Dimensionality Reduction
contd …

Aarti Singh

Machine Learning 10-601
Nov 10, 2011

Slides Courtesy: Tom Mitchell, Eric Xing, Lawrence Saul

Principal Component Analysis (PCA)

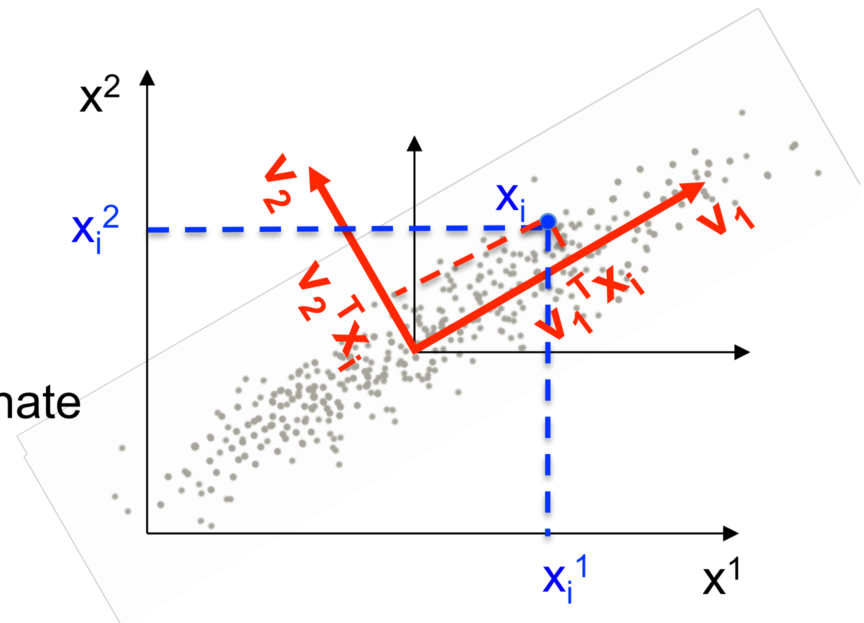
Principal Components are the eigenvectors of the matrix of sample correlations XX^T of the data

New set of axes $V = [v_1, v_2, \ldots, v_D]$ where $XX^T = V\Lambda V^T$

- Geometrically: centering followed by rotation
  - Linear transformation

Original representation of data points
$x_i = [x_i^1, x_i^2, \ldots, x_i^D]$

$$x_i^j = e_j^T x_i \quad \text{where } e_j = [\,0 \ldots 0\ \underbrace{1}_{j^{th}\text{ coordinate}}\ 0 \ldots 0\,]$$



Transformed representation of data points
$[v_1^T x_i, v_2^T x_i, \ldots v_D^T x_i]$

Dimensionality Reduction using PCA

Original Representation $[x_i^1, x_i^2, \ldots, x_i^D]$ (D-dimensional vector)

$$x_i = \sum_{j=1}^{D} x_i^j e_j = \sum_{j=1}^{D} (e_j^T x_i) e_j$$

$(x_i^j)^2 = (e_j^T x_i)^2$ = energy/variance of data point i along coordinate j

Transformed representation $[v_1^T x_i, v_2^T x_i, \ldots v_D^T x_i]$ (D-dimensional vector)

$$x_i = \sum_{j=1}^{D} (v_j^T x_i) v_j$$

$(v_j^T x_i)^2$ = energy/variance of data point i along principal component v_j

$$\lambda_j = \sum_{i=1}^{n} (v_j^T x_i)^2 = \text{energy/variance of all points along } v_j$$

Dimensionality reduction $[v_1^T x_i, v_2^T x_i, \ldots v_d^T x_i]$ (d-dimensional vector)

$$\hat{x}_i = \sum_{j=1}^{d} (v_j^T x_i) v_j$$

Only keep data projections onto principal components which capture enough energy/variance of the data $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_D$

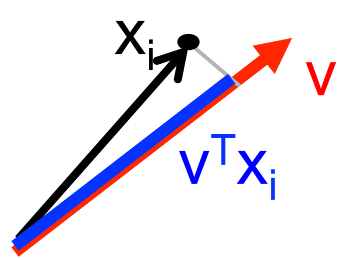
Another interpretation

Maximum Variance Subspace: PCA finds vectors v such that projections on to the vectors capture maximum variance in the data

$$\sum_{i=1}^{n} (\mathbf{v}^T \mathbf{x}_i)^2 = \mathbf{v}^T \mathbf{X}\mathbf{X}^T \mathbf{v}$$

Minimum Reconstruction Error: PCA finds vectors v such that projection on to the vectors yields minimum MSE reconstruction

$$\sum_{i=1}^{n} \|\mathbf{x}_i - \underbrace{(\mathbf{v}^T \mathbf{x}_i)\mathbf{v}}_{\text{One direction approximation}}\|^2$$



Recall: $$\mathbf{x}_i = \sum_k (\mathbf{v}_k^T \mathbf{x}_i) \cdot \mathbf{v}_k$$

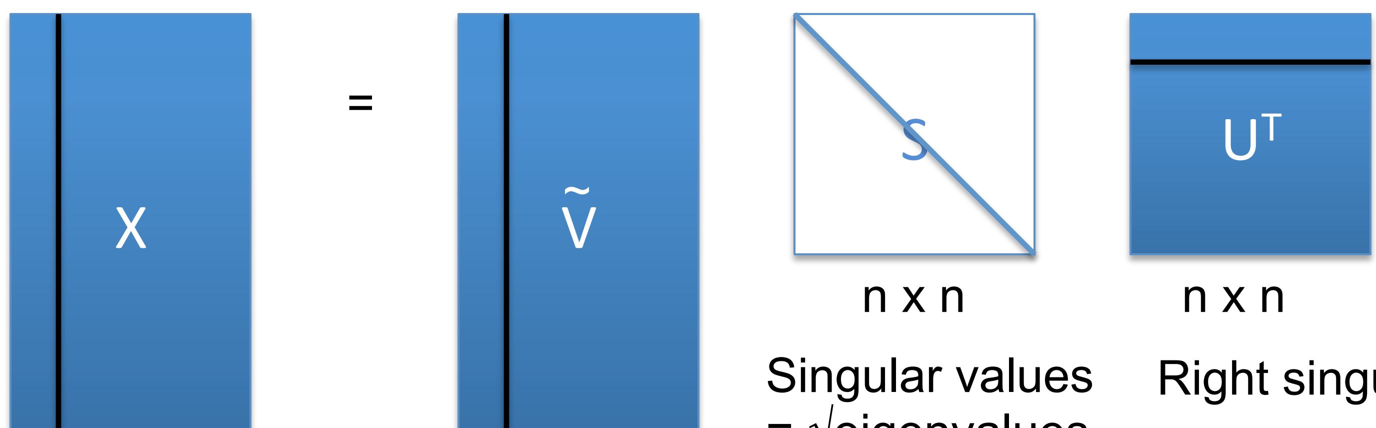
Another way to compute PCs

Principal Components – Eigenvectors of XX^T (D x D matrix)

Problematic for high-dimensional datasets!

Another way to compute PCs: **Singular Vector Decomposition (SVD)**

$$X = \tilde{V} S U^T \Longrightarrow X X^T = \tilde{V} S U^T U S \tilde{V}^T = \tilde{V} S^2 \tilde{V}^T = \tilde{V} \Lambda \tilde{V}^T$$



n x n

Singular values = √eigenvalues

n x n

Right singular vectors

$$x_i = \begin{bmatrix} x_i^1 \\ x_i^2 \\ \vdots \\ x_i^D \end{bmatrix}$$

D x n

D x n

Left singular vectors

v_j = Principal Components

Another way to compute PC projections

Singular Vector Decomposition $X = \tilde{V} S U^T$

Projection of data points on to PCs

$[v_1^T x_i, v_2^T x_i, \ldots v_n^T x_i] = [\sigma_1 u_1(i), \sigma_2 u_2(i), \ldots \sigma_n u_n(i)]$

SVD $\Longrightarrow$ $$\tilde{V}^T X = \tilde{V}^T \tilde{V} S U^T = S U^T$$

(since $\tilde{V}^T \tilde{V} = I$ eigenvectors are orthornormal)

U and S can be obtained by eigendecomposition of $X^T X$!

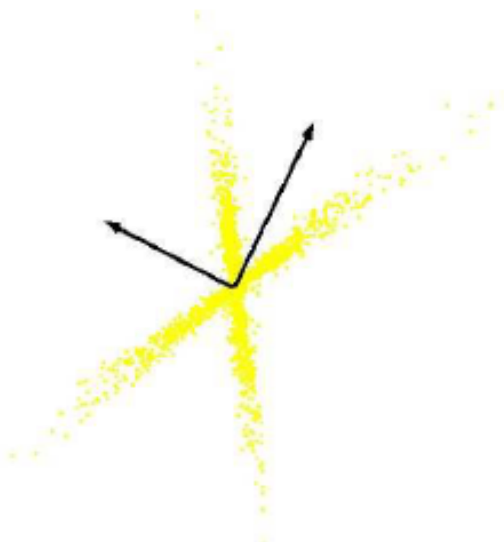
$$X^T X = U S \tilde{V}^T \tilde{V} S U^T = U S^2 U^T$$ (n x n matrix)

Principal Components are obtained by
Eigendecomposition of XX^T (D x D matrix)

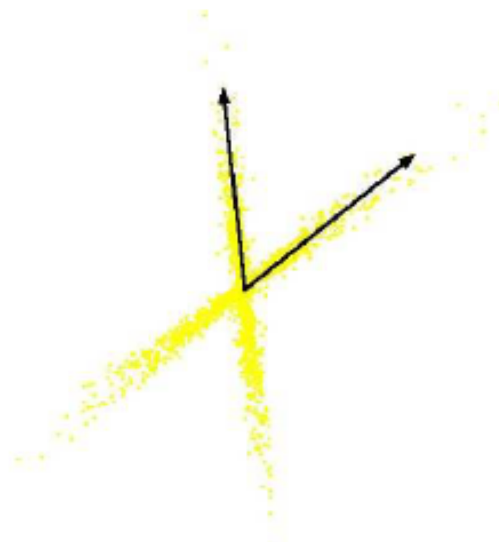
Projection of data points on to PCs can be obtained by
Eigendecomposition of $X^T X$ (n x n matrix)

Independent Component Analysis (ICA)

- PCA seeks "orthogonal" directions that capture maximum variance in data, or that minimize squared reconstruction error.
- ICA seeks "statistically independent" directions in the data

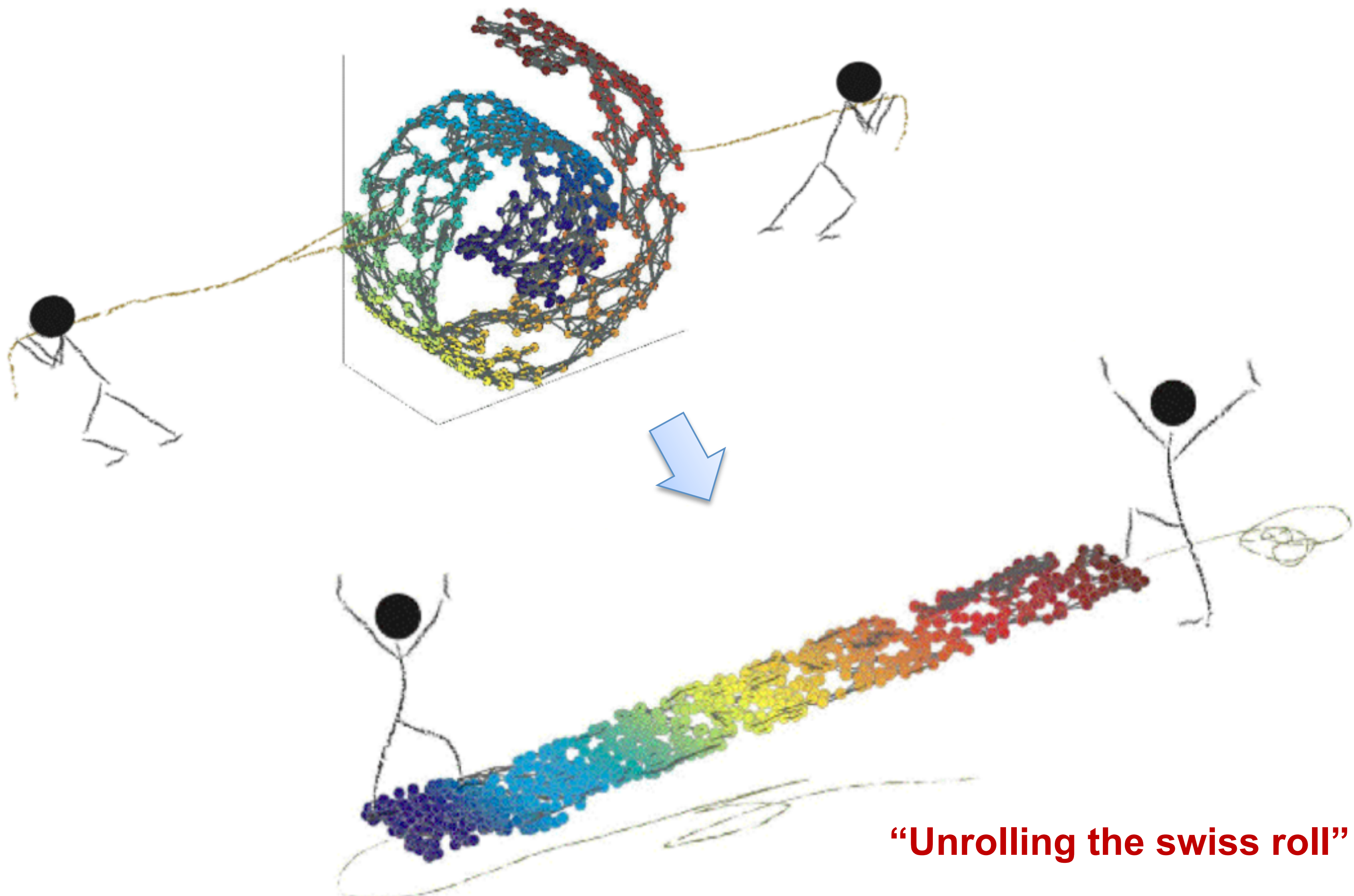


PCA
(orthogonal coordinate)



ICA
(non-orthogonal coordinate)

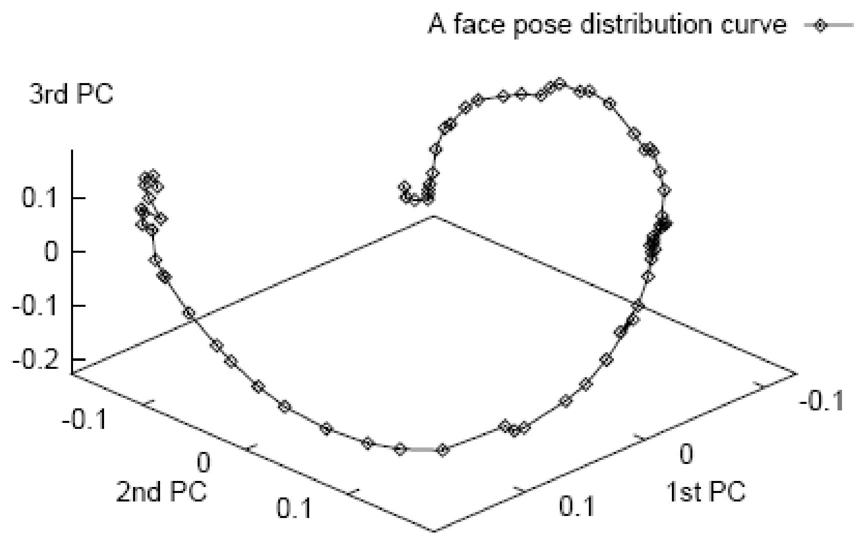
Dimensionality Reduction



"Unrolling the swiss roll"

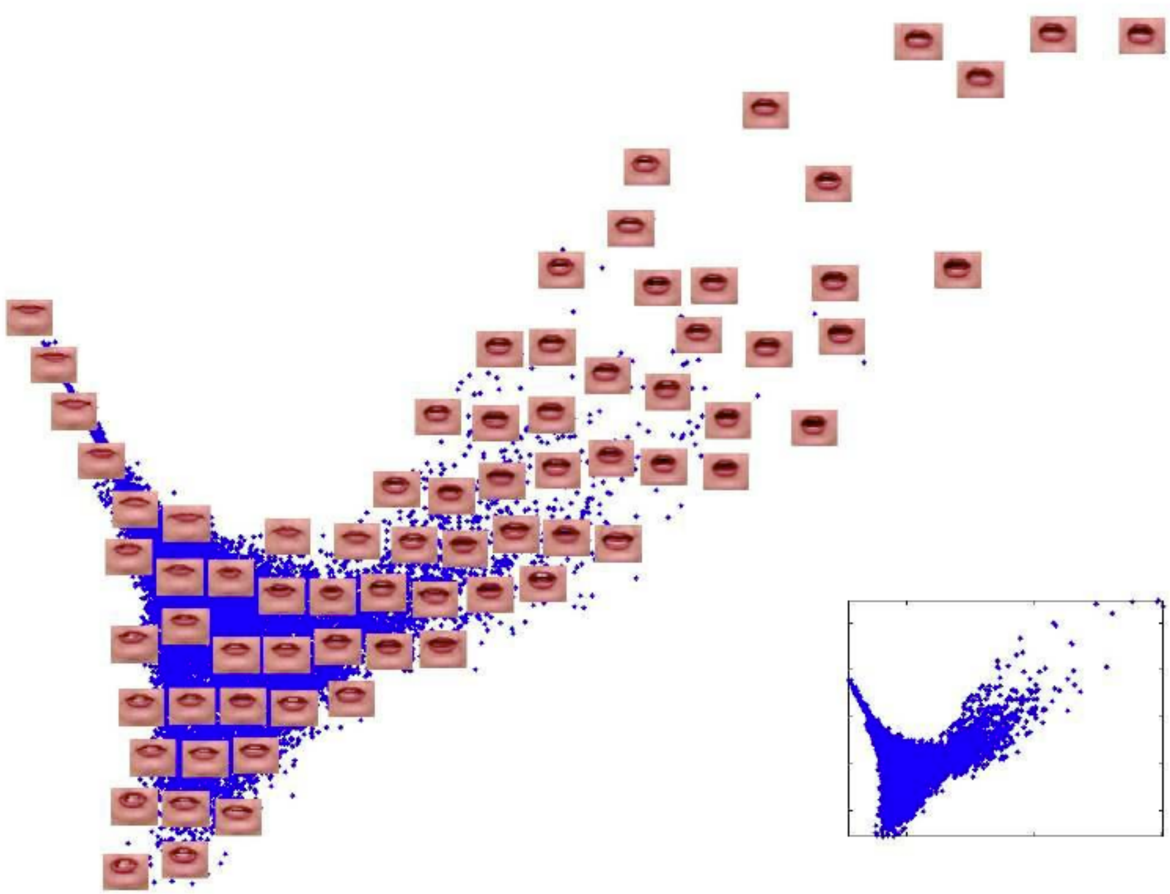
Nonlinear Methods

Data often lies on or near a nonlinear low-dimensional curve aka manifold.



Nonlinear Methods

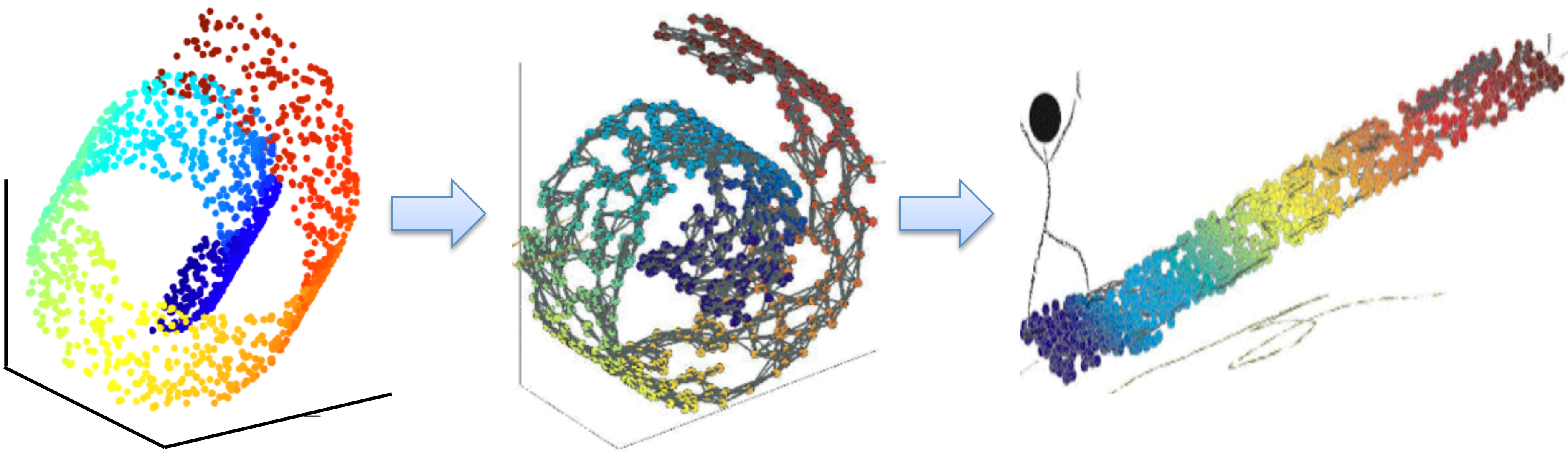
Data often lies on or near a nonlinear low-dimensional curve aka manifold.



Laplacian Eigenmaps

Linear methods – Lower-dimensional linear projection that preserves distances between **all** points

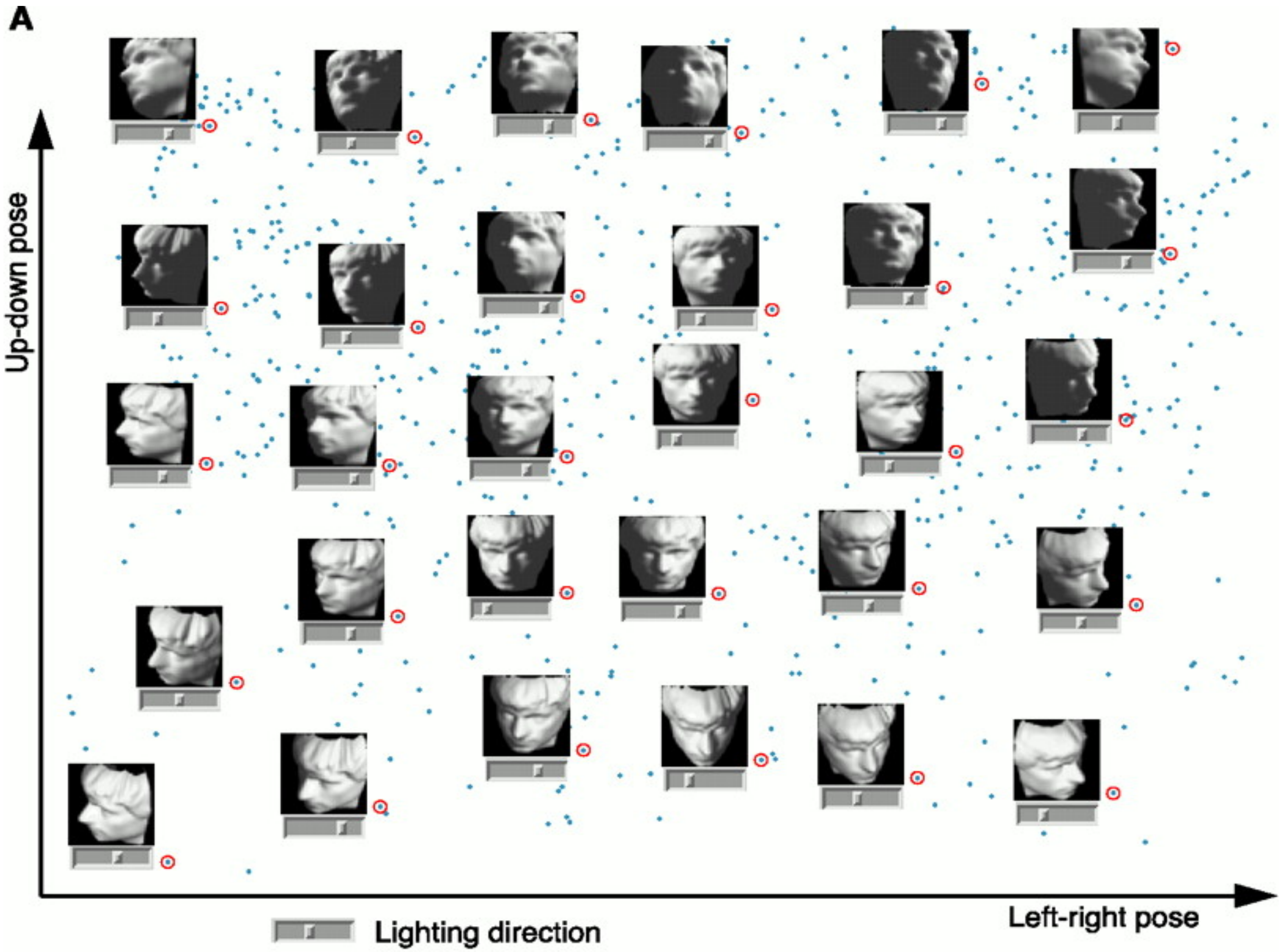
Laplacian Eigenmaps (key idea) – preserve **local** information only



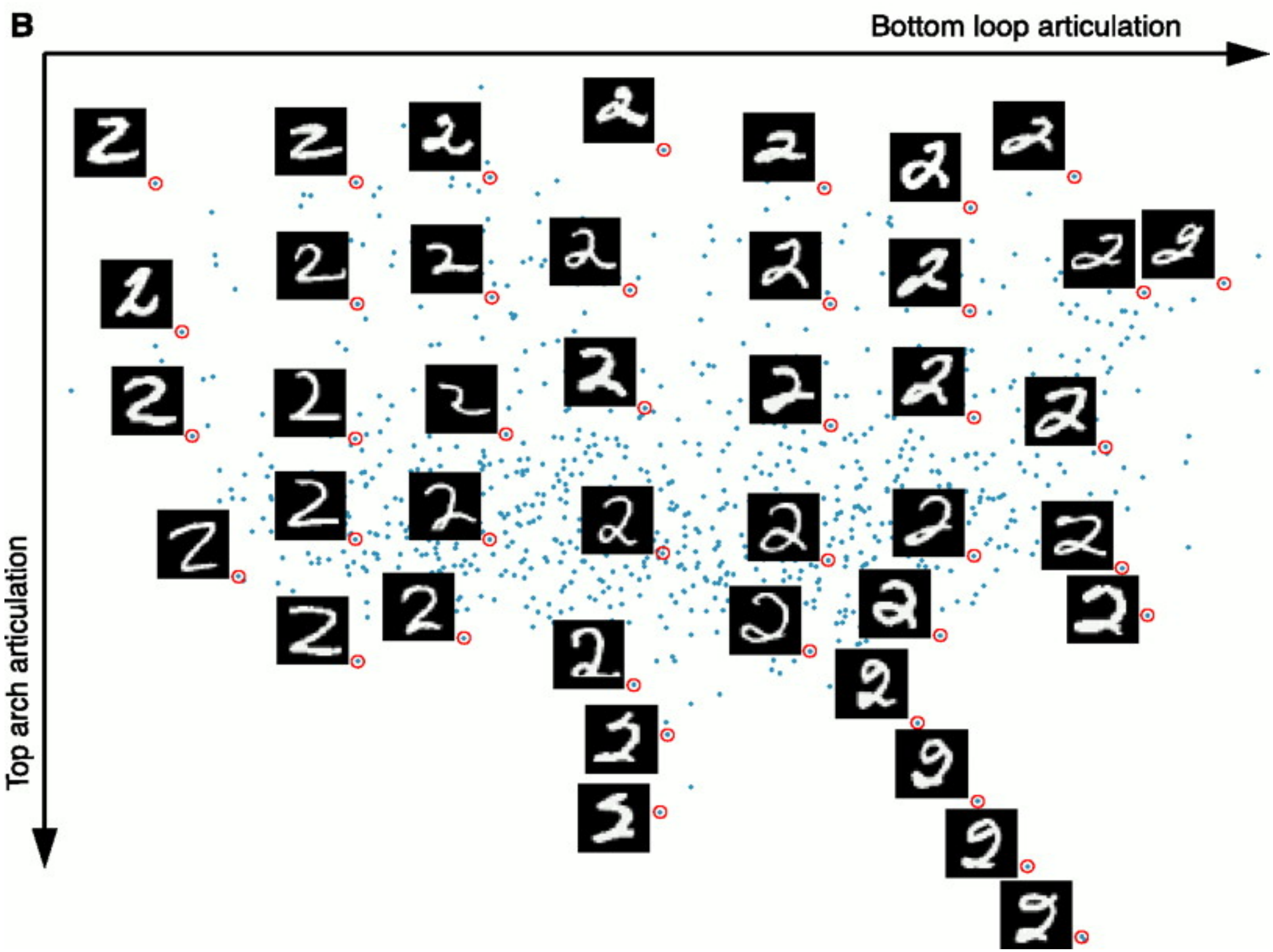
Construct graph from data points (capture local information)

Project points into a low-dim space using “eigenvectors of the graph”

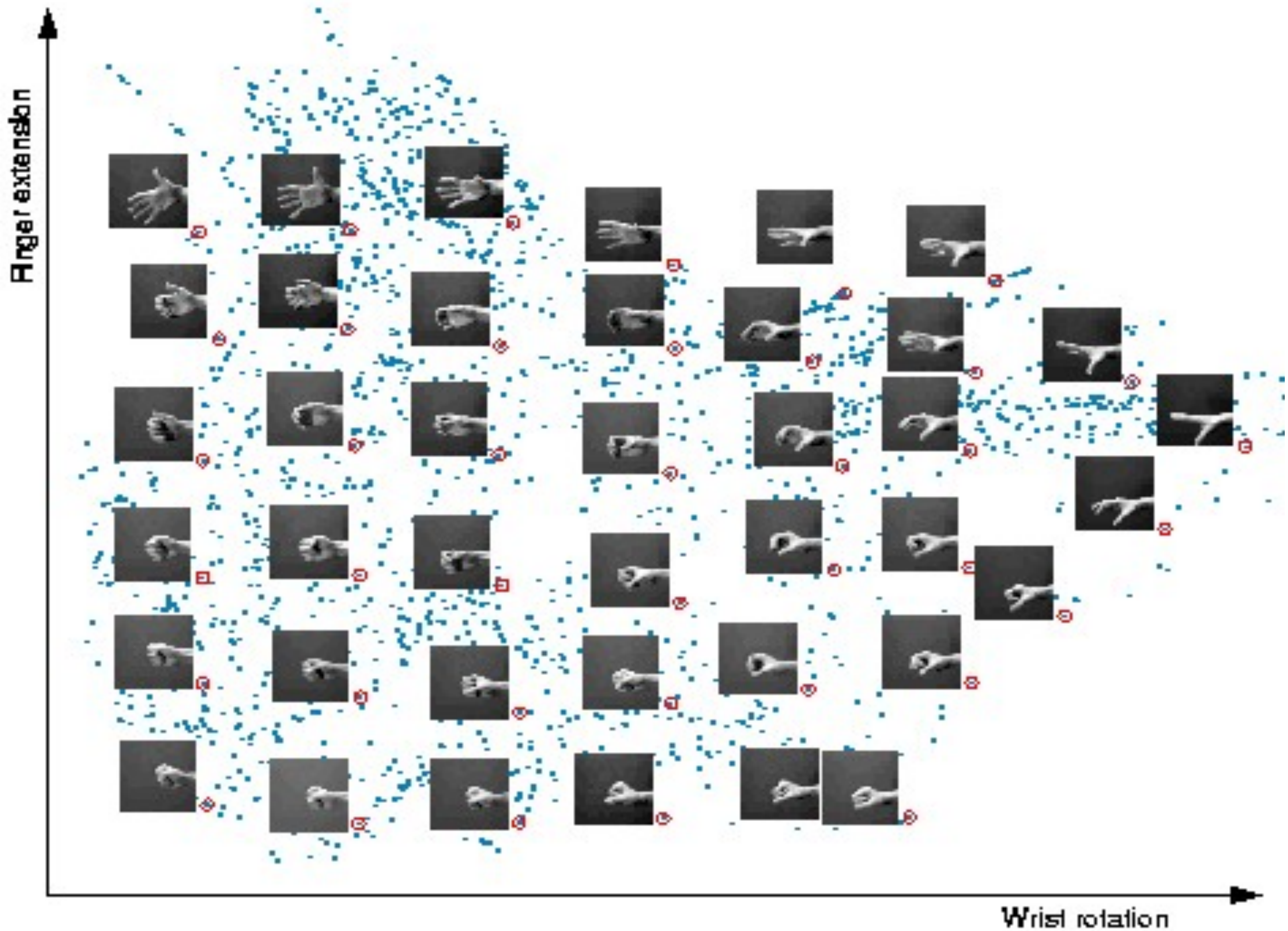
Nonlinear Embedding Results



Nonlinear Embedding Results



Nonlinear Embedding Results



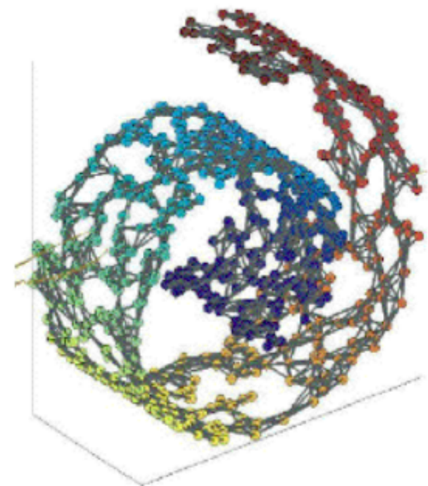
Step 1 - Graph Construction

Similarity Graphs: Model local neighborhood relations between data points

G(V,E,W)

V – Vertices (Data points)

E – Edges



(1) E – Edge if $||x_i - x_j|| \leq \varepsilon$ (ε – neighborhood graph)

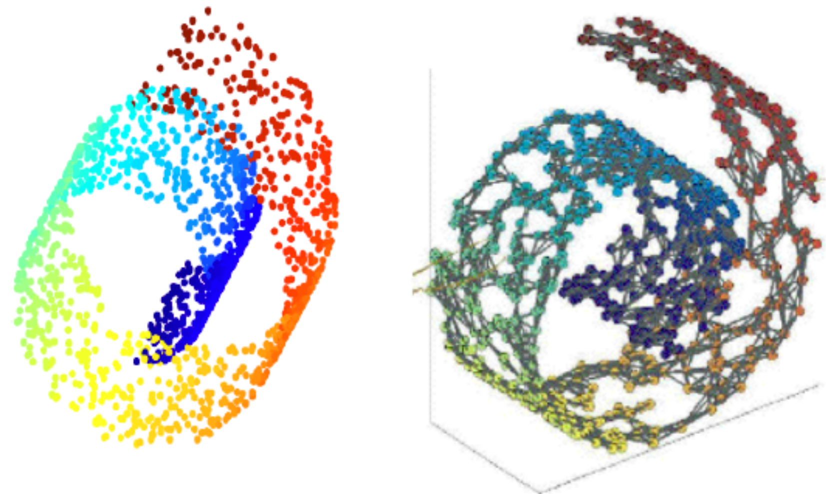
(2) E – Edge if k-nearest neighbor (k-NN graph)

Step 1 - Graph Construction

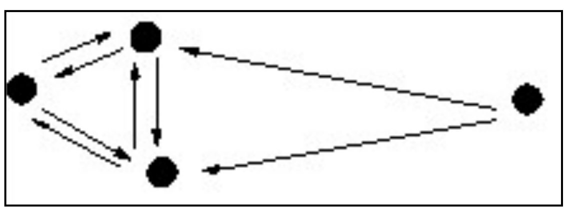
Similarity Graphs: Model local neighborhood relations between data points

(2) E – Edge if k-nearest neighbor (k-NN)

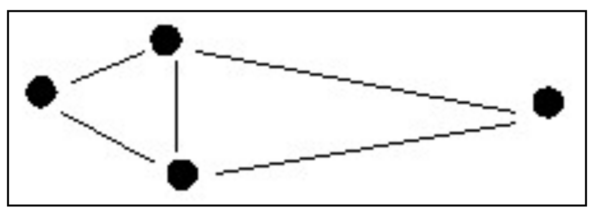
yields directed graph



connect A with B if A → B OR A ← B (symmetric kNN graph)
connect A with B if A → B AND A ← B (mutual kNN graph)



Directed nearest neighbors



(symmetric) kNN graph

mutual kNN graph

Step 1 - Graph Construction

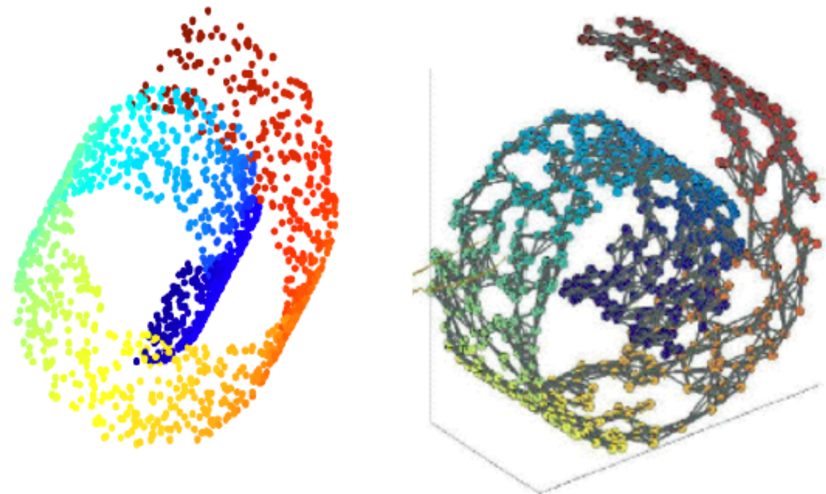
Similarity Graphs: Model local neighborhood relations between data points

G(V,E,W)

V – Vertices (Data points)

E – Edges

W – Weights



(1) W - W_{ij} = 1 if edge present, 0 otherwise

(2) W - $W_{ij} = e^{-\frac{\|x_i - x_j\|^2}{2\sigma^2}}$ Gaussian kernel similarity function (aka Heat kernel)

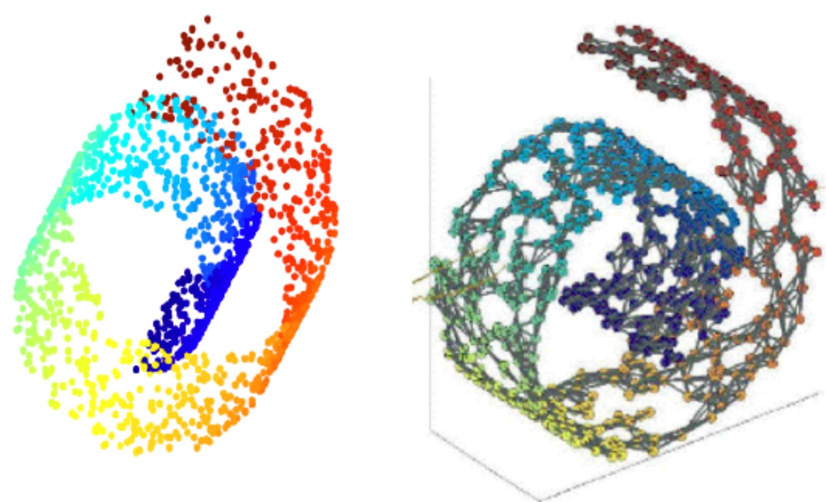
Step 1 - Graph Construction

Similarity Graphs: Model local neighborhood relations between data points

Choice of σ^2, ε and k :

ε, k - Chosen so that neighborhood on graphs represent neighborhoods on the manifold (no “shortcuts” connect different arms of the swiss roll)

Mostly ad-hoc

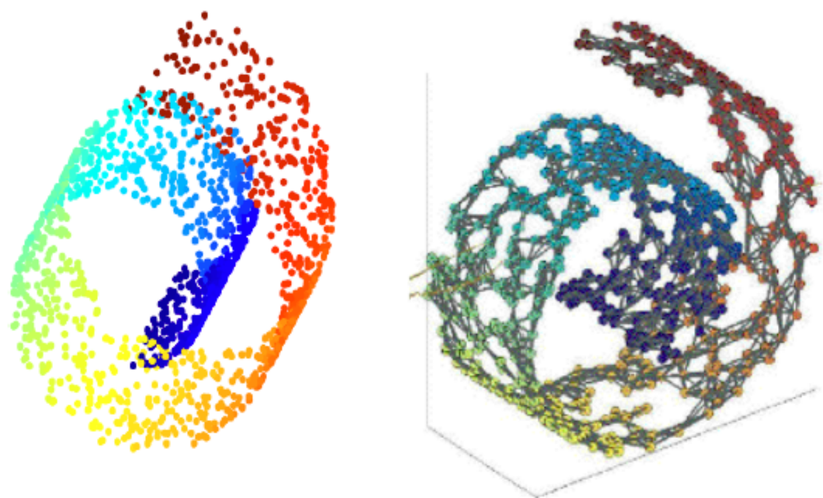


Step 2 – Projection using Graph

Original Representation		Transformed representation
data point		projections
x_i	$\rightarrow$	$(f_1(i), \ldots, f_d(i))$
(D-dimensional vector)		(d-dimensional vector)

Basic Idea: Find vector **f** such that, if x_i is close to x_j in the graph (i.e. W_{ij} is large), then projections of the points f(i) and f(j) are close

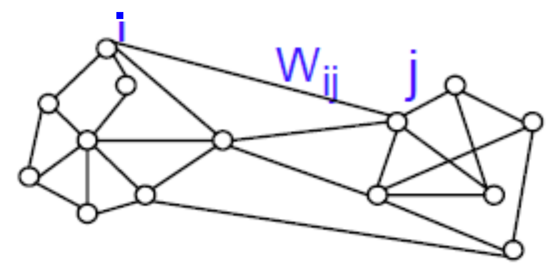
$$\min_{\mathbf{f}} \sum_{ij} W_{ij}(\mathbf{f}_i - \mathbf{f}_j)^2$$



Step 2 – Projection using Graph

- Graph Laplacian (unnormalized version)

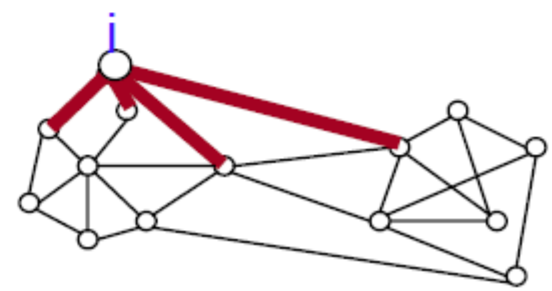
$$L = D - W \qquad \text{(n x n matrix)}$$



W – Weight matrix

D – Degree matrix = *diag*(d_1, …., d_n)

$d_i = \sum_j w_{ij}$ degree of a vertex



Note: If graph is connected,

1 is an eigenvector

with 0 as eigenvalue

$$\mathbf{L1} = \begin{bmatrix} d_1 - \sum_j w_{1j} \\ d_2 - \sum_j w_{2j} \\ \dots \\ d_n - \sum_j w_{nj} \end{bmatrix} = 0$$

Step 2 – Projection using Graph

- Justification – points connected on the graph stay as close as possible after embedding

$$\min_{\mathbf{f}} \sum_{ij} W_{ij}(\mathbf{f}_i - \mathbf{f}_j)^2 \equiv \min_{\mathbf{f}} \mathbf{f}^T \mathbf{L} \mathbf{f}$$

RHS = $f^T(D-W)\, f \;=\; f^T D\, f - f^T W\, f = \sum_i d_i f_i^2 - \sum_{i,j} f_i f_j w_{ij}$

$$= \frac{1}{2}\left(\sum_i \left(\sum_j w_{ij}\right) f_i^2 - 2\sum_{ij} f_i f_j w_{ij} + \sum_j \left(\sum_i w_{ij}\right) f_j^2\right)$$

$$= \frac{1}{2}\sum_{ij} w_{ij}(f_i - f_j)^2$$ **= LHS**

Step 2 – Projection using Graph

- Justification – points connected on the graph stay as close as possible after embedding

$$\min_{\mathbf{f}} \sum_{ij} W_{ij}(\mathbf{f}_i - \mathbf{f}_j)^2 \equiv \min_{\mathbf{f}} \mathbf{f}^T \mathbf{L} \mathbf{f} \qquad s.t.\ \mathbf{f}^T \mathbf{f} = 1$$

Similar to PCA with XX^T replaced by L

Lagrangian: $\min_{\mathbf{f}} \mathbf{f}^{\mathbf{T}} \mathbf{L} \mathbf{f} - \lambda \mathbf{f}^T \mathbf{f}$

Wrap constraint into the objective function

$\partial / \partial \mathbf{f} = 0$ $\qquad (\mathbf{L} - \lambda \mathbf{I})\mathbf{f} = 0$ $\qquad \boxed{\mathbf{L}\mathbf{f} = \lambda \mathbf{f}}$

Step 2 – Projection using Graph Laplacian

- Graph Laplacian (unnormalized version)

  $L = D - W$ (n x n matrix)

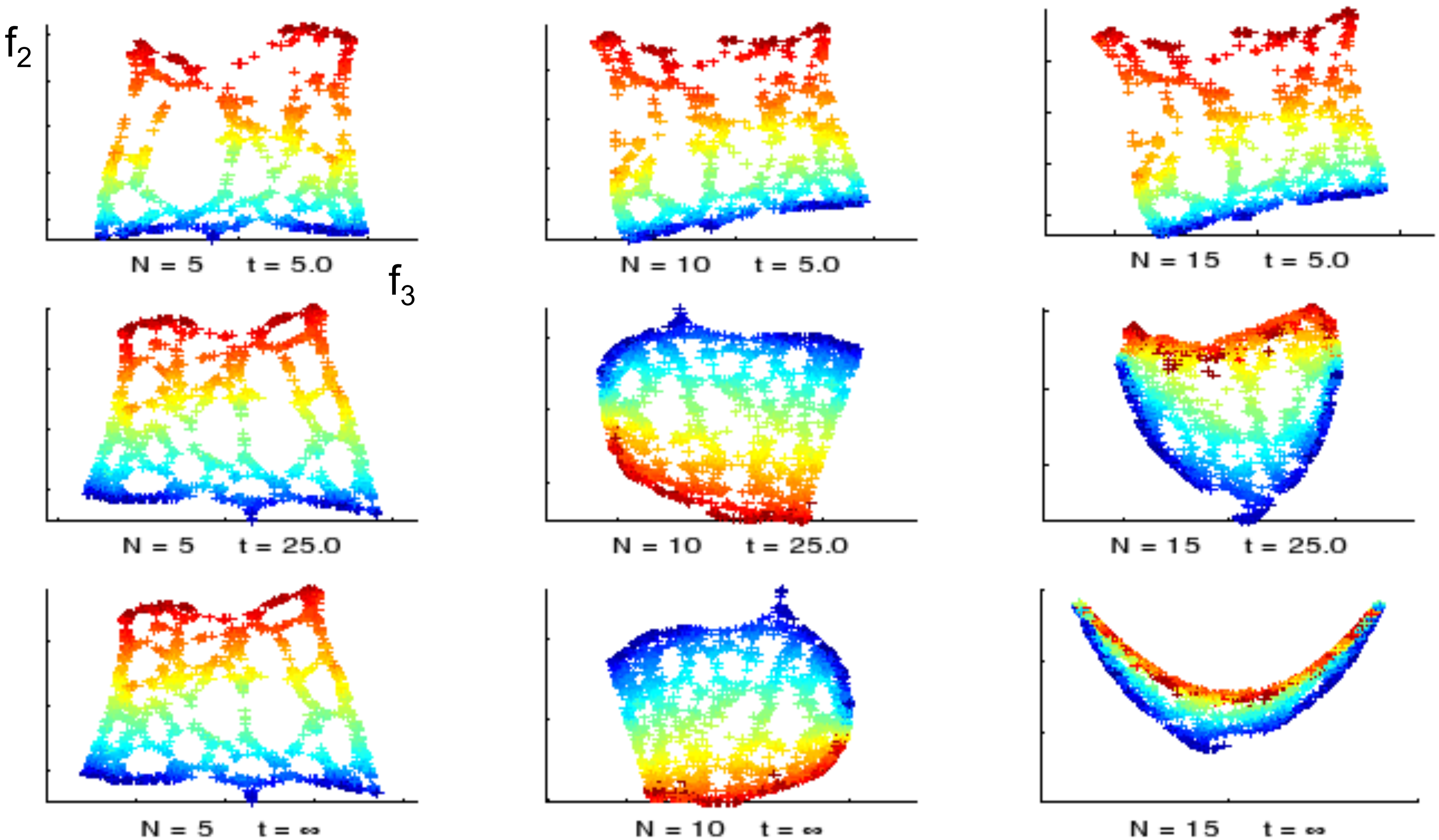
  Find eigenvectors of the graph Laplacian $\boxed{\mathbf{L}\mathbf{f} = \lambda \mathbf{f}}$

  Ordered eigenvalues $0 = \lambda_1 \leq \lambda_2 \leq \lambda_3 \leq \ldots \leq \lambda_n$

  To embed data points in d-dim space, project data points onto eigenvectors associated with $\lambda_1, \lambda_2, \ldots, \lambda_d$

Original Representation		Transformed representation
data point		projections
x_i	→	$(f_1(i), \ldots, f_d(i))$
(D-dimensional vector)		(d-dimensional vector)

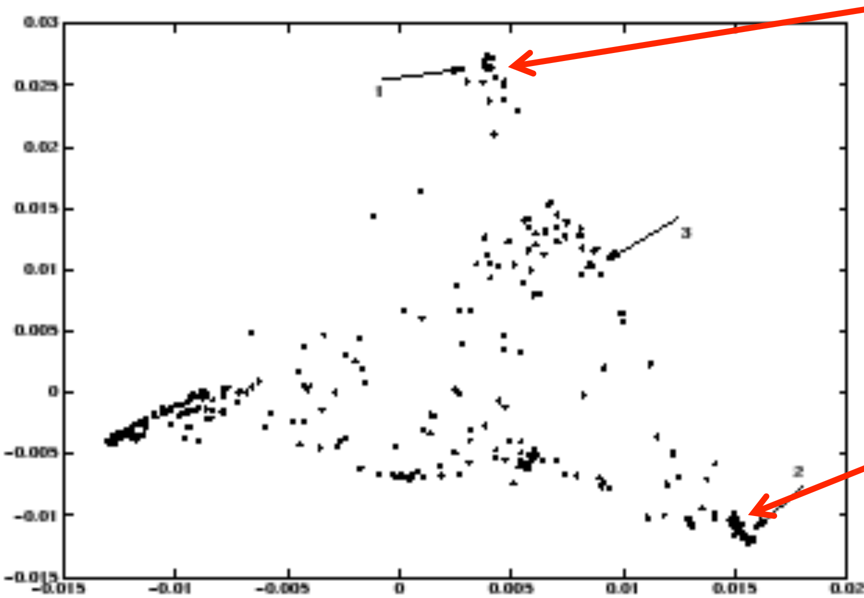
Unrolling the swiss roll

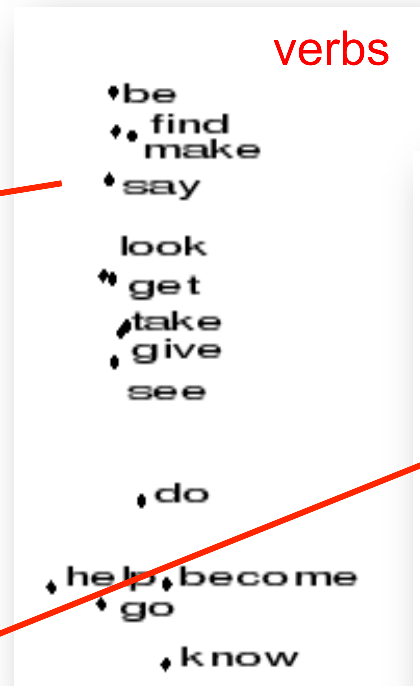


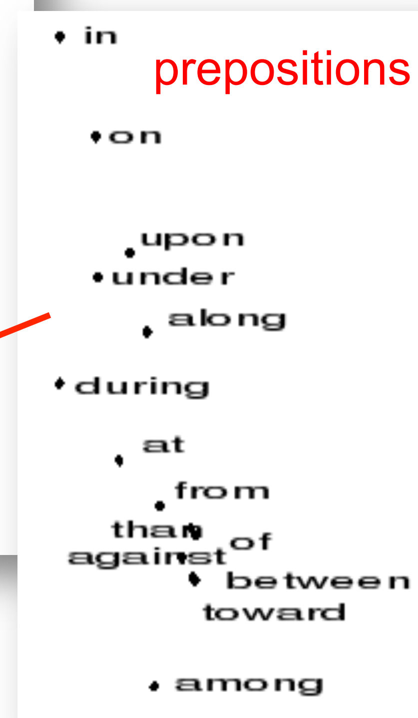
N=number of nearest neighbors, t = the heat kernel parameter (Belkin & Niyogi'03)

Example – Understanding syntactic structure of words

- 300 most frequent words of Brown corpus
- Information about the frequency of its left and right neighbors (600 Dimensional space.)
- The algorithm run with N = 14, t = 1







PCA vs. Laplacian Eigenmaps

PCA	Laplacian Eigenmaps
Linear embedding	Nonlinear embedding
based on largest eigenvectors of $D \times D$ correlation matrix $\Sigma = XX^T$ between features	based on smallest eigenvectors of $n \times n$ Laplacian matrix $L = D - W$ between data points
eigenvectors give latent features - to get embedding of points, project them onto the latent features	eigenvectors directly give embedding of data points
$\underset{D \times 1}{x_i} \rightarrow \underset{d \times 1}{[v_1^T x_i, v_2^T x_i, \ldots v_d^T x_i]}$	$\underset{D \times 1}{x_i} \rightarrow \underset{d \times 1}{[f_1(i), \ldots, f_d(i)]}$

Dimensionality Reduction Methods

- Feature Selection - Only a few features are relevant to the learning task

    Score features (mutual information, prediction accuracy, domain knowledge)
    Regularization

- Latent features – Some linear/nonlinear combination of features provides a more efficient representation than observed feature

    Linear: Low-dimensional linear subspace projection

        PCA (Principal Component Analysis),
        MDS (Multi Dimensional Scaling),
        Factor Analysis, ICA (Independent Component Analysis)

    Nonlinear: Low-dimensional nonlinear projection that preserves local information along the manifold

        Laplacian Eigenmaps
        ISOMAP, Kernel PCA, LLE (Local Linear Embedding),
        Many, many more …

Spectral Clustering

Laplacian Eigenmaps

- Construct graph
- Compute graph Laplacian L = D – W
- Embed points using graph Laplacian $\hat{x}_i = [f_1(i), \ldots, f_d(i)]$

Spectral Clustering

- Run k-means on the embedded points $\{\hat{x}_i\}^n_{i=1}$

K-Means

Algorithm

Input – Desired number of clusters, *k*

Initialize – the *k* cluster centers (randomly if necessary)

Iterate –

1. Assign the objects to the nearest cluster centers

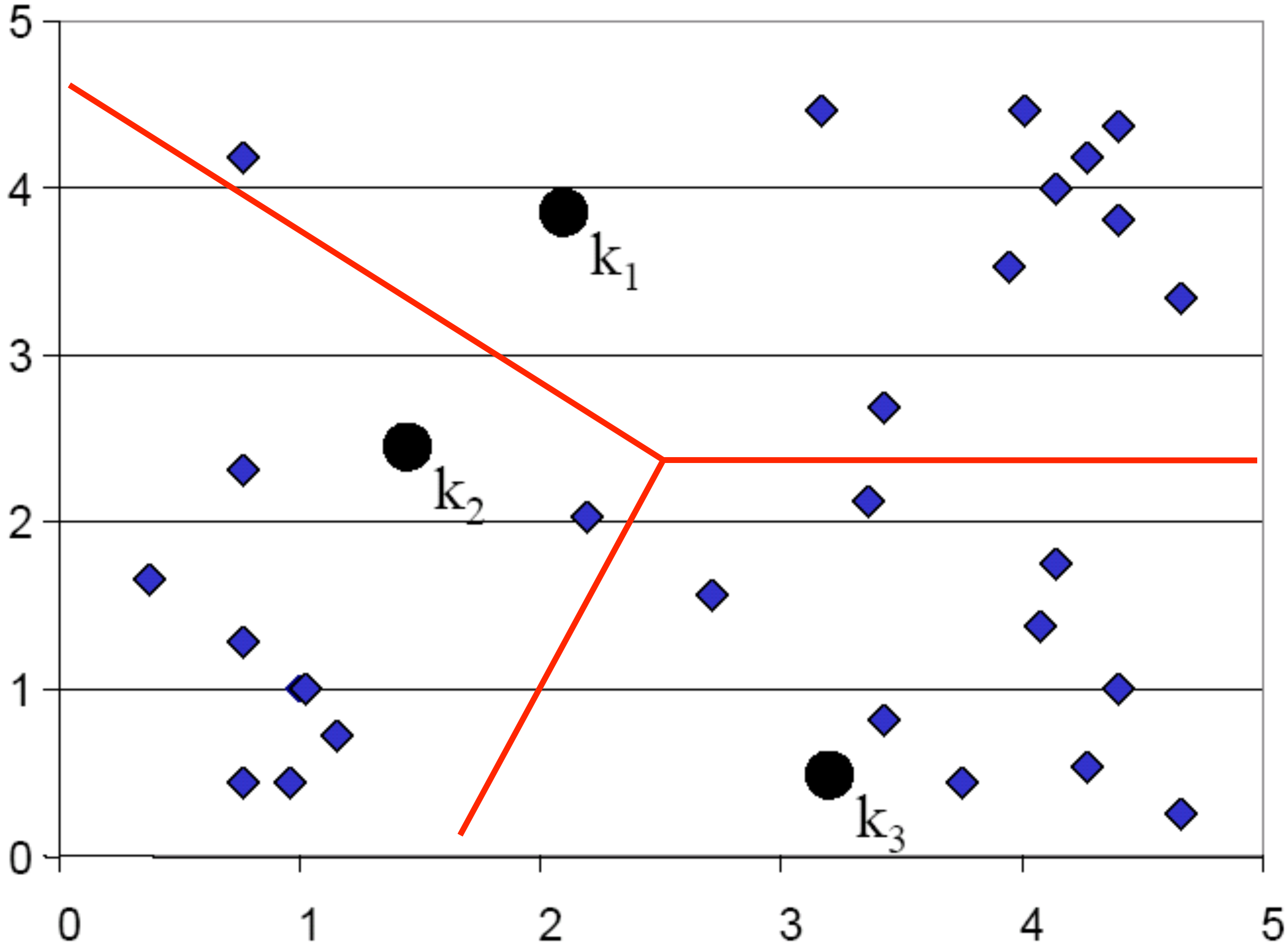
2. Re-estimate the *k* cluster centers (aka the centroid or mean) based on current assignment

$$\vec{\mu}_k = \frac{1}{\mathcal{C}_k} \sum_{i \in \mathcal{C}_k} \vec{x}_i$$

Termination –

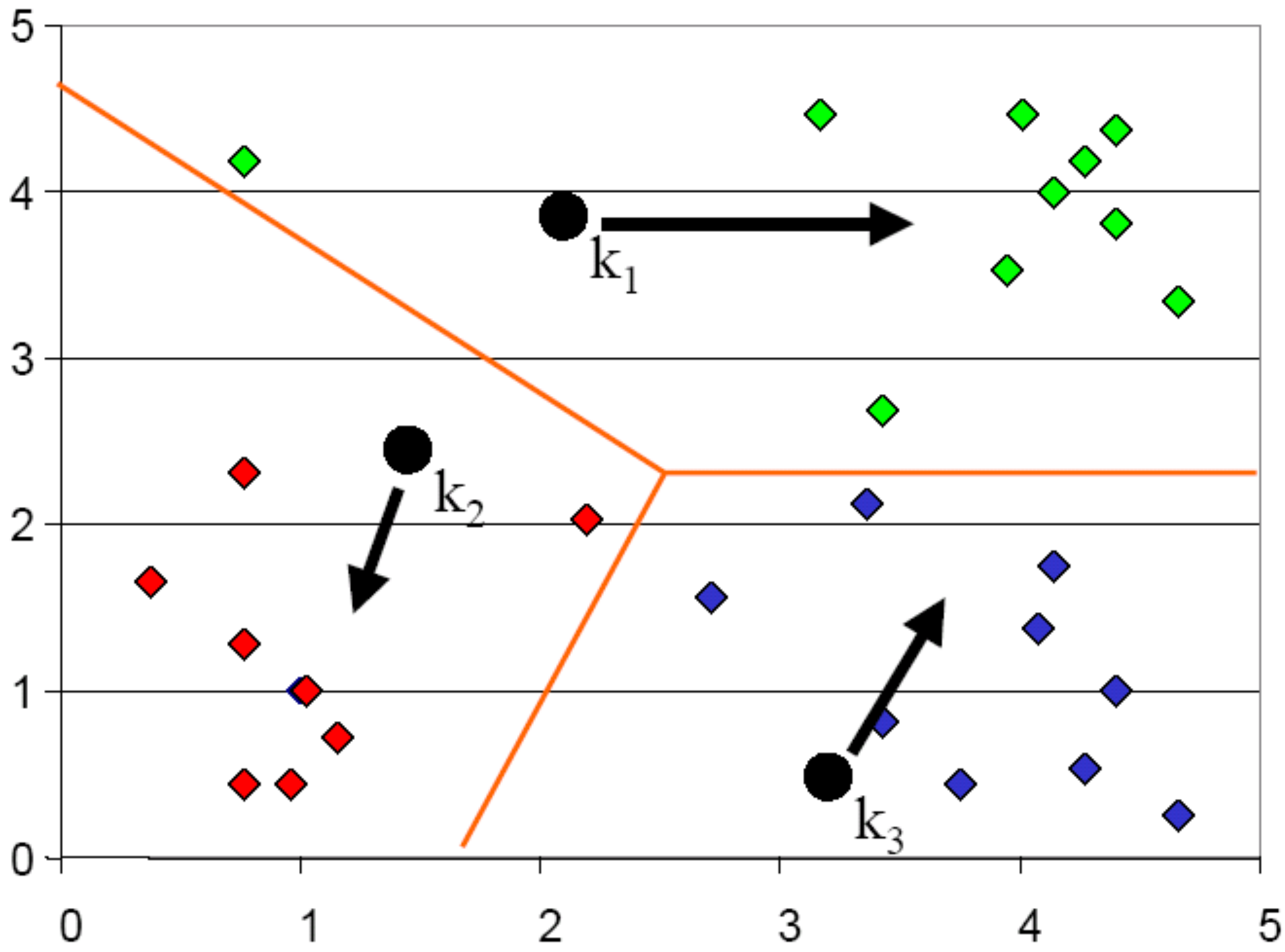
If none of the assignments changed in the last iteration, exit. Otherwise go to 1.

K-means Clustering: Step 1

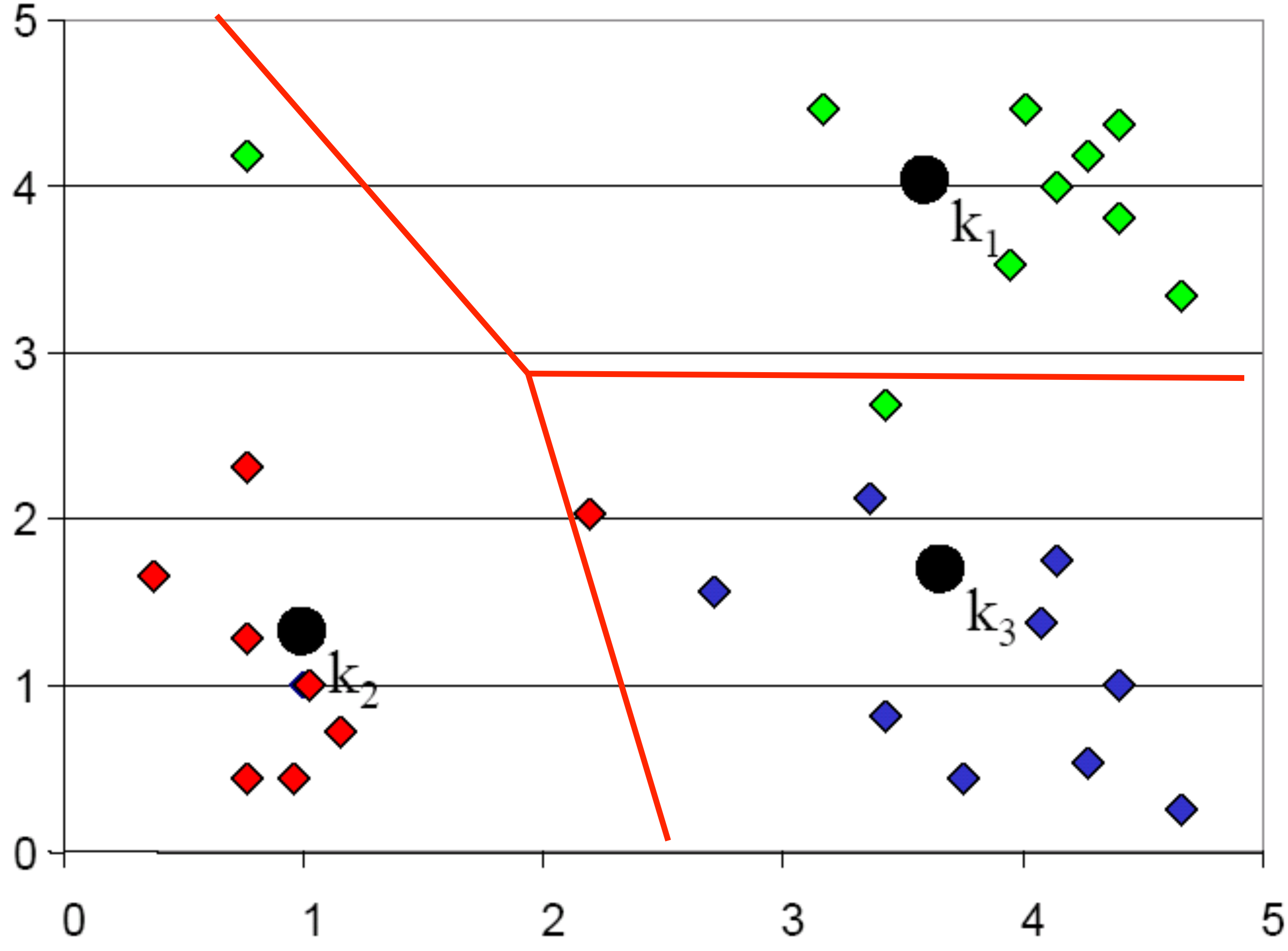


Voronoi diagram

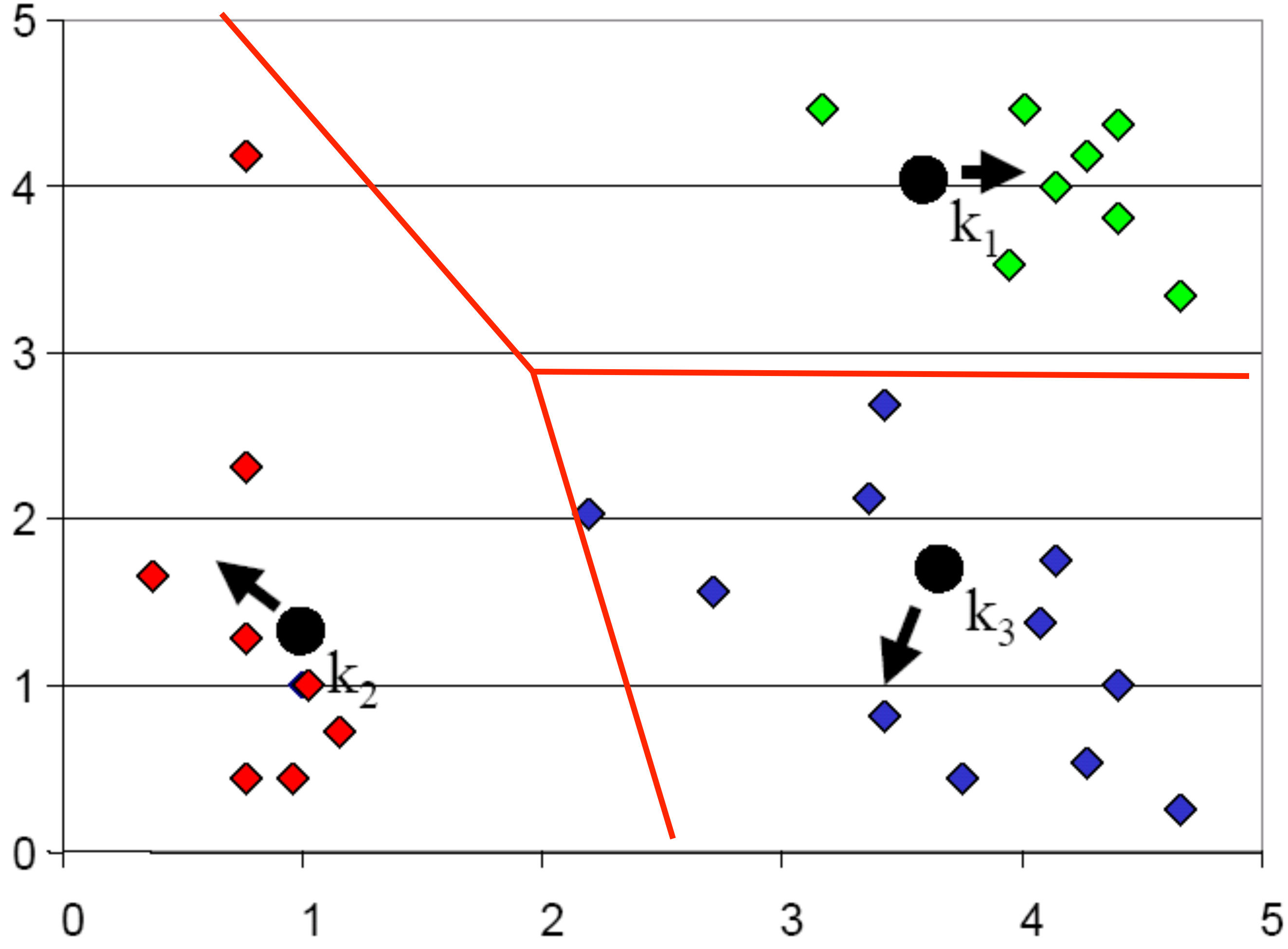
K-means Clustering: Step 2



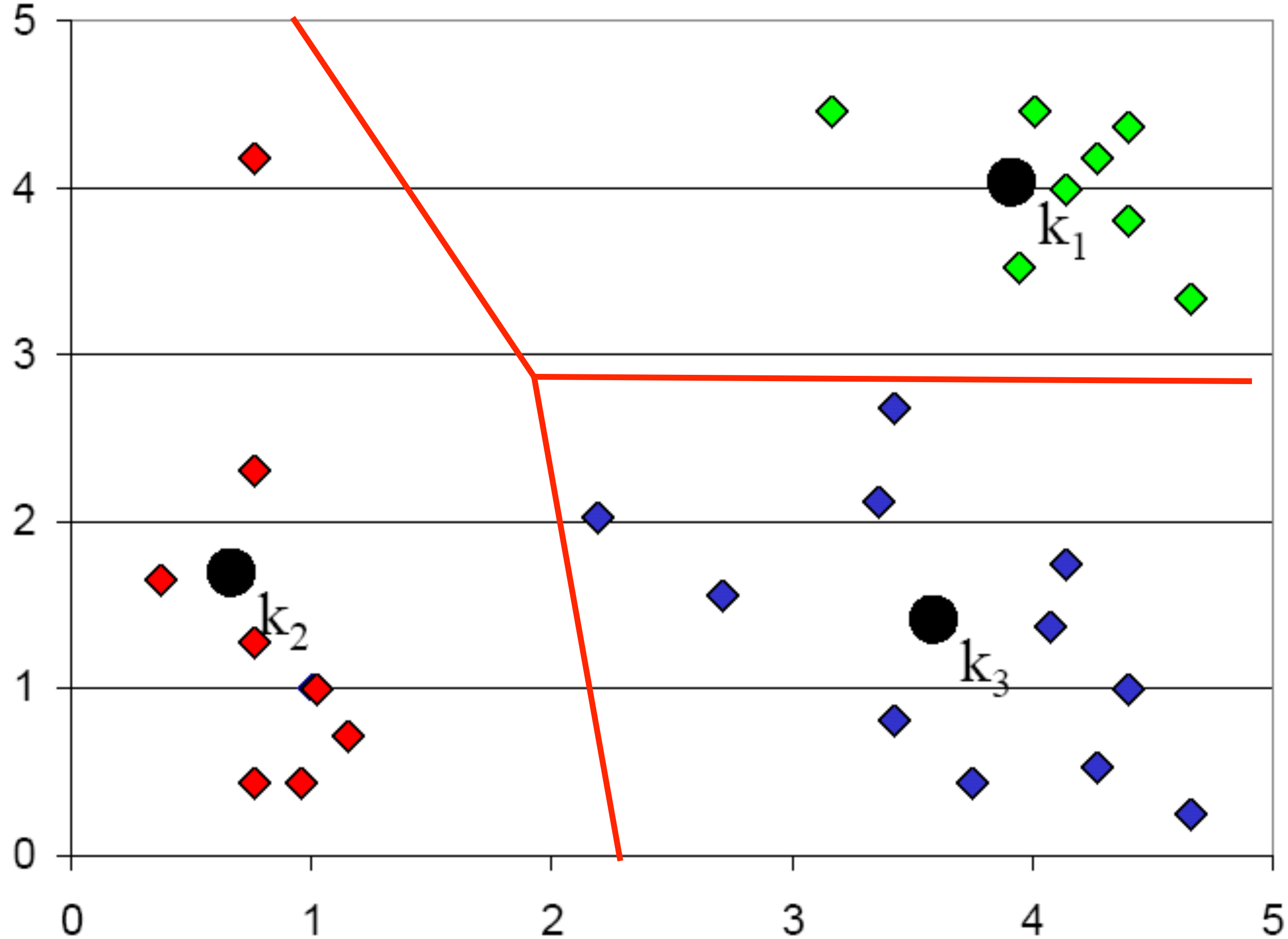
K-means Clustering: Step 3



K-means Clustering: Step 4

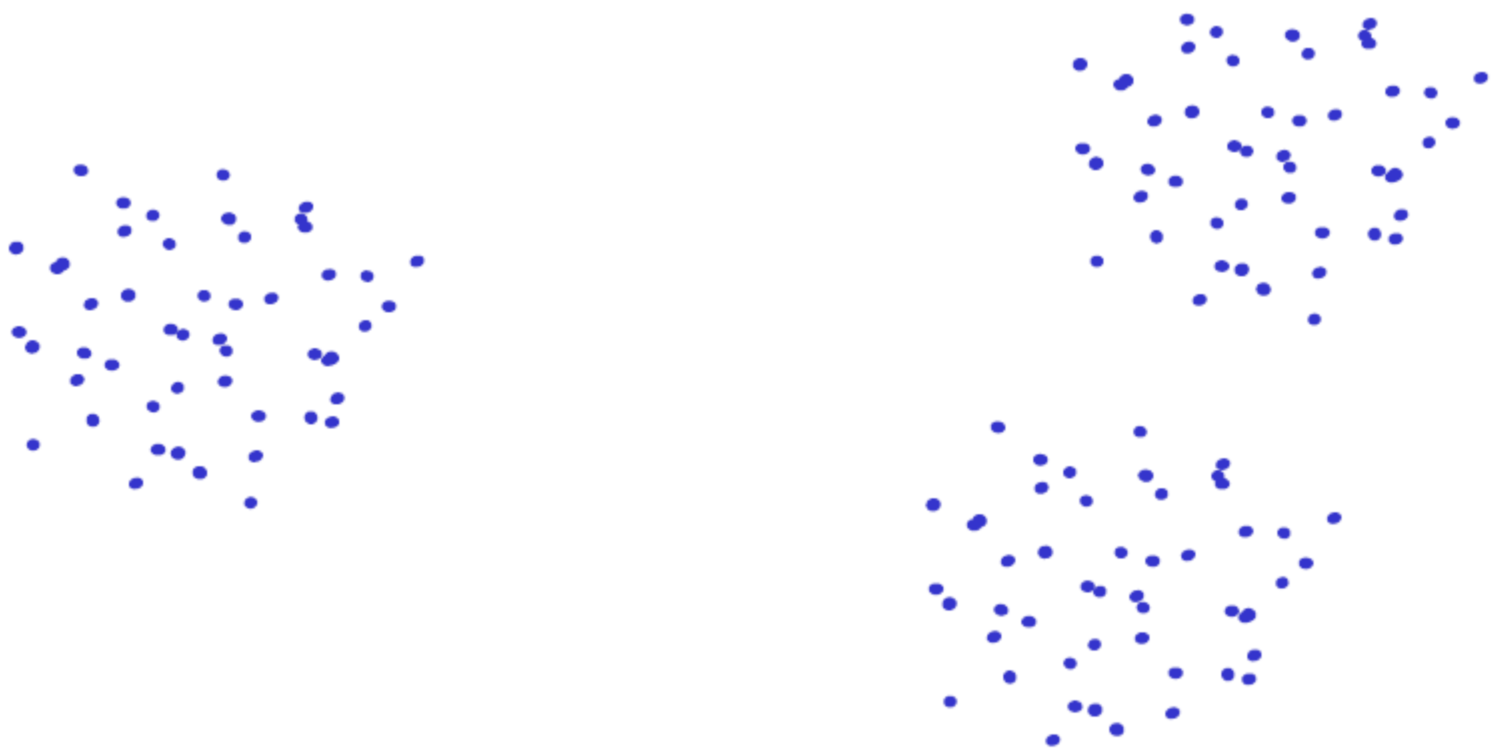


K-means Clustering: Step 5



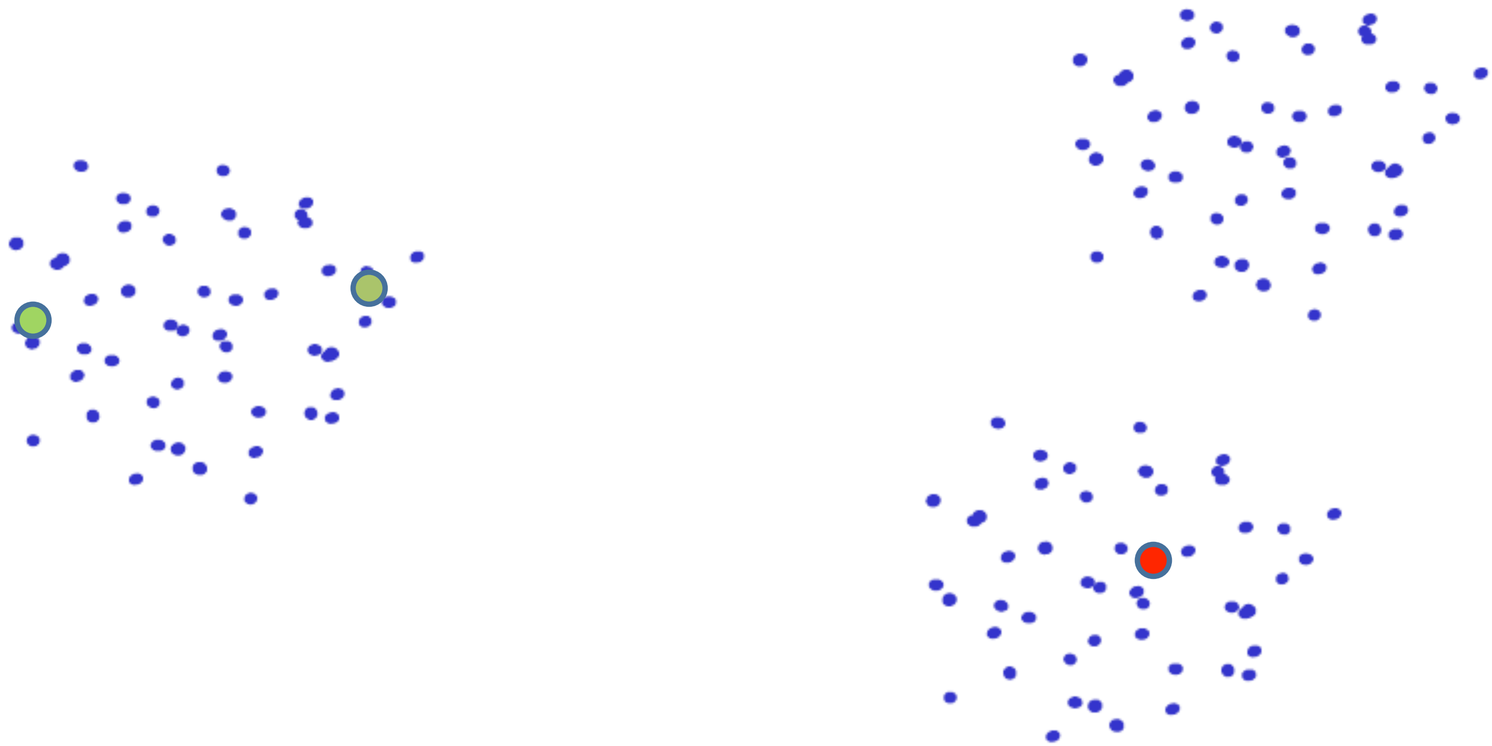
Seed Choice

- Results are quite sensitive to seed selection.



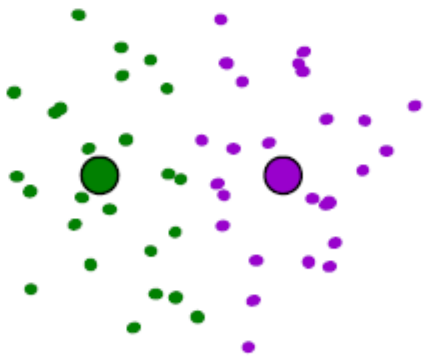
Seed Choice

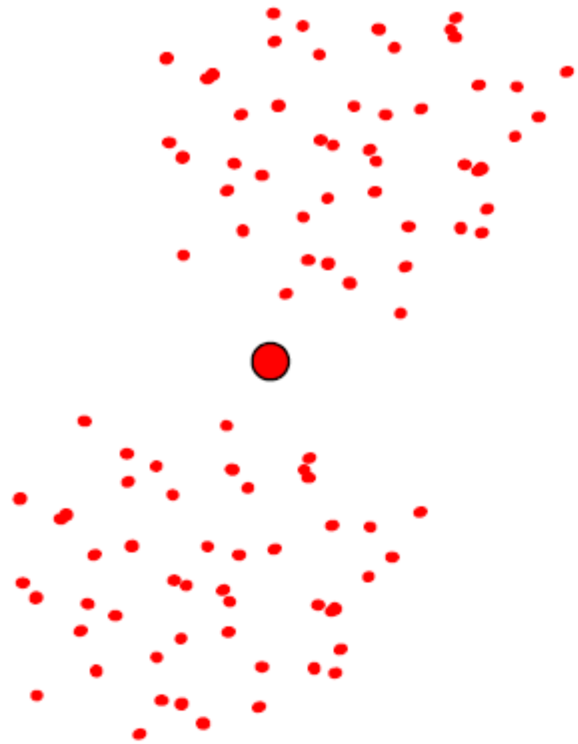
- Results are quite sensitive to seed selection.



Seed Choice

- Results are quite sensitive to seed selection.

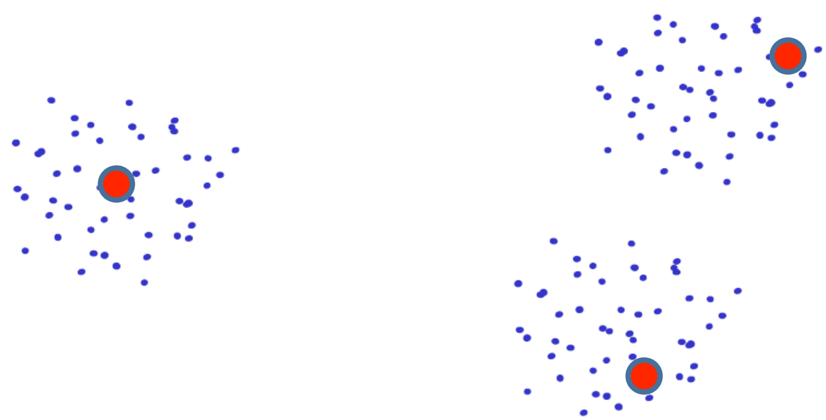




Seed Choice

- Results can vary based on random seed selection.

- Some seeds can result in poor convergence rate, or convergence to sub-optimal clustering.
  - Try out multiple starting points (very important!!!)
  - Initialize with the results of another method.
  - k-means ++ algorithm of Arthur and Vassilvitskii

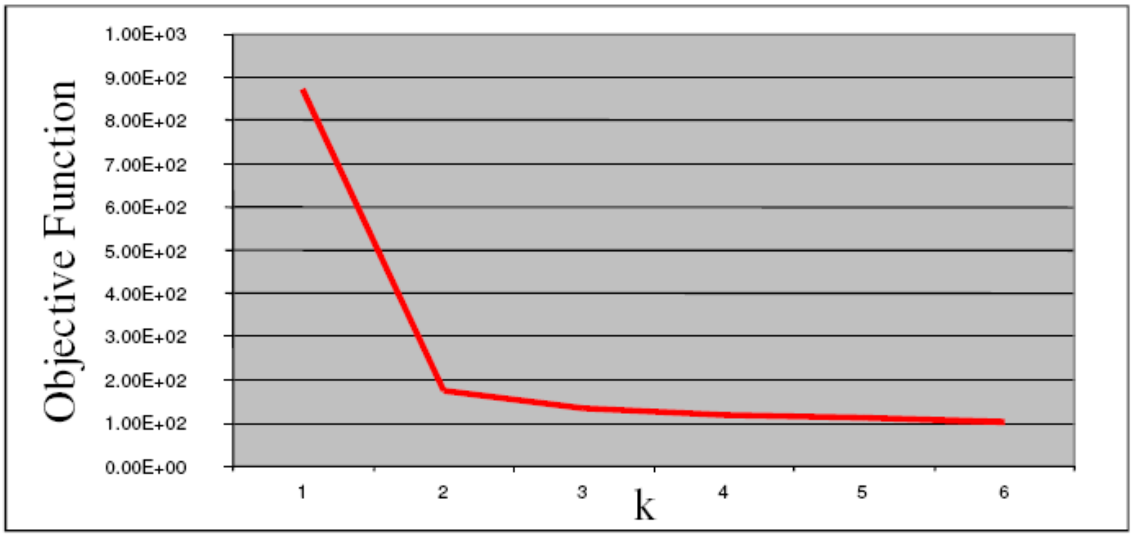


Other Issues

- Number of clusters K
  - Objective function

$$\sum_{j=1}^{m} ||\mu_{C(j)} - x_j||^2$$

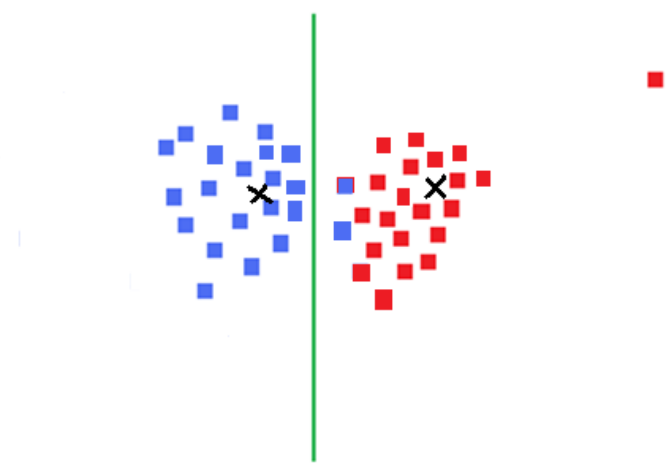
  - Look for "Knee" in objective function

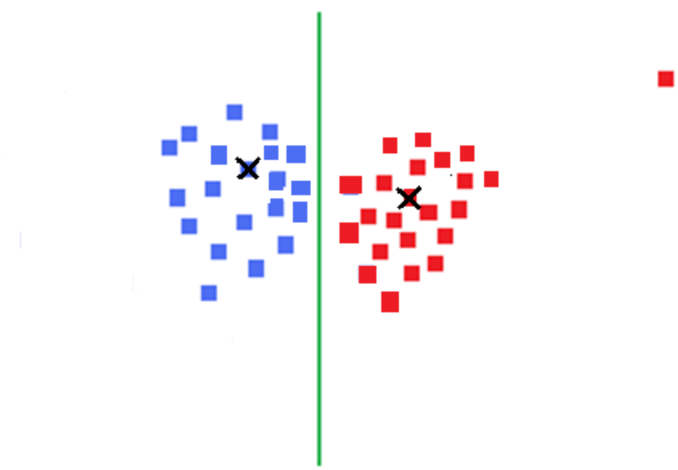


  - Can you pick K by minimizing the objective over K? NO!

Other Issues

- Sensitive to Outliers
  - use K-medoids

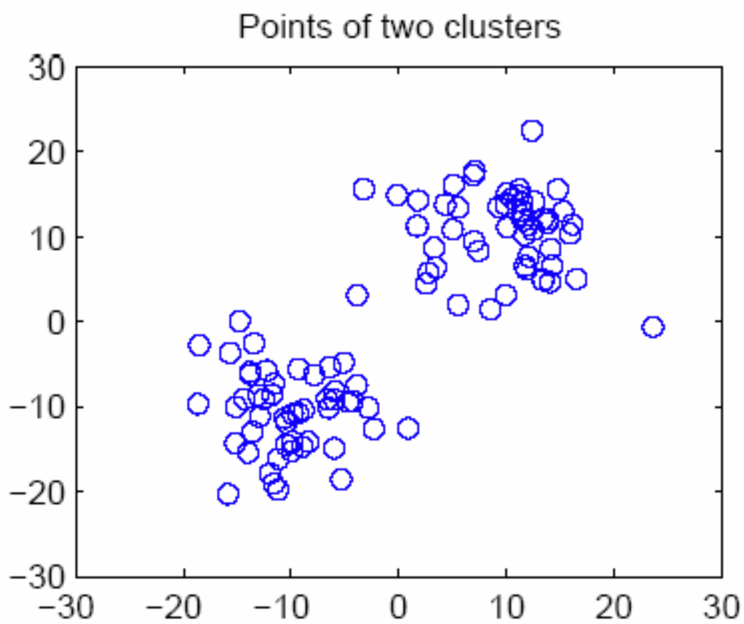




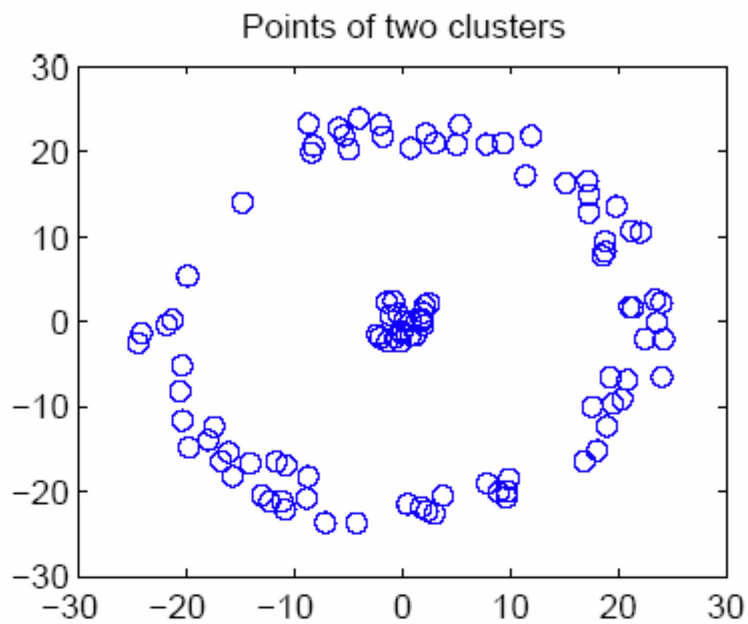
- Shape of clusters
  - Assumes isotopic, convex clusters

k-means vs Spectral clustering

Applying k-means to laplacian eigenvectors allows us to find cluster with non-convex boundaries.



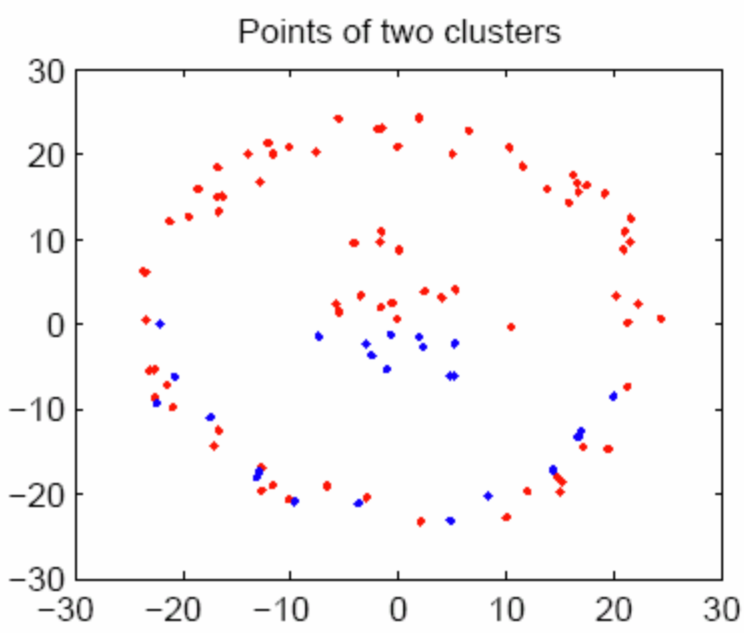
Both perform same



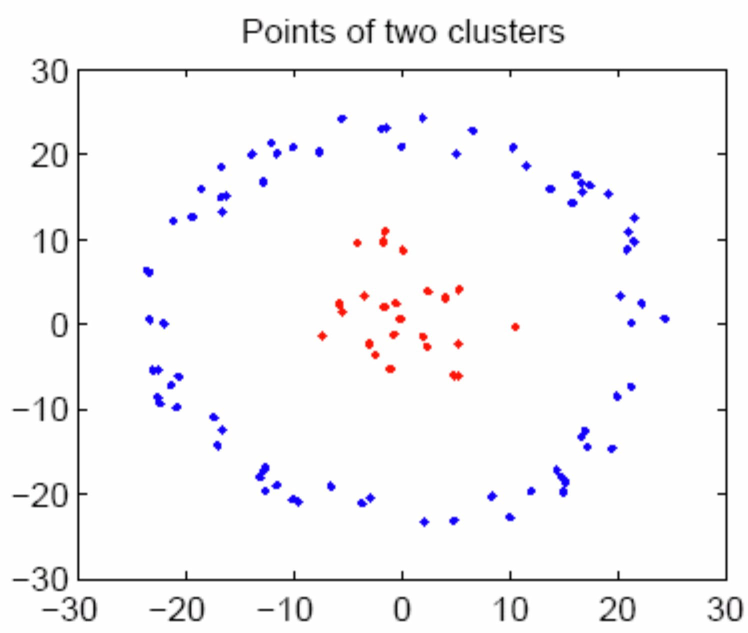
Spectral clustering is superior

k-means vs Spectral clustering

Applying k-means to laplacian eigenvectors allows us to find cluster with non-convex boundaries.



k-means output

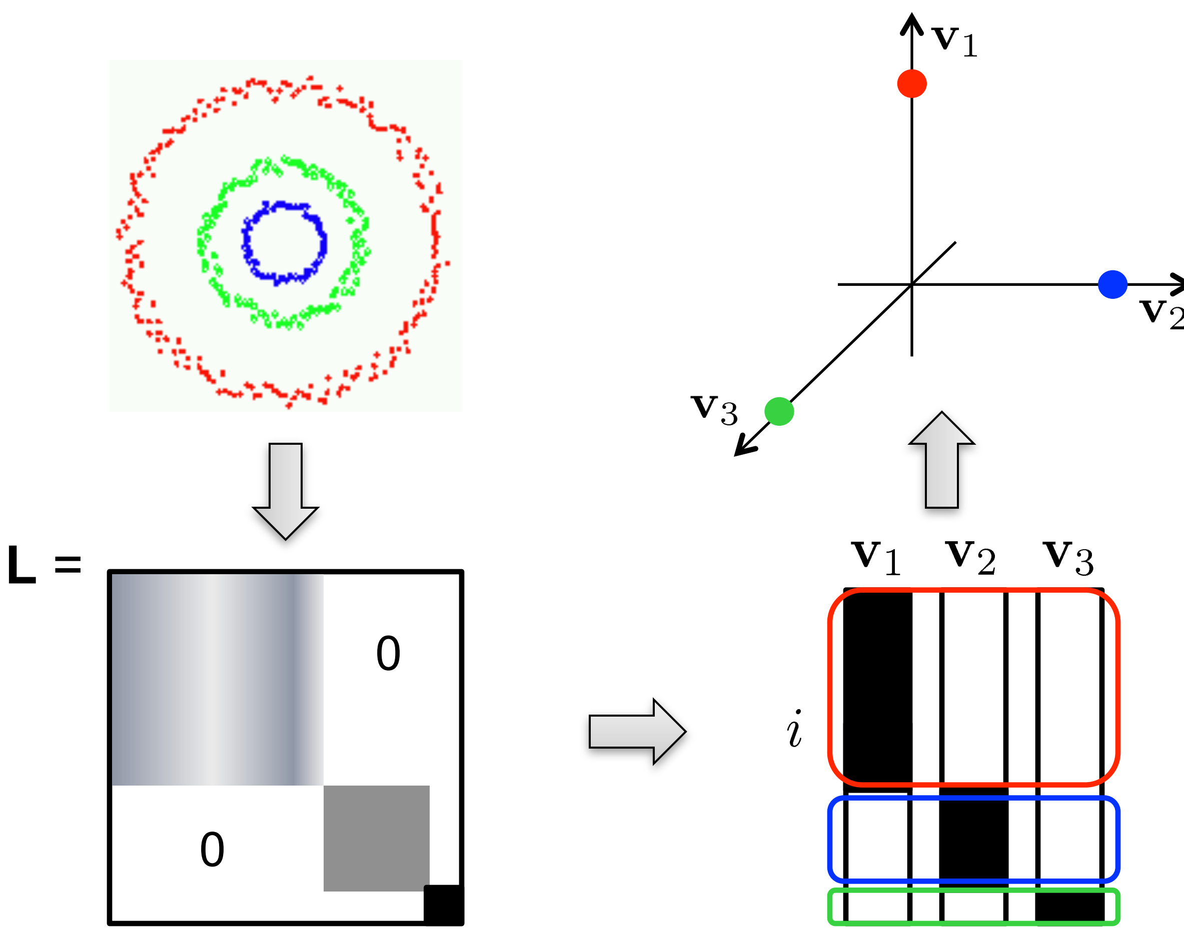


Spectral clustering output

Spectral Clustering – Intuition

If graph is appropriately constructed, results in disconnected subgraphs

Laplacian eigenvectors are constant on connected subgraphs



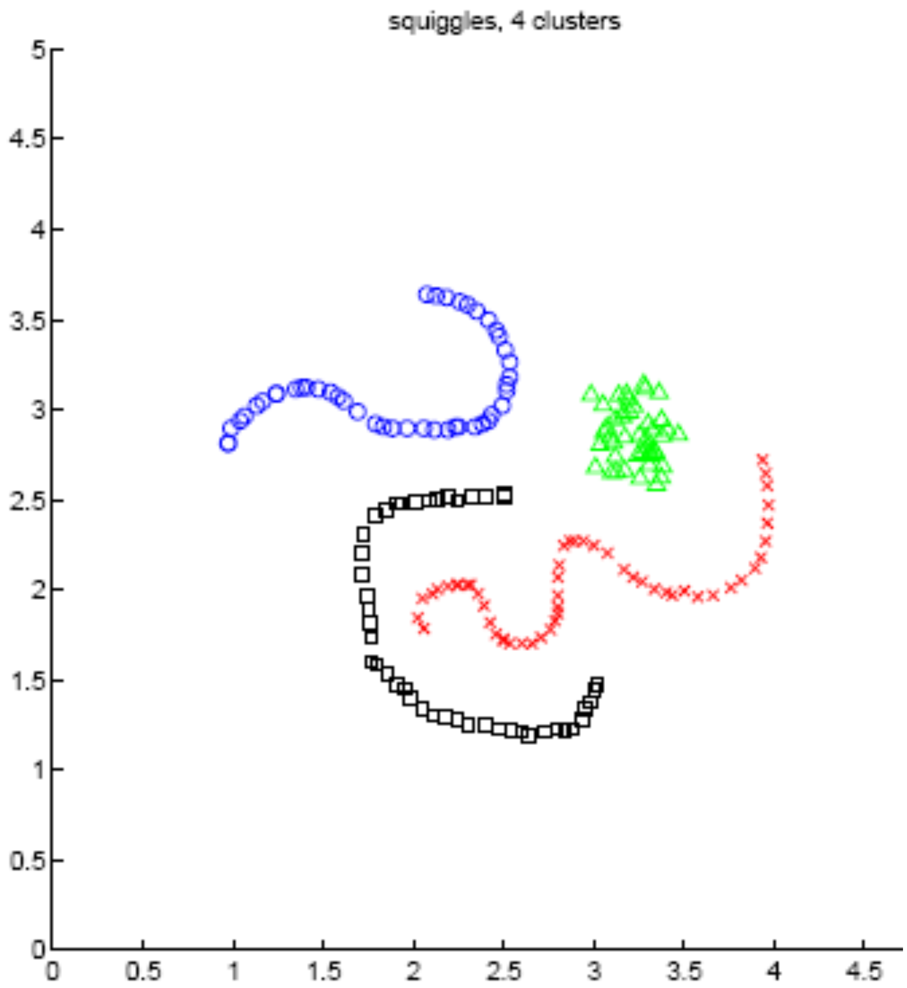
Points are easy to cluster in embedded space e.g. using k-means

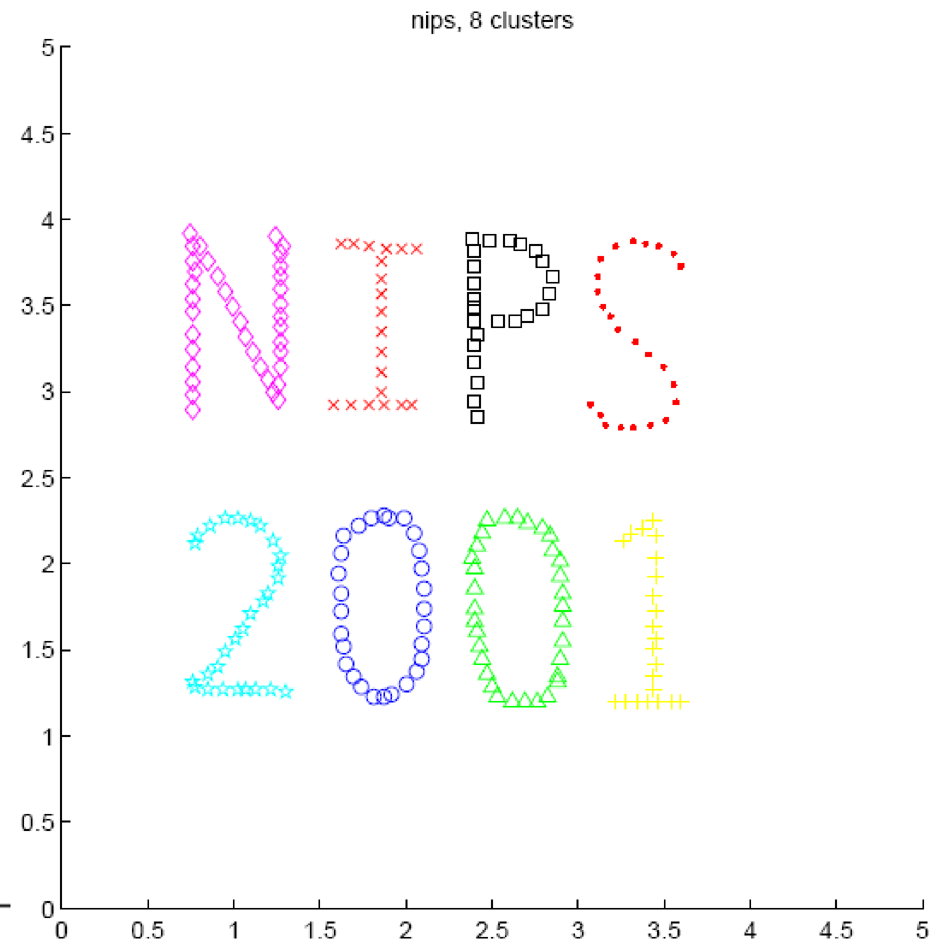
Embedding of point i

$$[\mathbf{v}_1(i), \mathbf{v}_2(i), \mathbf{v}_3(i)]$$

Examples

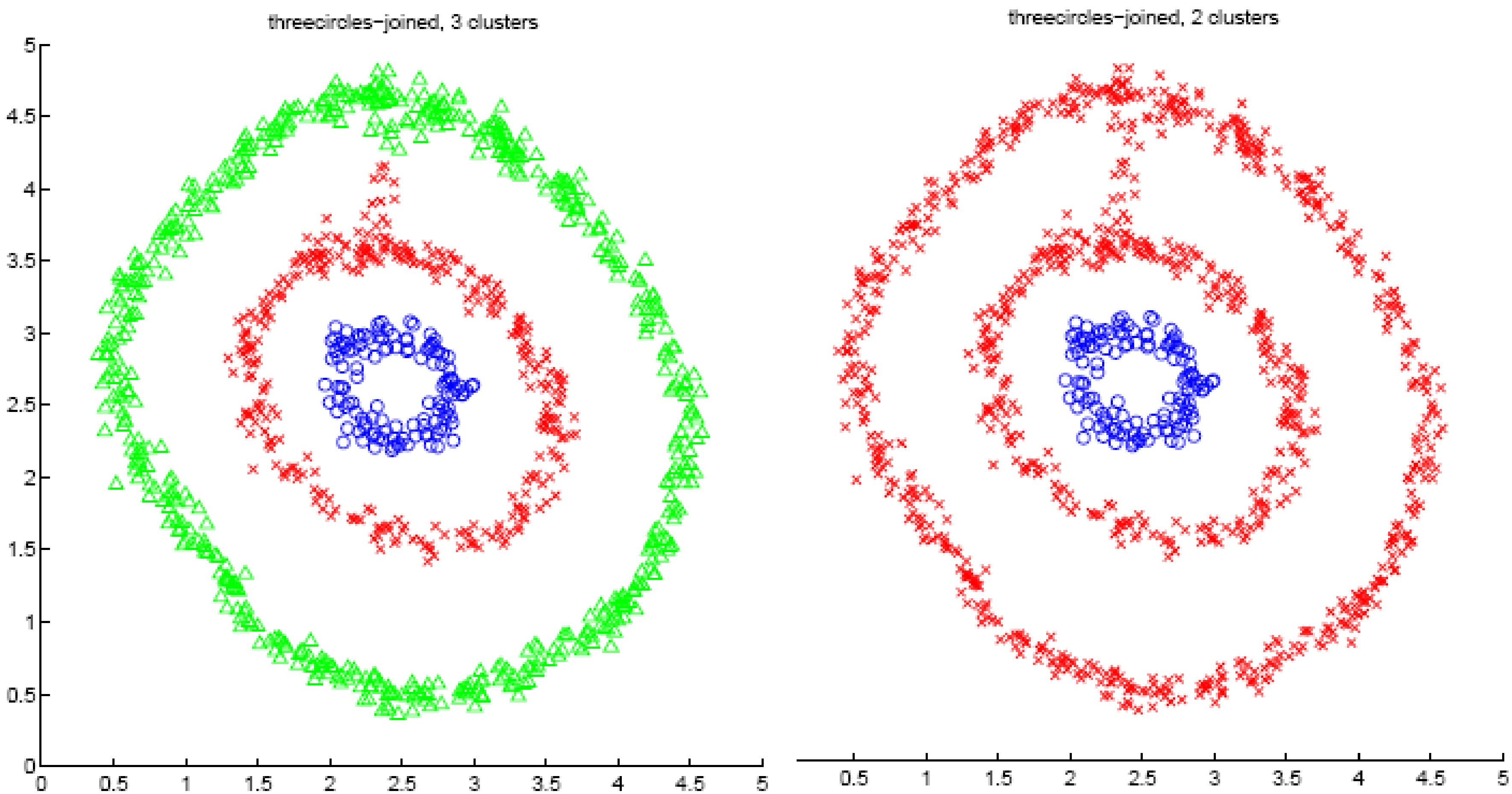
Ng et al 2001





Examples (Choice of k)

Ng et al 2001



Some Issues

- Choice of parameters (ε,k,σ) used in constructing graph
- Choice of number of clusters k

  Most stable clustering is usually given by the value of k that maximizes the eigengap (difference between consecutive eigenvalues)

$$\Delta_k = \left|\lambda_k - \lambda_{k-1}\right|$$