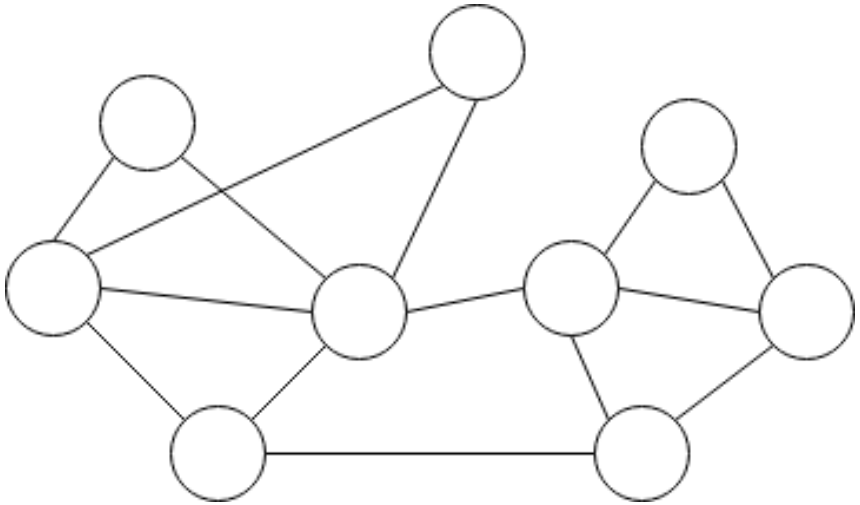
Proximity-based Clustering

Clustering with no distance information

- What if one wants to cluster objects where only similarity relationships are given?

Consider the following visualization of relationships between 9 objects



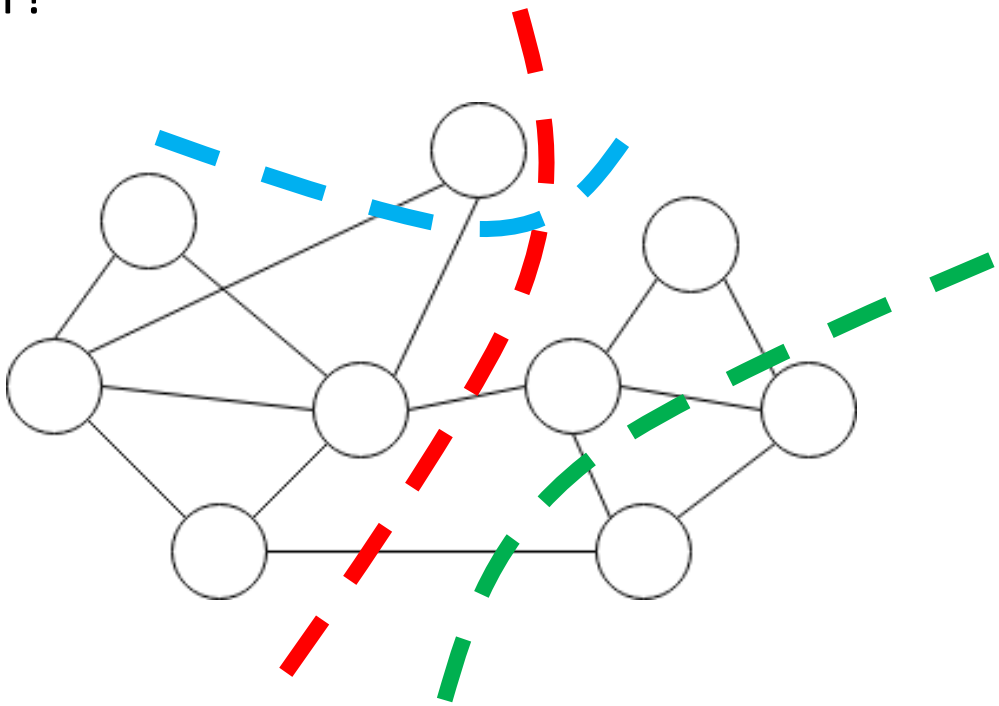
- Nodes are the objects
- Edges are pairwise relationships

- Not embeddable in Euclidean space
- Not even a metric space! ☹

So how can we proceed with clustering??

Clustering with no distance information

- Say k = 2 (ie partition the objects in two cluster), what would be a reasonable answer?

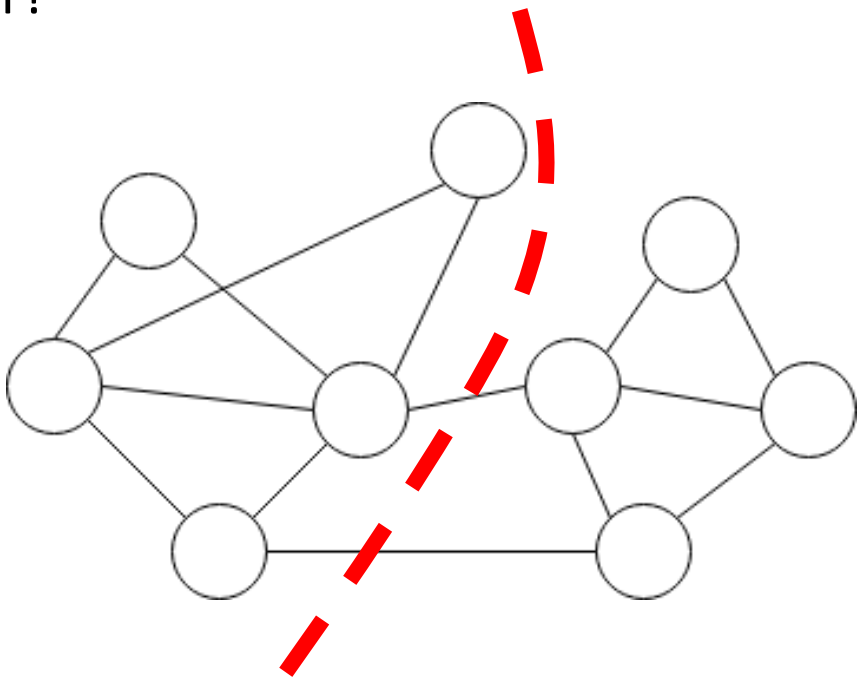


Since edges indicate similarity, want to find a cut that minimizes crossings

Which of the three partitions is most preferable? Why?

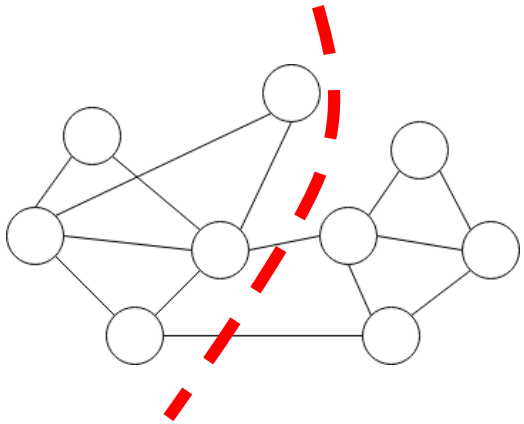
Clustering with no distance information

- Say k = 2 (ie partition the objects in two cluster), what would be a reasonable answer?



Want a cut which minimizes crossings, but also keep cluster/partition sizes large

Clustering by finding “balanced” cut



Let the two partitions be P and P’, then we can minimize the following

$$\min_{P,P'} \frac{\text{cut}(P,P')}{\text{vol}(P)} + \frac{\text{cut}(P',P)}{\text{vol}(P')}$$

‘cut’ is the number of edges across a partition

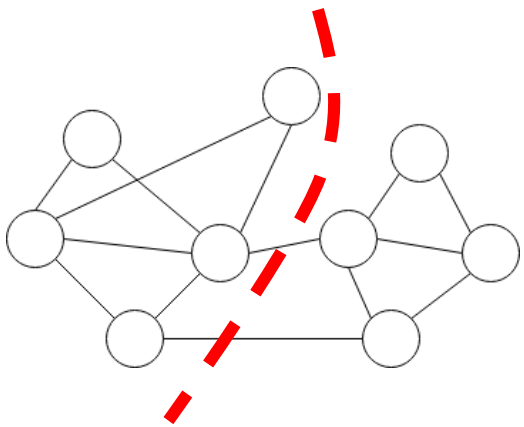
‘vol’ is the number of edges within a partition

[Shi and Malik ’00]

In general, for *k* partitions the optimization generalizes to

$$\min_{P_1,\ldots,P_k} \sum_i \frac{\text{cut}(P_i,\bar{P}_i)}{\text{vol}(P_i)}$$

Clustering by finding “balanced” cut



Let the two partitions be P and P’, then we can minimize the following

$$\min_{P,P'} \frac{\mathrm{cut}(P,P')}{\mathrm{vol}(P)} + \frac{\mathrm{cut}(P',P)}{\mathrm{vol}(P')}$$

‘cut’ is the number of edges across the partition

So how can we minimize above?

Let’s simplify it further…

$$\frac{\mathrm{cut}(P,P')}{\mathrm{vol}(P)} = \frac{(\mathbf{1}_P)^\mathsf{T} L(\mathbf{1}_P)}{\mathrm{vol}(P)}$$

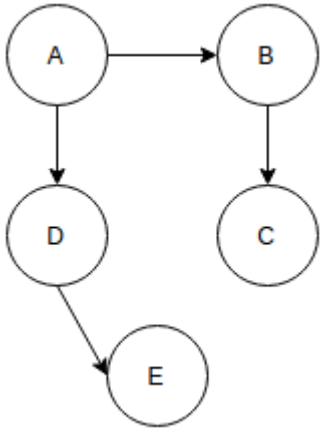
$$= \left(\frac{\mathbf{1}_P}{\sqrt{\mathrm{vol}(P)}}\right)^\mathsf{T} L \left(\frac{\mathbf{1}_P}{\sqrt{\mathrm{vol}(P)}}\right)$$

$\mathbf{1}_P$ = indicator vector on P

L = graph Laplacian

Detour: The (graph) Laplacian

Given an (unweighted) directed graph G = (V, E)



Consider the incidence matrix C representation of the graph G

Vertices

Edges	A	B	C	D	E
e_1	1	-1			
e_2	1			-1	
e_3		1	-1		
e_4				1	-1

For each edge in the graph:

- +1 on source vertex
- -1 on the destination vertex

Define Graph Laplacian L as… $L := C^T C$

The graph Laplacian

Hence, $L = C^T C = \begin{pmatrix} e_1 & e_2 & \dots & e_m \end{pmatrix} \begin{pmatrix} e_1^T \\ e_2^T \\ \dots \\ e_m^T \end{pmatrix} = \sum_k e_k e_k^T$

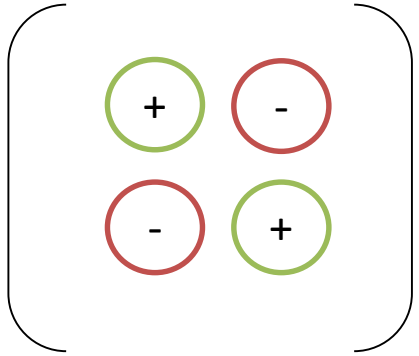
$C = \begin{pmatrix} e_1^T \\ e_2^T \\ \dots \\ e_m^T \end{pmatrix}$

PSD!

Say e_k is an edge (i,j), then

$$e_k = \begin{pmatrix} \dots \\ 1 \\ \dots \\ -1 \\ \dots \end{pmatrix} \begin{matrix} \\ i \\ \\ j \\ \end{matrix}$$

$$e_k e_k^T = \begin{matrix} \\ i \\ \\ j \\ \end{matrix} \begin{pmatrix} \dots \\ 1 \\ \dots \\ -1 \\ \dots \end{pmatrix} \begin{matrix} i & j \\ \dots & 1 \dots -1 \dots \end{matrix}$$



- diagonals always positive
- off-diagonals always negative

$L = D - W$

- D degree matrix (diagonal)
- W weight matrix

But why is L=D-W called a Laplacian?

Let's consider the Laplace operator from calculus…

For a function $f : R^d \rightarrow R$, Laplace Δ of f is defined as

Δf := divergence of the gradient of f

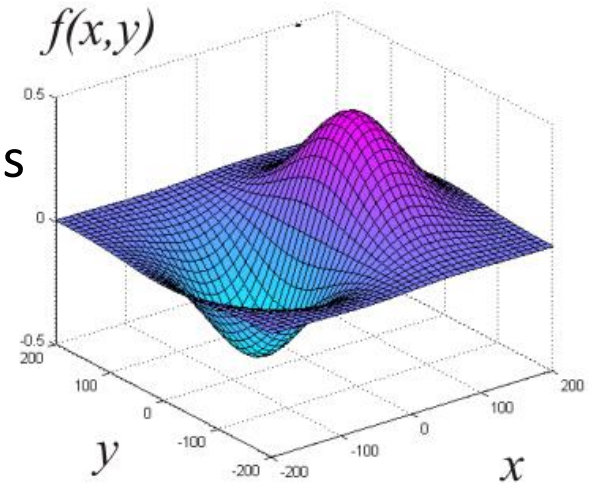
$= \nabla . \nabla f$

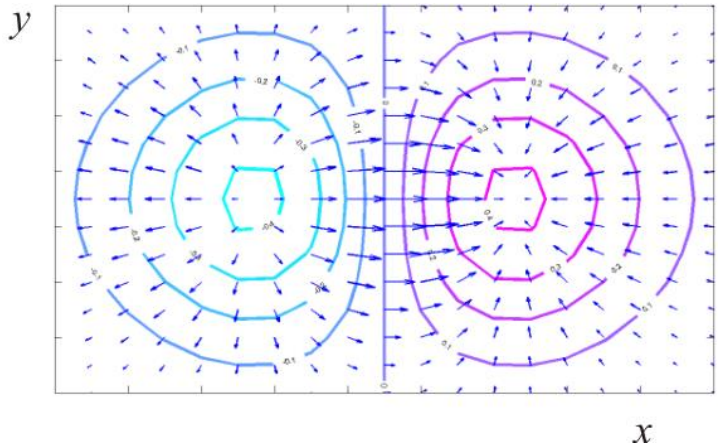
$$= \begin{pmatrix} \partial/x_1 \\ \partial/x_2 \\ \dots \\ \partial/x_d \end{pmatrix} \cdot \begin{pmatrix} \partial/x_1 \\ \partial/x_2 \\ \dots \\ \partial/x_d \end{pmatrix} f$$

$= \sum_i \partial^2 f / \partial x_i^2$

= Trace of the Hessian of f

$\approx$ (mean) curvature



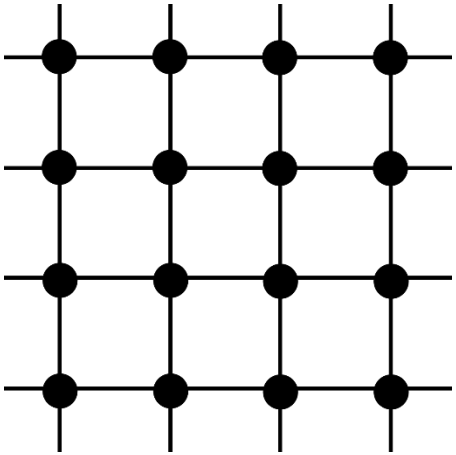


L pos, if net gradient flow is OUT (ie pos divergence)

L neg, if net gradient flow is IN (ie neg divergence)

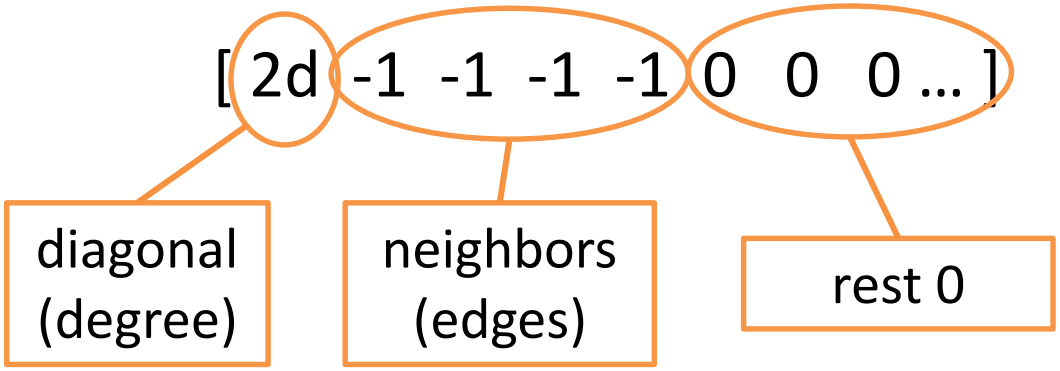
Relationship of Laplacian to graph Laplacian

Consider a discretization of R^d , ie a regular lattice graph



The (graph) Laplacian of this graph

Each row/col of L looks as:

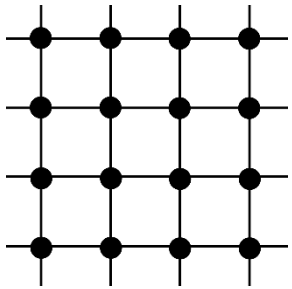


For better understanding, consider each coordinate direction

[… 0 0 0 -1 2 -1 0 0 0 …]

This acts like (discretized version of) the (negative) second derivative!!

Graph Laplacian of Regular Lattice



Each coordinate looks like

[… 0 0 0 -1 2 -1 0 0 0 …]

This acts like (discretized version of) the (negative) second derivative!!

Consider the finite difference method for derivatives…

- (forward) difference: $f' = f(x+h) - f(x) / h$
- (backward) difference: $f' = f(x) - f(x-h) / h$

So the second order (central) difference:

[+1 -2 +1]

That is, -2 on self, +1 on neighbors

$$f'' = \frac{\frac{f(x+h)-f(x)}{h} - \frac{f(x)-f(x-h)}{h}}{h} = \frac{f(x+h) - 2f(x) + f(x-h)}{h^2}$$

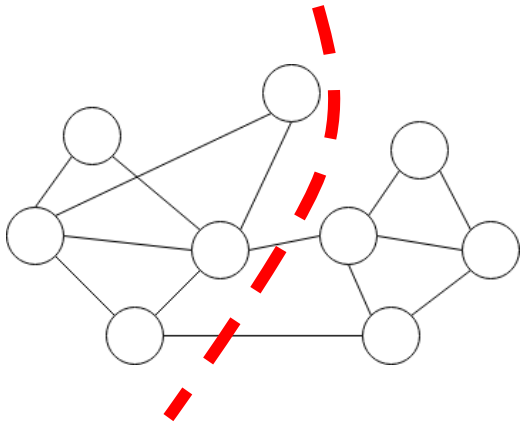
Graph Laplacian Properties

The **graph Laplacian** captures the second order information about a function (on vertices), it can quantify how 'wiggly' a (vertex) function is.

Applications:

- Quantify the (average) rate of change of a function (on vertices)
- One can try to **minimize the curvature** to derive 'flatter' representations
- Can be used as a **regularizer** to penalize the complexity of a function
- Can be used for **clustering**!!
- …

OK… Back to Clustering



Let the two partitions be P and P', then we can minimize the following

$$\min_{P,P'} \frac{\text{cut}(P,P')}{\text{vol}(P)} + \frac{\text{cut}(P',P)}{\text{vol}(P')}$$

'cut' is the number of edges across the partition

So how can we minimize above?

Let's simplify it further…

$$\frac{\text{cut}(P,P')}{\text{vol}(P)} = \frac{(\mathbf{1}_P)^\top L(\mathbf{1}_P)}{\text{vol}(P)}$$

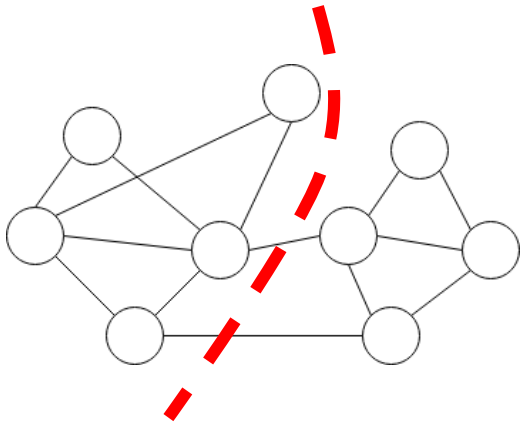
$$= \left(\frac{\mathbf{1}_P}{\sqrt{\text{vol}(P)}}\right)^\top L \left(\frac{\mathbf{1}_P}{\sqrt{\text{vol}(P)}}\right)$$

$\mathbf{1}_P$ = indicator vector on P

L = graph Laplacian

OK… Back to Clustering

So the optimization $\min_{P,P'} \frac{\text{cut}(P,P')}{\text{vol}(P)} + \frac{\text{cut}(P',P)}{\text{vol}(P')}$



can be re-written as

$$\min_{f_1,f_2} \sum_{i=1}^{2} f_i^{\mathsf{T}} L f_i$$

$$\text{s.t.} \quad f_i^{\mathsf{T}} D f_i = 1 \qquad f_i^{\mathsf{T}} f_j = 0$$

all entries of f_i are equal

Since we are minimizing a quadratic form subject to orthogonality constraints, we can approximate the solution via a generalized eigenvalue system!

Generalized eigensystem… $Ax = \lambda Dx$

Since spectral decomposition in used to determine f ie clusters, this methodology is called spectral clustering

Spectral Clustering: the Algorithm

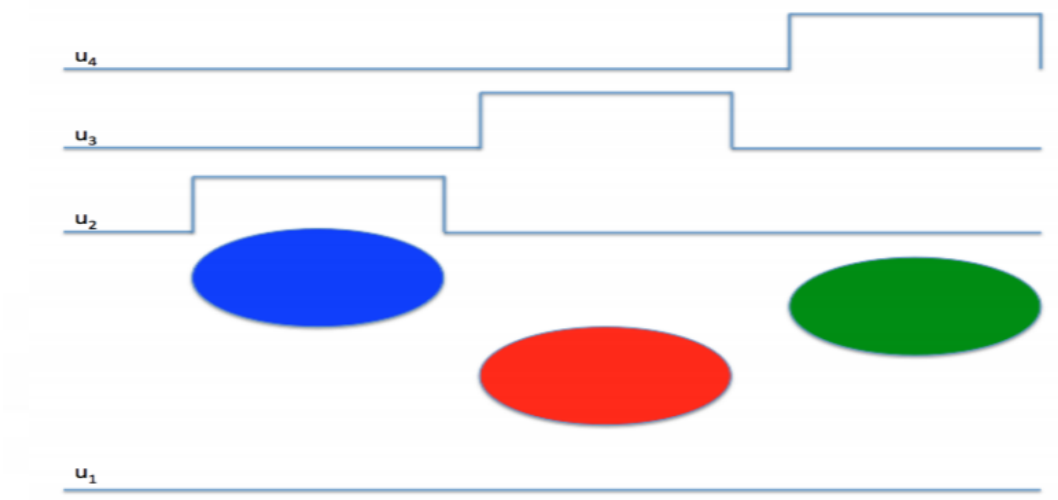
Input: *S*: *n* x *n* similarity matrix (on *n* datapoints), *k*: # of clusters

- Compute the degree matrix *D* and adjacency matrix *W* from the *weighted* graph induced by *S* *since the graph is weighted, $d_i = \sum_j s_{ij}$, $w_{ij} = s_{ij}$*
- Compute the graph Laplacian L = D – W
- Compute the *bottom k* eigenvectors $u_1,\ldots,u_k$ of the generalized eigensystem: $\mathbf{Lu} = \lambda\mathbf{Du}$
- Let U be the *n* x *k* matrix containing vectors $u_1,\ldots,u_k$ as columns
- Let y_i be the i^{th} row of U; it corresponds to the *k* dimensional representation of the datapoint x_i
- Cluster points $y_1,\ldots,y_n$ into *k* clusters via a centroid-based alg. like *k*-means

Output: the partition of *n* datapoints returned by *k*-means as the clustering

Spectral Clustering: the Geometry

- The eigenvectors are an approximation to the *f* partition 'indicator' vectors in the normalized cut problem.

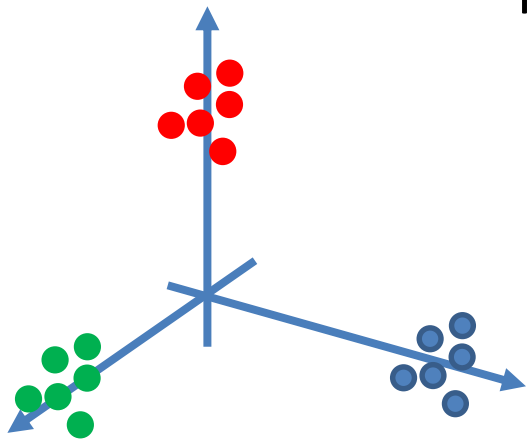


Learned Indicator vectors

Spectral trans-formation via L

R^k

Data in original space, similar points can be located ***anywhere*** in the original space



Data is ***easy*** to cluster in the new transformation

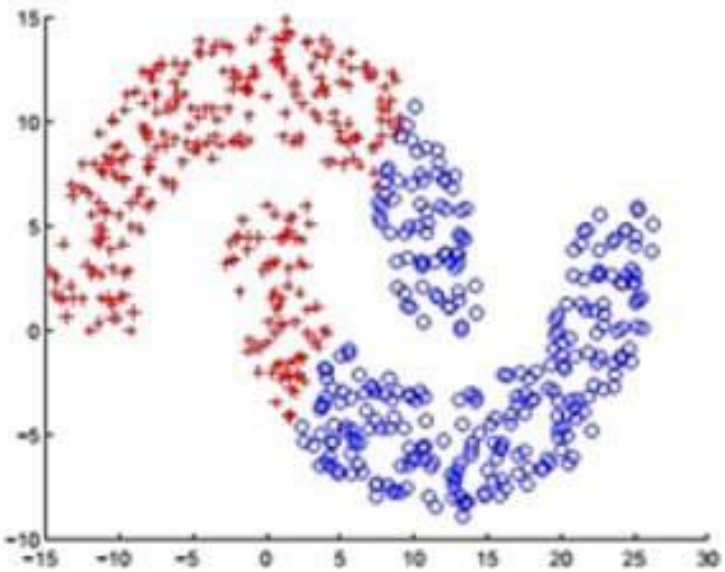
Spectral Clustering: Dealing with Similarity

- What if similarity information is unavailable?

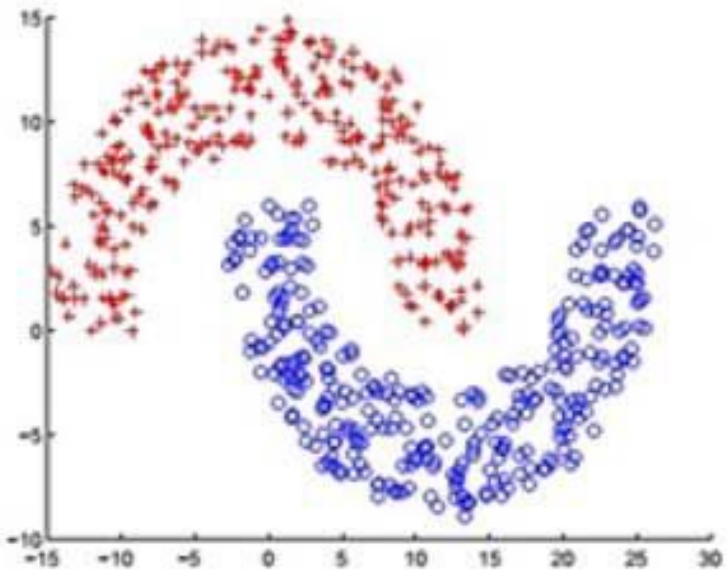
  If distance information is available, one can usually compute similarity as

$$e^{-\mathrm{dist}^2/\sigma^2}$$

Spectral Clustering in Action

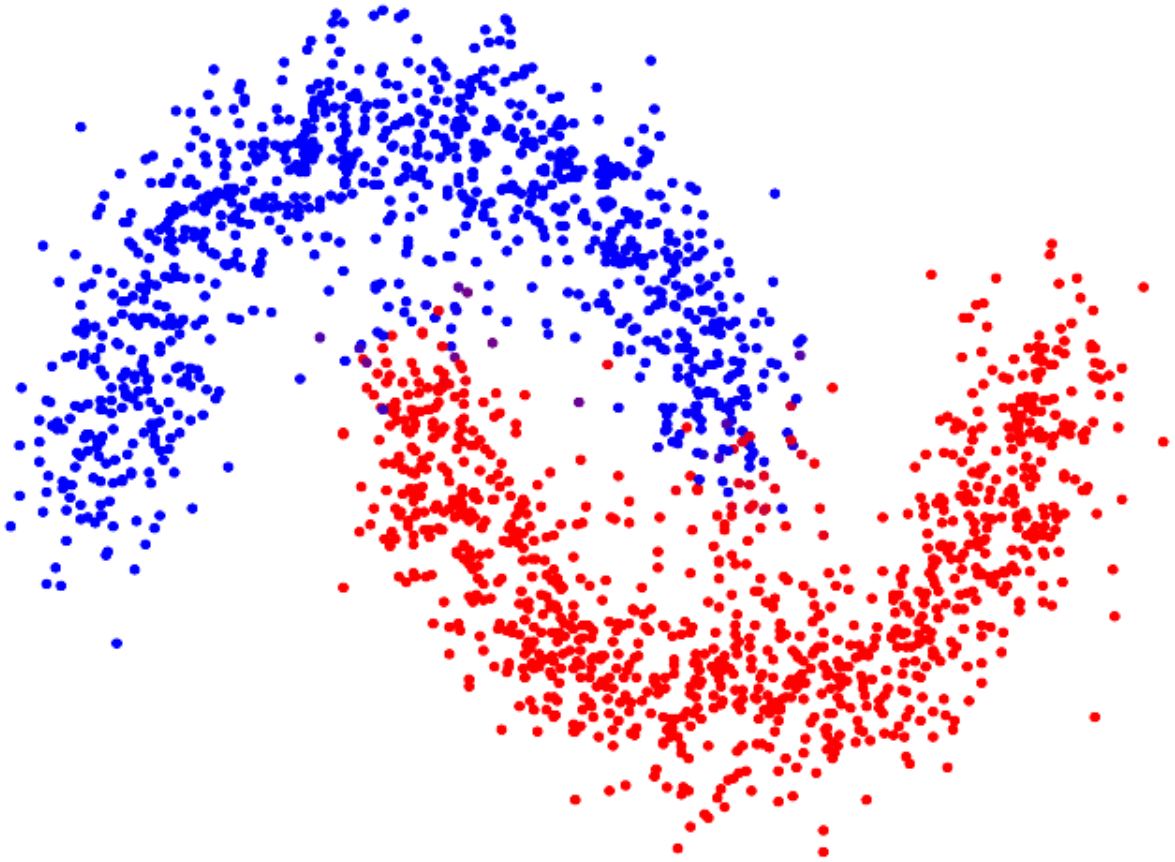


(a) K-means

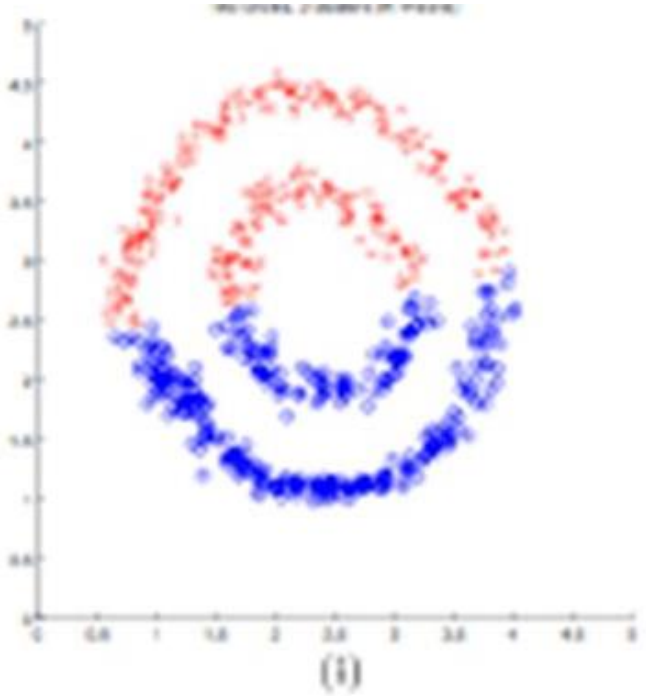


(b) Spectral Clustering

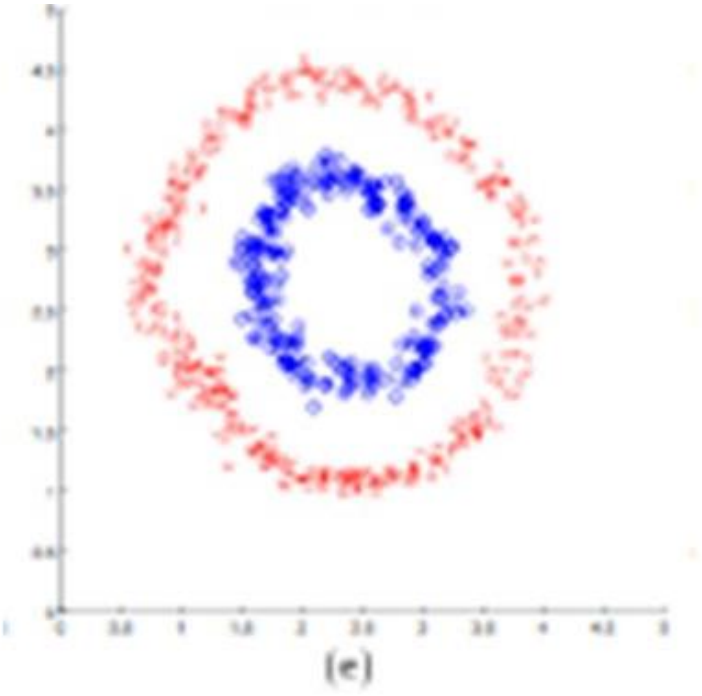
Spectral Clustering in Action



Spectral Clustering in Action



K-means



Spectral Clustering

Spectral Clustering in Action