 1 & & & & & & & \\ & 3 & & & & & & \\ & & 1 & & & & & \\ & & & 2 & & & & \\ & & & & 2 & & & \\ & & & & & 1 & & \\ & & & & & & 2 & \\ & & & & & & & 2 \end{bmatrix}$$

Example

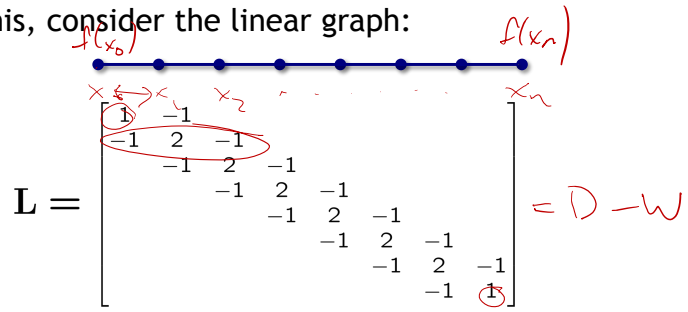
$$\mathbf{W} = \begin{bmatrix} 0&0&0&0&0&0&1&0 \\ 0&0&0&0&1&0&1&1 \\ 0&0&0&1&0&0&0&0 \\ 0&0&1&0&0&1&0&0 \\ 0&1&0&0&0&0&0&1 \\ 0&0&0&1&0&0&0&0 \\ 1&1&0&0&0&0&0&0 \\ 0&1&0&0&1&0&0&0 \end{bmatrix} \quad \mathbf{D} = \begin{bmatrix} 1 & & & & & & & \\ & 3 & & & & & & \\ & & 1 & & & & & \\ & & & 2 & & & & \\ & & & & 2 & & & \\ & & & & & 1 & & \\ & & & & & & 2 & \\ & & & & & & & 2 \end{bmatrix}$$

$$\mathbf{L} = \begin{bmatrix} 1&0&0&0&0&0&-1&0 \\ 0&3&0&0&-1&0&-1&-1 \\ 0&0&1&-1&0&0&0&0 \\ 0&0&-1&2&0&1&0&0 \\ 0&-1&0&0&2&0&0&-1 \\ 0&0&0&-1&0&1&0&0 \\ -1&-1&0&0&0&0&2&0 \\ 0&-1&0&0&-1&0&0&2 \end{bmatrix}$$

Graph Laplacian intuition

If $\mathbf{f} \in \mathbb{R}^n$ contains values of a function (whose domain is the vertices of the graph), applying the graph Laplacian to $\mathbf{f}$ gives you something like the second derivative

To see this, consider the linear graph:



$$\mathbf{L} = \begin{bmatrix} 1 & -1 & & & & & & \\ -1 & 2 & -1 & & & & & \\ & -1 & 2 & -1 & & & & \\ & & -1 & 2 & -1 & & & \\ & & & -1 & 2 & -1 & & \\ & & & & -1 & 2 & -1 & \\ & & & & & -1 & 2 & -1 \\ & & & & & & -1 & 1 \end{bmatrix}$$

Up to a scaling factor, this is precisely the finite difference approximation of the second derivative

Properties of $\mathbf{L}$

1. For any $\mathbf{f} \in \mathbb{R}^n$

$$\mathbf{f}^T\mathbf{L}\mathbf{f} = \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} w_{i,j}(f(i) - f(j))^2$$

2. $\mathbf{L}$ is positive semidefinite.
3. $\mathbf{L}$ satisfies $\mathbf{L}\mathbf{1} = \mathbf{0}$. Thus $\mathbf{1} \in \text{Null}(\mathbf{L})$, or equivalently, is an eigenvector of $\mathbf{L}$ with eigenvalue 0.

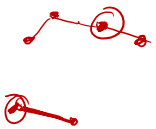
It turns out that the null space of $\mathbf{L}$ is key to recognizing how many connected components the graph has

In fact, the dimension of the null space of $\mathbf{L}$ is exactly the number of connected components

Connected components

Let $\mathcal{A} \subset \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ be a subset of the nodes in the graph and let $\mathbf{1}_\mathcal{A}$ be the indicator vector:

$$\mathbf{1}_\mathcal{A}(j) = \begin{cases} 1, & \mathbf{x}_j \in \mathcal{A} \\ 0, & \mathbf{x}_j \notin \mathcal{A} \end{cases}$$



We say that a subset of nodes $\mathcal{A}$ is ***connected*** if there is a path between every pair of nodes in $\mathcal{A}$

Fact. If a graph has k connected components $\mathcal{A}_1, \ldots, \mathcal{A}_k$, then

1. $\dim(\text{Null}(\mathbf{L})) = k$, and
2. $\text{Span}(\{\mathbf{1}_{\mathcal{A}_1}, \ldots, \mathbf{1}_{\mathcal{A}_k}\}) = \text{Null}(\mathbf{L})$

Fully connected case

In other words, the indicator functions of the connected components (which are clearly linearly independent) form a basis for the null space of the graph Laplacian

For $k = 1$ we have $\mathcal{A}_1 \subset \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$

We already know that $\mathbf{1}_\mathcal{A} = \mathbf{1} \in \text{Null}(\mathbf{L})$

All that remains is to show that $\mathbf{f} \in \text{Null}(\mathbf{L}) \Rightarrow \mathbf{f} = \alpha\mathbf{1}$

Suppose that $\mathbf{f} \in \text{Null}(\mathbf{L}) \Rightarrow \mathbf{L}\mathbf{f} = \mathbf{0} \Rightarrow \mathbf{f}^T\mathbf{L}\mathbf{f} = 0$

$$\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} w_{i,j}(f(i) - f(j))^2 = 0$$

This can only be zero if $f(i) - f(j) = 0$ for every $\mathbf{x}_i$ and $\mathbf{x}_j$ which are connected by an edge

Since the graph is connected, this implies that $\mathbf{f} = \alpha\mathbf{1}$

General case

For $k > 1$, we can order the nodes so that the weight matrix $\mathbf{W}$ and graph Laplacian $\mathbf{L}$ are ***block diagonal***

$$\mathbf{W} = \begin{bmatrix} \mathbf{W}_1 & & 0 \\ & \ddots & \\ 0 & & \mathbf{W}_k \end{bmatrix} \quad \mathbf{L} = \begin{bmatrix} \mathbf{L}_1 & & \\ & \ddots & \\ & & \mathbf{L}_k \end{bmatrix}$$

We can think of each component as its own fully connected subgraph, with $\mathbf{L}_j$ as the Laplacian for this subgraph and

1. $\mathbf{L}\mathbf{1}_{\mathcal{A}_j} = \mathbf{0}$ for $j = 1, \ldots, k$, and
2. If $\mathbf{L}\mathbf{f} = \mathbf{0}$, then we can write $\mathbf{f} = \sum_{j=1}^{k} \alpha_j \mathbf{1}_{\mathcal{A}_j}$, and so $\{\mathbf{1}_{\mathcal{A}_1}, \ldots, \mathbf{1}_{\mathcal{A}_k}\}$ is a basis for $\mathsf{Null}(\mathbf{L})$

Arbitrary null space basis

Let $\{\mathbf{u}_1, \ldots, \mathbf{u}_k\}$ be a basis for the null space of $\mathbf{L}$

Let $\mathbf{y}_i \in \mathbb{R}^k$ be defined as

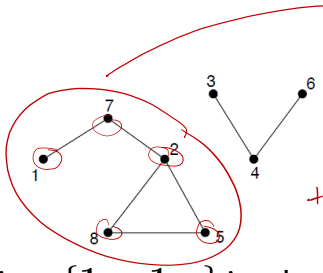
$$\mathbf{y}_i = \begin{bmatrix} u_1(i) \\ \vdots \\ u_k(i) \end{bmatrix} \qquad i = 1, \ldots, n$$

i.e., $\mathbf{y}_i^T$ is the i^{th} row of $\mathbf{U} = [\mathbf{u}_1 \cdots \mathbf{u}_k]$

Then

$$\mathbf{y}_i = \mathbf{y}_j \quad \Leftrightarrow \quad \mathbf{x}_i \text{ is connected to } \mathbf{x}_j$$

Example



$$\alpha_1 \mathbf{1}_{\mathcal{A}_1} = [1\ 1\ 0\ 0\ 1\ 0\ 1\ 1]^T$$

$$+\ \beta_1 \mathbf{1}_{\mathcal{A}_2} = [0\ 0\ 1\ 1\ 0\ 1\ 0\ 0]^T$$

Since $\{\mathbf{1}_{\mathcal{A}_1}, \mathbf{1}_{\mathcal{A}_2}\}$ is a basis for $\mathsf{Null}(\mathbf{L})$, we have that for any other basis $\{\mathbf{u}_1, \mathbf{u}_2\}$, there must exist $\alpha_1, \alpha_2, \beta_1, \beta_2 \in \mathbb{R}$ such that

$$\mathbf{u}_1 = \alpha_1 \mathbf{1}_{\mathcal{A}_1} + \beta_1 \mathbf{1}_{\mathcal{A}_2}$$
$$\mathbf{u}_2 = \alpha_2 \mathbf{1}_{\mathcal{A}_1} + \beta_2 \mathbf{1}_{\mathcal{A}_2}$$

$$\mathbf{U} = \begin{bmatrix} \alpha_1 & \alpha_1 & \beta_1 & \beta_1 & \alpha_1 & \beta_1 & \alpha_1 & \alpha_1 \\ \alpha_2 & \alpha_2 & \beta_2 & \beta_2 & \alpha_2 & \beta_2 & \alpha_2 & \alpha_2 \end{bmatrix}^T$$

Implications for clustering

Of course, in a typical setting, the graph we build from the data will not be ***perfectly*** separated into distinct components

More typically, we can think of the graph Laplacian as

$$\mathbf{L} = \begin{bmatrix} \mathbf{L}_1 & & \\ & \ddots & \\ & & \mathbf{L}_k \end{bmatrix} + \begin{bmatrix} & \text{noise} & \end{bmatrix}$$

In this case, we estimate a basis for the null space of the "clean" matrix via the eigenspace corresponding to the k smallest eigenvalues

The $\mathbf{y}_i$ should cluster around k different values

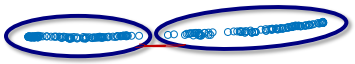
Simply use k-means clustering!

Spectral clustering

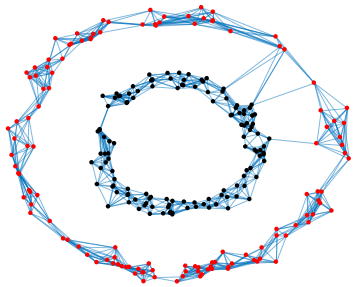
1. Input $\mathbf{x}_1, \ldots, \mathbf{x}_n$
2. Construct graph and graph Laplacian $\mathbf{L}$
3. Compute eigendecomposition of $\mathbf{L}$
   Set $\mathbf{u}_1, \ldots, \mathbf{u}_k$ to be the eigenvectors corresponding to the k smallest eigenvalues of $\mathbf{L}$
4. Apply k-means to the rows of $\mathbf{U} = [\mathbf{u}_1 \cdots \mathbf{u}_k]$
   Assign $\mathbf{x}_i$ to the same cluster as the i^{th} row of $\mathbf{U}$

Example

Scatter plot of rows of $\mathbf{U}$



Resulting clustering



Remarks

The ***normalized graph Laplacian*** is $\widetilde{\mathbf{L}} = \mathbf{D}^{-1}\mathbf{L}$

It is easy to check that we can substitute $\widetilde{\mathbf{L}}$ for $\mathbf{L}$ and get another spectral clustering algorithm

It is usually preferable to use $\widetilde{\mathbf{L}}$ because this accounts for the fact that some nodes are more highly connected than others

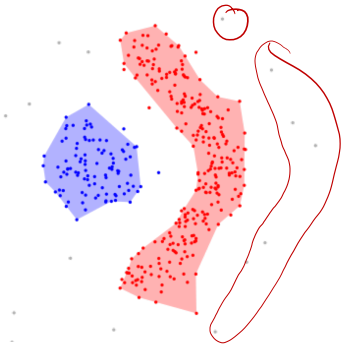
Selecting the optimal value of k typically boils down to looking for a jump in the sorted eigenvalues from “near zero” to “not near zero”

There are alternative derivations/interpretations of subspace clustering based on graph cuts and based on random walks

Density-based clustering

Yet another approach to clustering is built on the assumption that clusters can be defined by identifying areas of high density in the dataset

Produces a clustering of data points which are in dense areas, but typically leaves some data points un-labeled



DBSCAN

***Density-based spatial clustering of applications with noise* (DBSCAN)** is probably the most popular density-based clustering algorithm

Divides the dataset into three categories of points

- **Core points:** points in dense regions
- **Reachable points:** points which are near a core point
- **Outliers:** everything left over

DBSCAN has two parameters (p_{min} and ϵ) which help quantify how dense a cluster needs to be

Note: Does not require a priori determination of the number of clusters

Core points and reachability

An $\mathbf{x}_i$ is a ***core point*** if there are at least p_{min} points in the set $\{\mathbf{x} : \|\mathbf{x}_i - \mathbf{x}\|_2 \leq \epsilon\}$

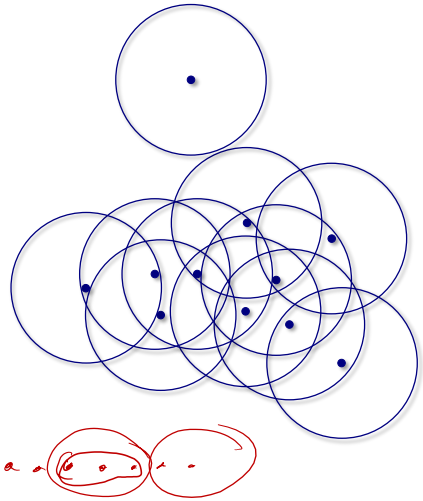
- these points are said to be *reachable* from $\mathbf{x}_i$

An $\mathbf{x}_j$ is ***reachable*** from $\mathbf{x}_i$if there is a sequence of points $\mathbf{p}_1, \ldots, \mathbf{p}_\ell$ satisfying

- $\mathbf{p}_1 = \mathbf{x}_i$ and $\mathbf{p}_\ell = \mathbf{x}_j$
- each $\mathbf{p}_k$ must be directly reachable from $\mathbf{p}_{k-1}$ (all $\mathbf{p}_1, \ldots, \mathbf{p}_{\ell-1}$ must be core points)

If $\mathbf{x}_i$ is a core point, then it forms a cluster together with all points (core and non-core) that are reachable from $\mathbf{x}_i$

Example



$p_{\text{min}} = 4$

Remarks

- Clustering can be computed in a simple iterative fashion
  - find a core point
  - add its reachable points
  - iterate until all points are either clustered or labeled as outliers
- Algorithm can lead to slightly different clusterings depending on the order in which the points are processed
  - “border points” can sometimes be assigned to multiple clusters, depending on which one is grown first
- Can struggle with clusters of varying density
  - see extensions, e.g., OPTICS

Hierarchical clustering

In practice, we might be interested in learning a ***hierarchical clustering***, where clusters are nested inside larger clusters

There are two main approaches to hierarchical clustering

- Bottom-up: ***agglomerative clustering*** builds up a hierarchy by starting with a fine-grain clustering and merging clusters together
- Top-down: ***divisive clustering*** builds up a hierarchy by iteratively dividing existing clusters into smaller components

Agglomerative clustering

Initialize by defining each $\mathbf{x}_1, \ldots, \mathbf{x}_n$ to be its own cluster

At each iteration compute evaluate some metric measuring how similar each possible pair of clusters are

Merge the most similar pair of clusters

Repeat until all clusters are merged

Cluster similarity metrics

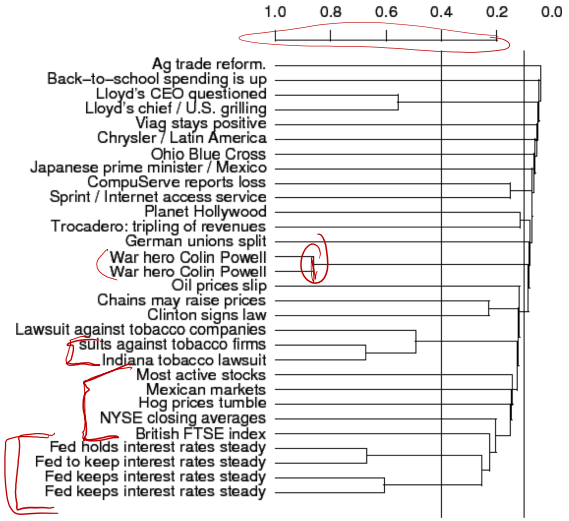
Suppose that $\mathcal{A}$ and $\mathcal{B}$ represent the index sets for two clusters which are candidates for being merged

How can we measure how similar these two clusters are?

Some common choices include:

- nearest neighbor: $\min_{i \in \mathcal{A}, j \in \mathcal{B}} \|\mathbf{x}_i - \mathbf{x}_j\|_2$
- farthest neighbor: $\max_{i \in \mathcal{A}, j \in \mathcal{B}} \|\mathbf{x}_i - \mathbf{x}_j\|_2$
- average: $\frac{1}{|\mathcal{A}||\mathcal{B}|} \sum_{i \in \mathcal{A}} \sum_{j \in \mathcal{B}} \|\mathbf{x}_i - \mathbf{x}_j\|_2$

Example



dendrogram

Manning, Raghavan, and Schütze (2008)

Divisive clustering

Begin with the entire dataset belonging to a single cluster

At each iteration, need to

- select an existing cluster to divide
  - can use a metric like within-cluster scatter
- divide the selected cluster into two child clusters
  - e.g., using any clustering algorithm we have seen so far

Can be computationally demanding, and is somewhat less well studied than agglomerative clustering