Unsupervised Learning

Instructor: Steven C.H. Hoi

School of Computer Engineering

Singapore Management University

Email: chhoi@smu.edu.sg

Outline

- Clustering
  - K-means algorithm for clustering
  - Expectation Maximization (EM) algorithm for clustering
- Dimension Reduction
  - Linear method: PCA
  - Non-linear method: LLE

Unsupervised Learning

- Supervised learning for prediction: $\mathbf{x} \longrightarrow \mathbf{y}$
  classification, regression
  - Applications: face recognition, speech recognition, etc
  - Techniques: SVM, logistic regressions, etc

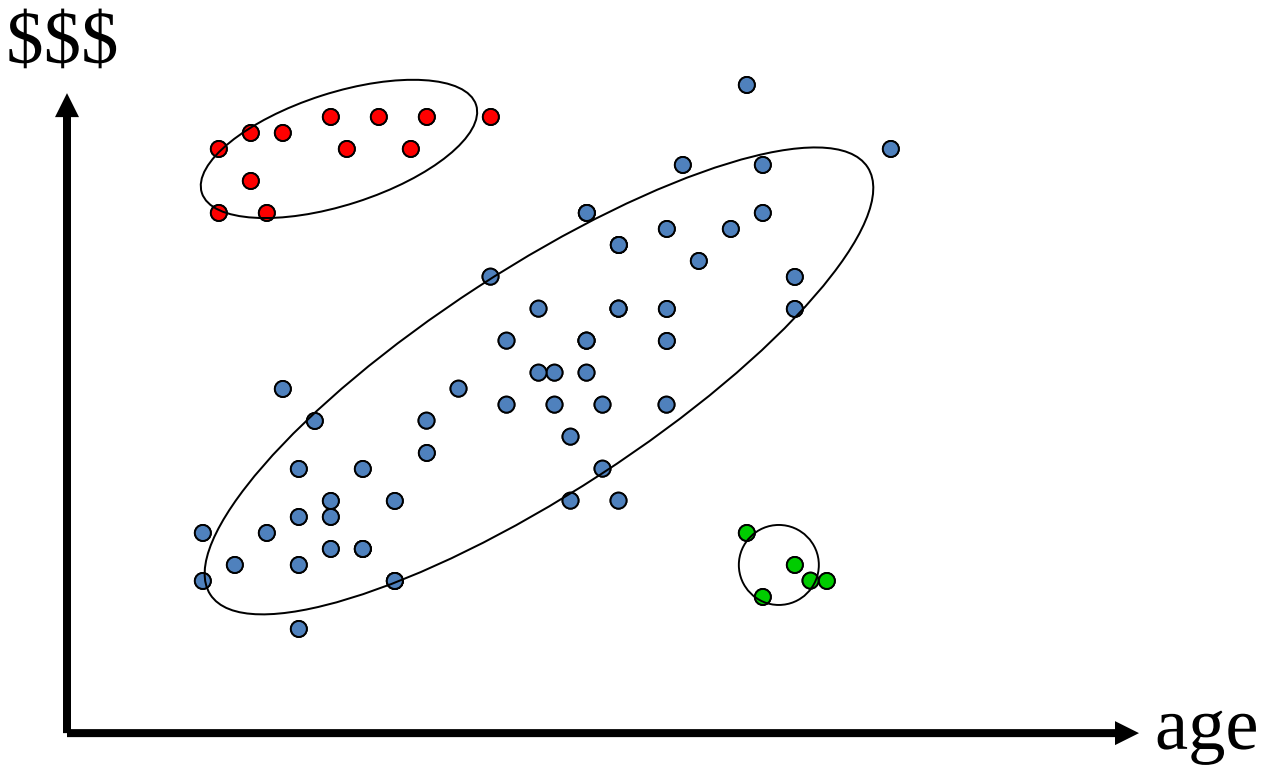
- Unsupervised learning for structure discovery:

$$\mathbf{x} \longrightarrow \mathbf{z}$$

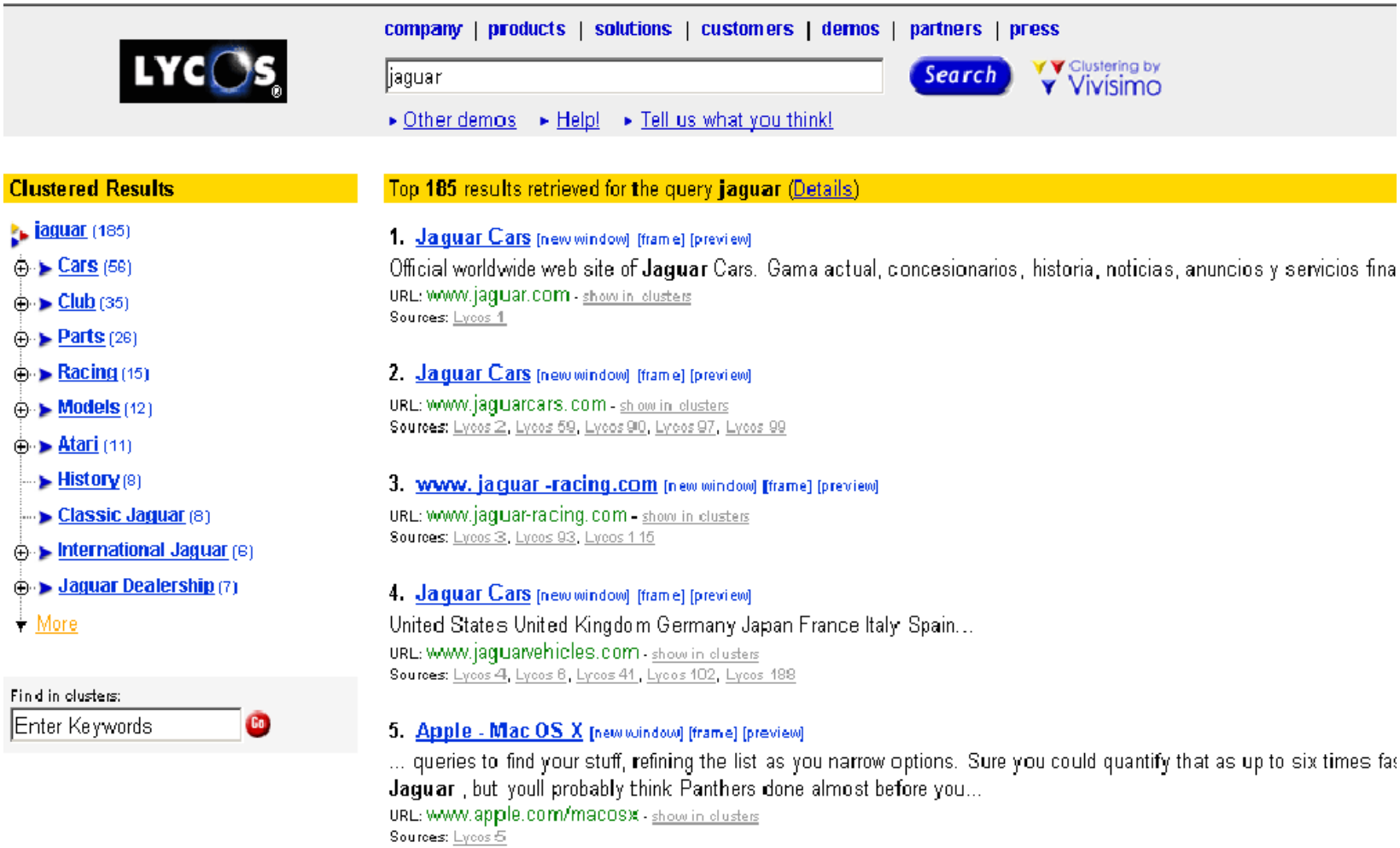
  Find an alternative representation $\mathbf{z}$ of data $\mathbf{x}$
  - Applications: visualization, compression, etc
  - Techniques: clustering, dimensionality reduction, sparse coding, representation learning, etc

Clustering

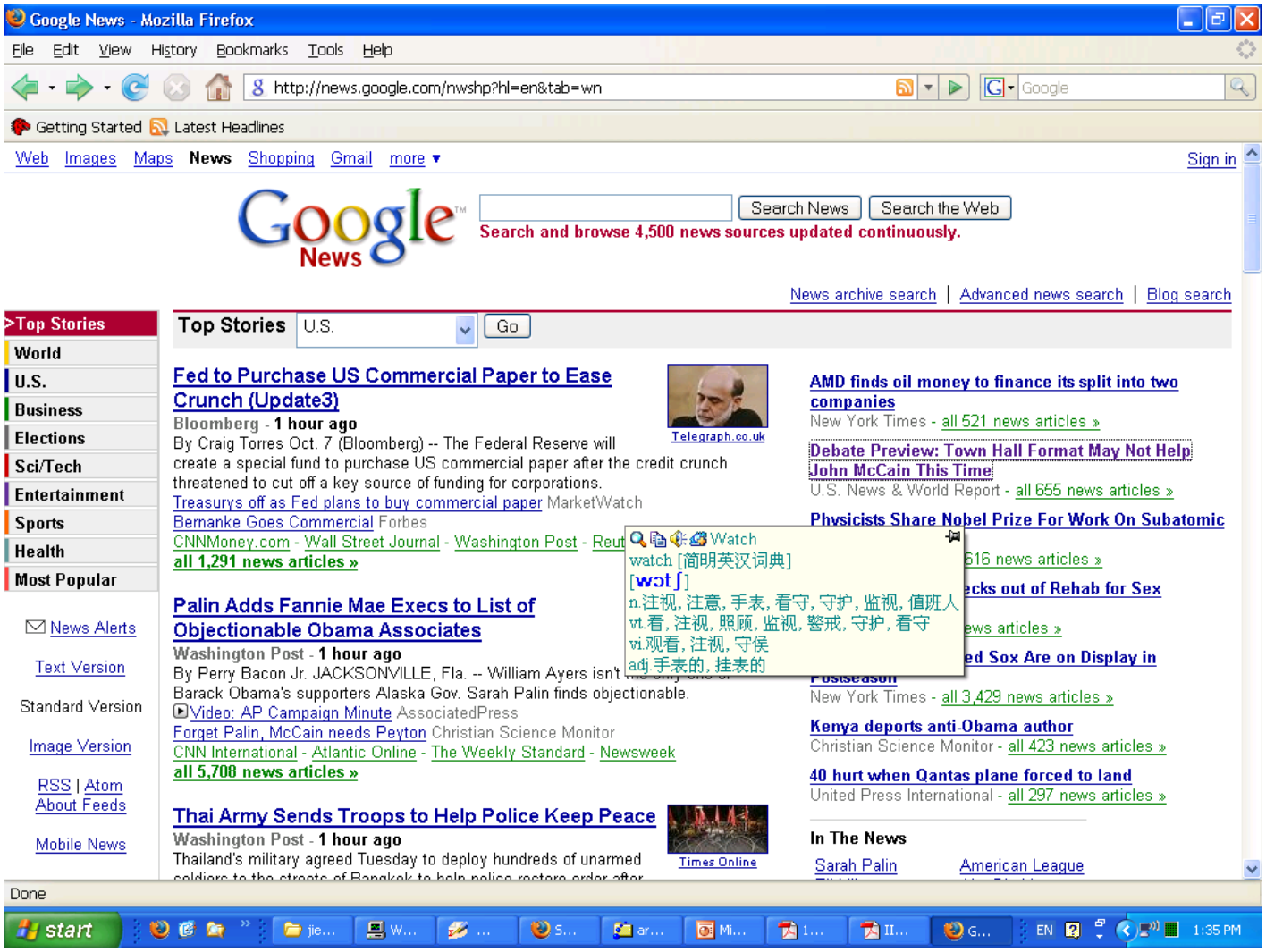
- Find out underlying structure for given data
- Unsupervised learning: no label info available



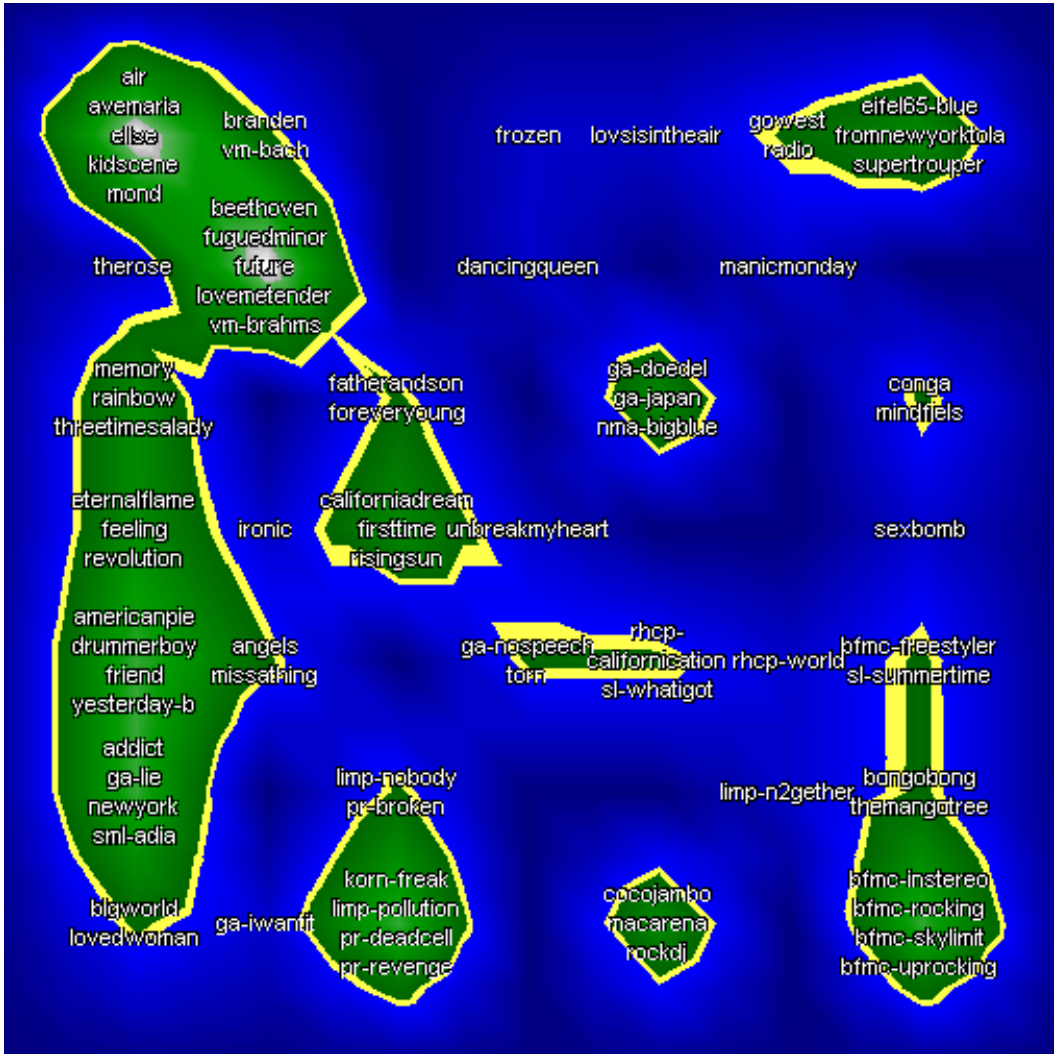
Application: Search Result Clustering



Application: Google News

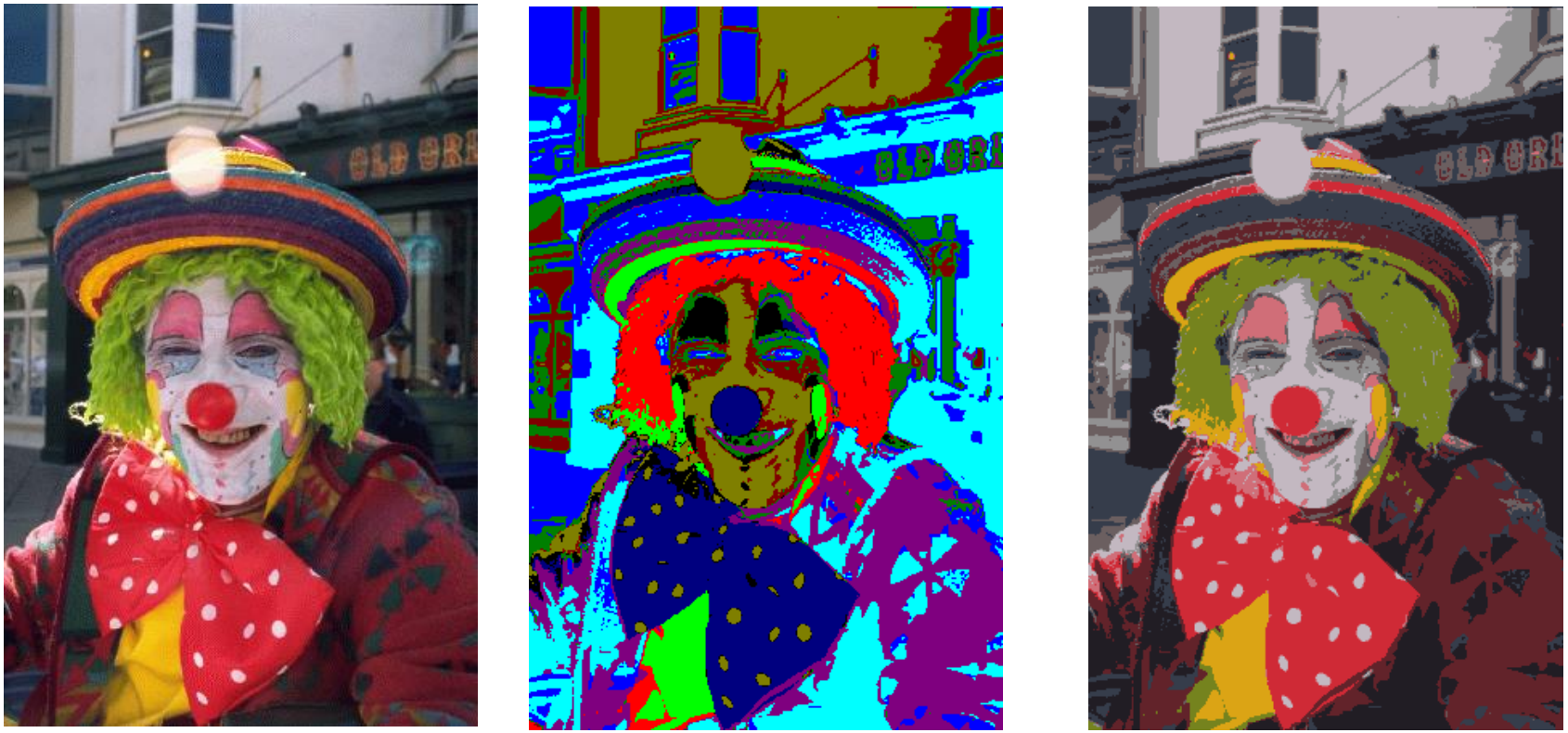


Application: Visualization



Islands of music
(Pampalk et al., KDD' 03)

Application: Image Compression



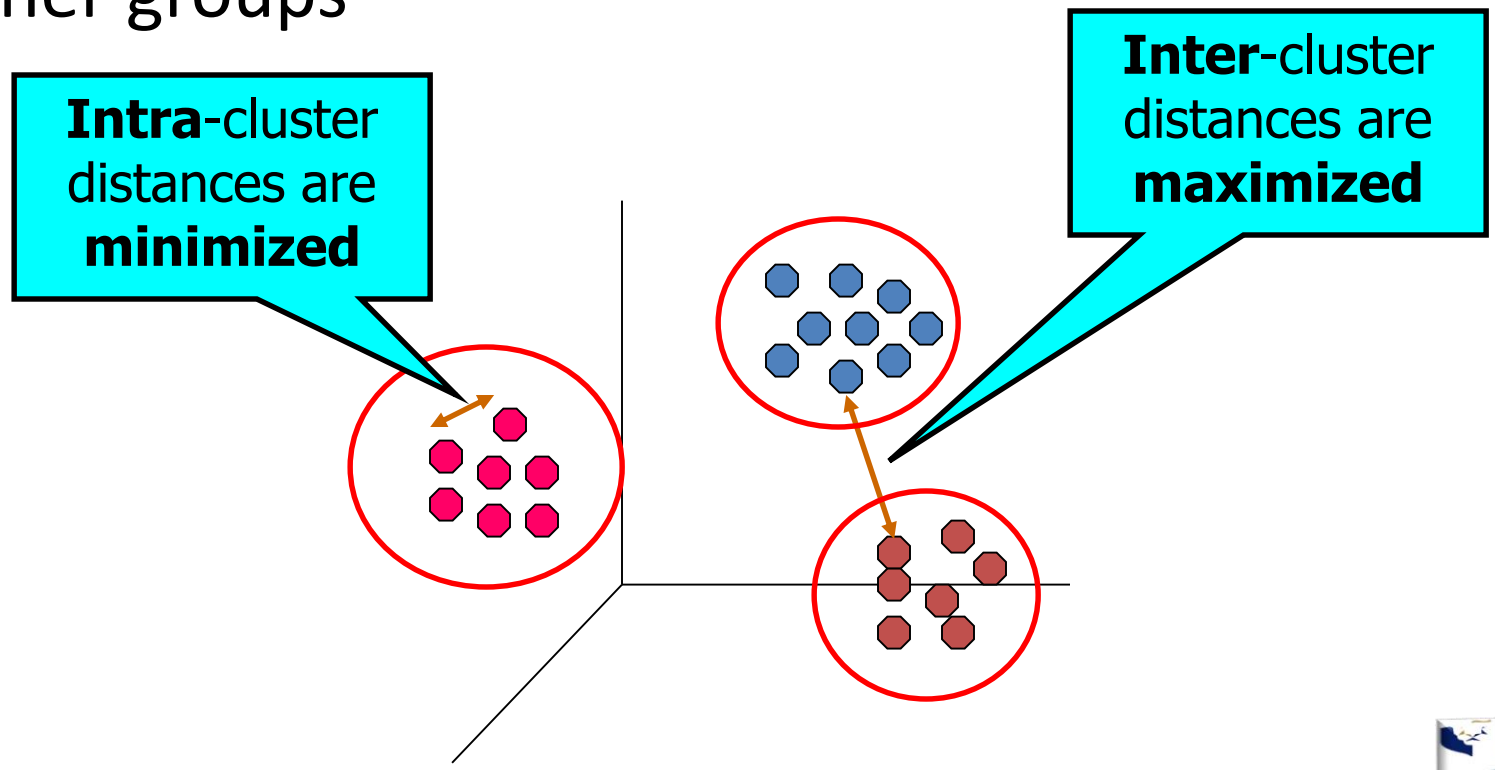
http://www.ece.neu.edu/groups/rpl/kmeans/

Application: Market Segmentation

- **Market segmentation:** dividing a broad target market into subsets of consumers, businesses, or countries that have common needs, interests, and priorities.

Clustering: Principle

- Finding groups of objects such that the objects in a group will be similar (or related) to one another and different from (or unrelated to) the objects in other groups



Example:
How to Find good Clustering?

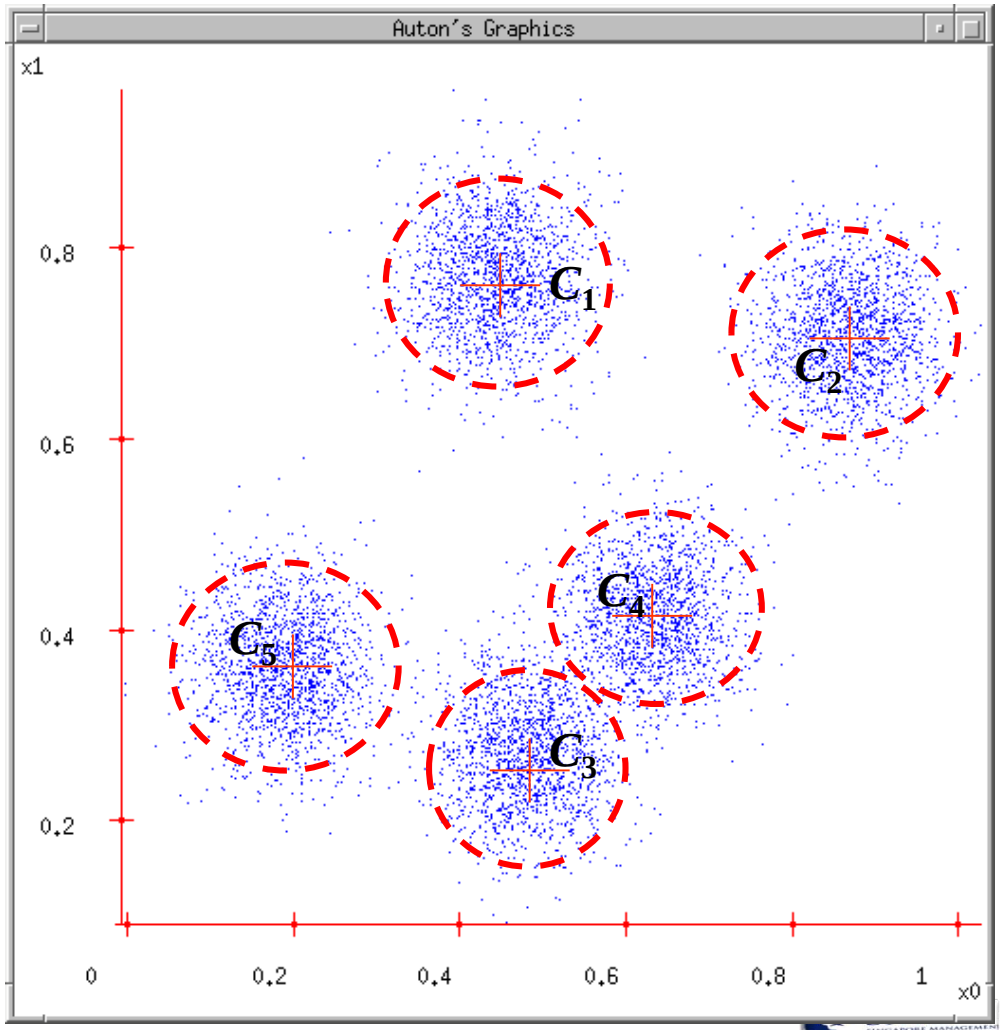
- Minimize the sum of distance within clusters

$$\arg\min_{\{\vec{C}_j, m_{i,j}\}} \sum_{j=1}^{5} \sum_{i=1}^{n} m_{i,j} \left(\vec{x}_i - \vec{C}_j \right)^2$$

$$m_{i,j} = \begin{cases} 1 & \vec{x}_i \in \text{the j-th cluster} \\ 0 & \vec{x}_i \notin \text{the j-th cluster} \end{cases}$$

$$\sum_{j=1}^{5} m_{i,j} = 1$$

$$\rightarrow \text{any } \vec{x}_i \in \text{ a single cluster}$$



How to Efficiently Clustering Data?

- Goal: $$\underset{\{\vec{C}_j, m_{i,j}\}}{\arg\min} \sum_{j=1}^{k} \sum_{i=1}^{n} m_{i,j} \left(\vec{x}_i - \vec{C}_j \right)^2$$

  Memberships $\{m_{i,j}\}$ and centers $\{C_j\}$ are correlated.

- Step 1: estimate membership

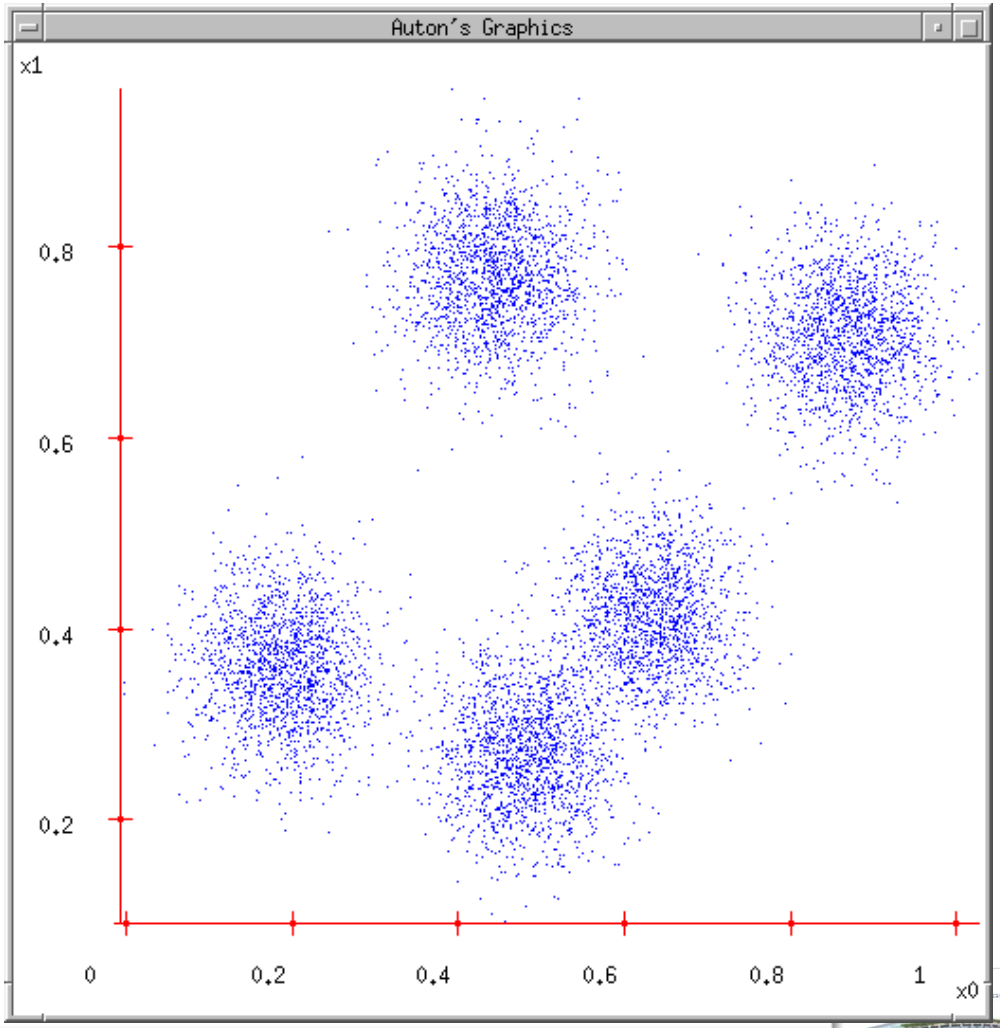
$$\text{Given centers } \{\vec{C}_j\},\ m_{i,j} = \begin{cases} 1 & j = \arg\min_k (\vec{x}_i - \vec{C}_j)^2 \\ 0 & \text{otherwise} \end{cases}$$

- Step 2: estimate cluster center

$$\text{Given memberships } \{m_{i,j}\},\ \vec{C}_j = \frac{\sum_{i=1}^{n} m_{i,j} \vec{x}_i}{\sum_{i=1}^{n} m_{i,j}}$$

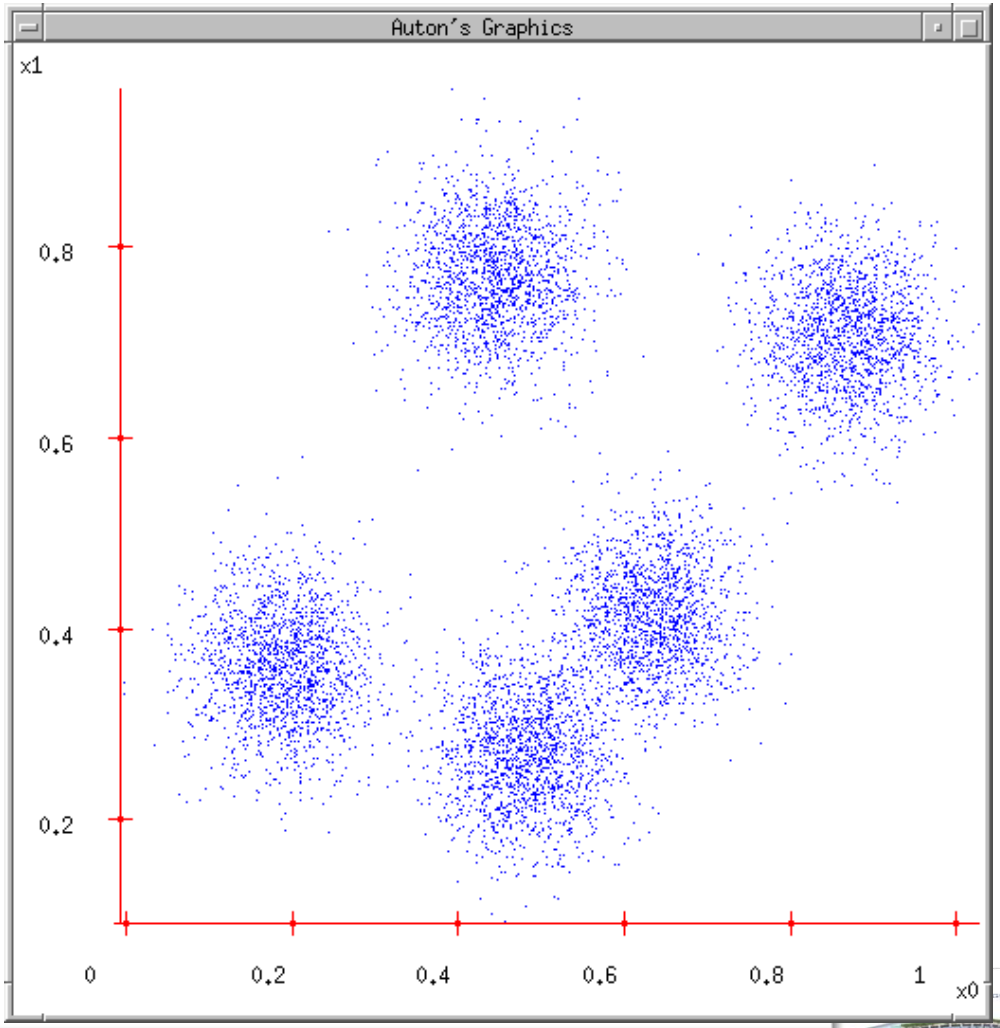
K-means for Clustering

- K-means algorithm
  - Step 0: Start with a random guess of cluster centers
  - Step 1: Determine the membership of each data points
  - Step 2: Adjust the cluster centers



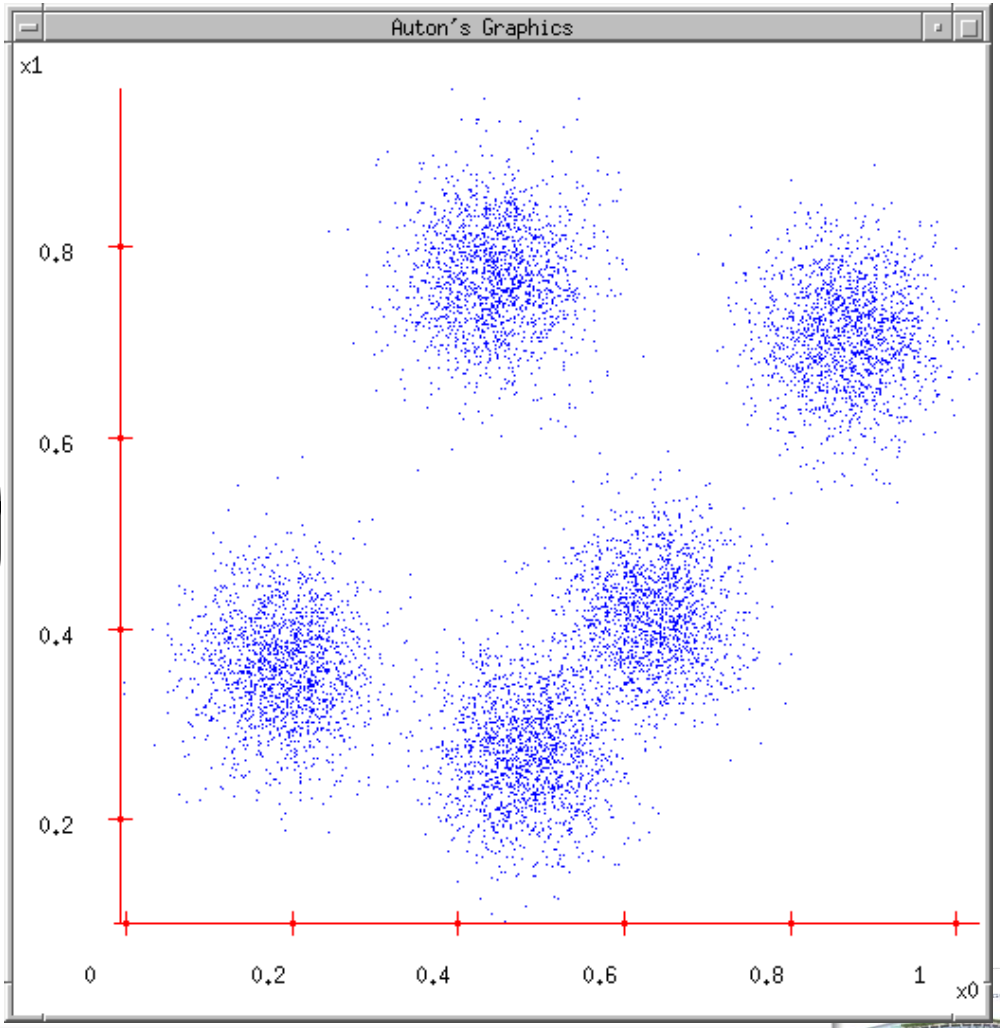
K-means for Clustering

- K-means algorithm
  - Step 0: Start with a random guess of cluster centers
  - Step 1: Determine the membership of each data points
  - Step 2: Adjust the cluster centers



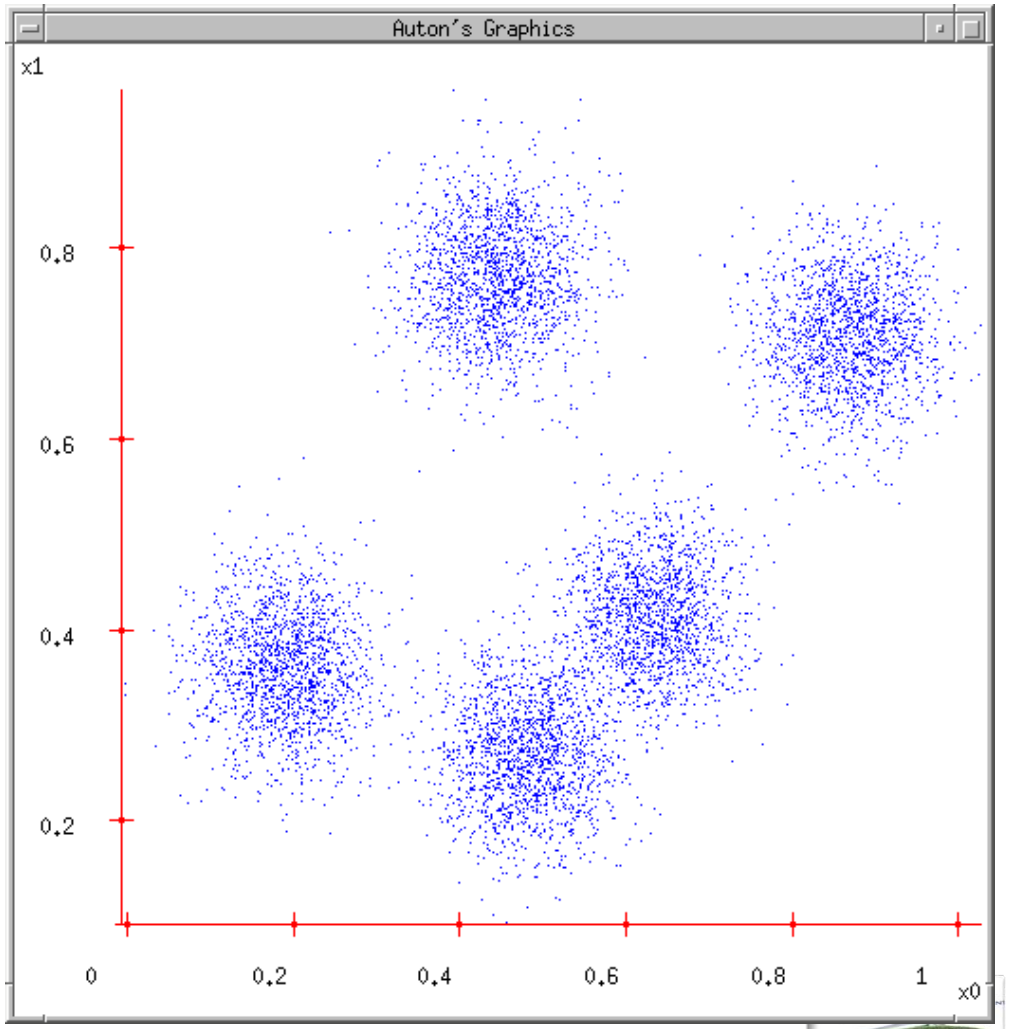
K-means for Clustering

- K-means algorithm
  - Step 0: Start with a random guess of cluster centers
  - Step 1: Determine the membership of each data points
  - Step 2: Adjust the cluster centers



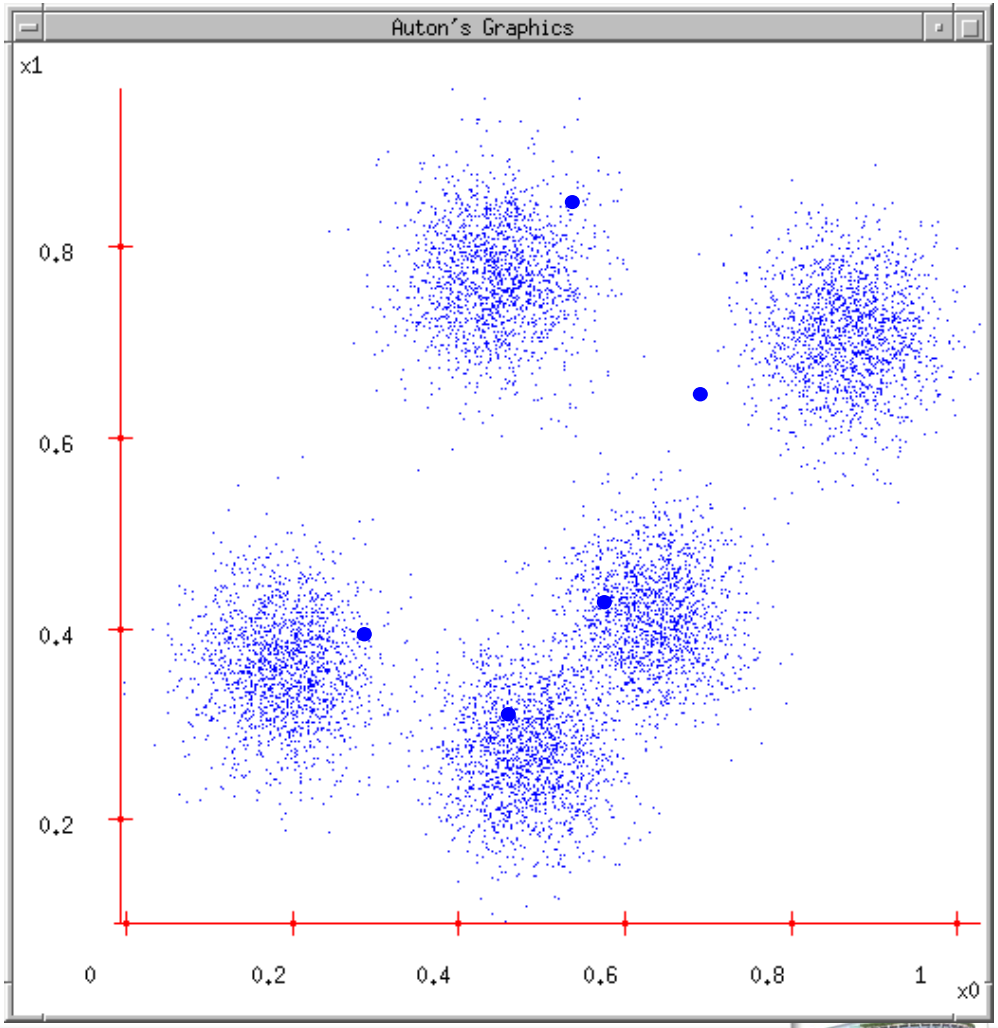
K-means

1. Ask user how many clusters they'd like. (e.g. k=5)



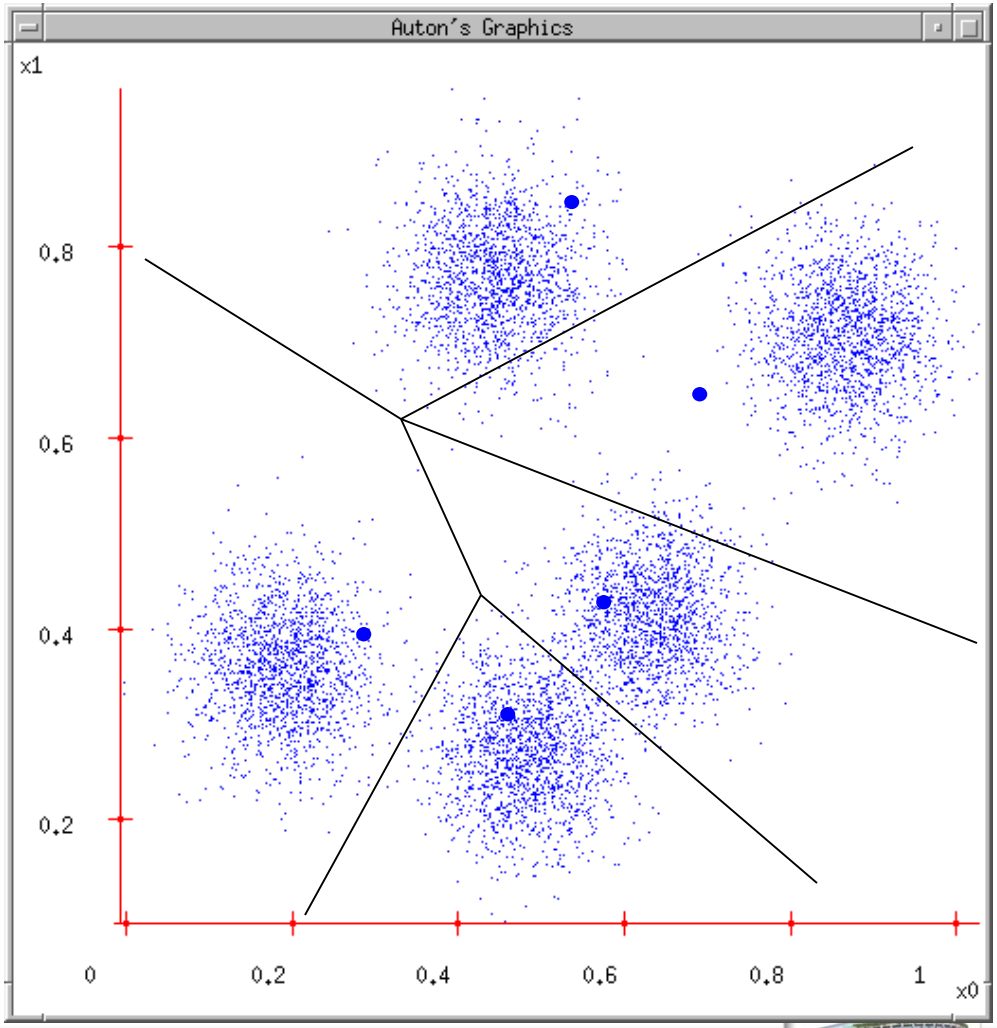
K-means

1. Ask user how many clusters they'd like. (e.g. k=5)
2. Randomly guess k cluster Center locations



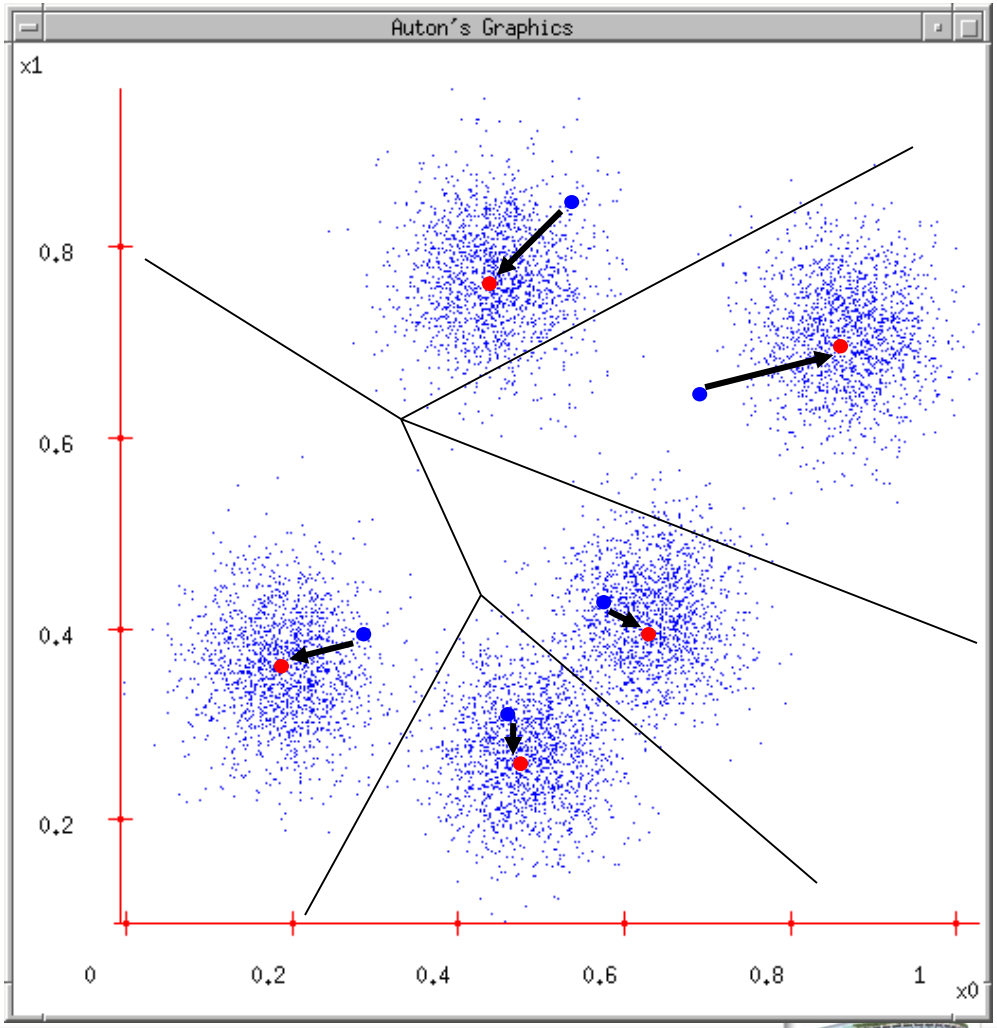
K-means

1. Ask user how many clusters they'd like. *(e.g. k=5)*
2. Randomly guess k cluster Center locations
3. Each datapoint finds out which Center it's closest to. (Thus each Center "owns" a set of datapoints)



K-means

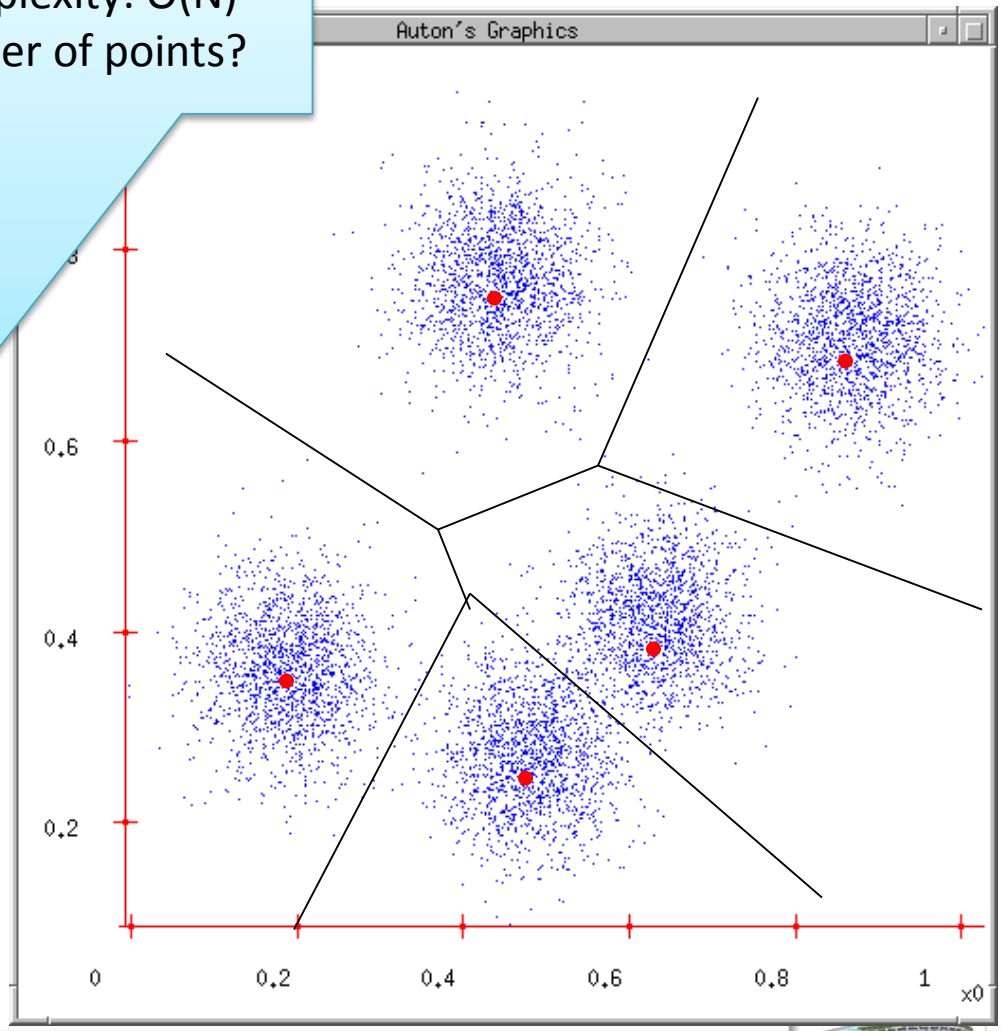
1. Ask user how many clusters they'd like. (e.g. k=5)
2. Randomly guess k cluster Center locations
3. Each datapoint finds out which Center it's closest to.
4. Each Center finds the centroid of the points it owns



K-means

Computational Complexity: O(N) where N is the number of points?

1. Ask user how many clusters they'd like. (e.g. k=5)
2. Randomly guess k cluster Center locations
3. Each datapoint finds out which Center it's closest to.
4. Each Center finds the centroid of the points it owns



Limitations with K-means

- Computational efficiency
  - Assume N – dataset size, D – dimensionality, k- number of clusters, T – number of iterations
  - Time complexity: O(N*D*k*T)
  - Solutions: efficient data structures (e.g., KD-tree)
- Model limitation
  - Assume sphere shape of a cluster (“perfectly round”)
  - Solutions: Gaussian Mixture Models

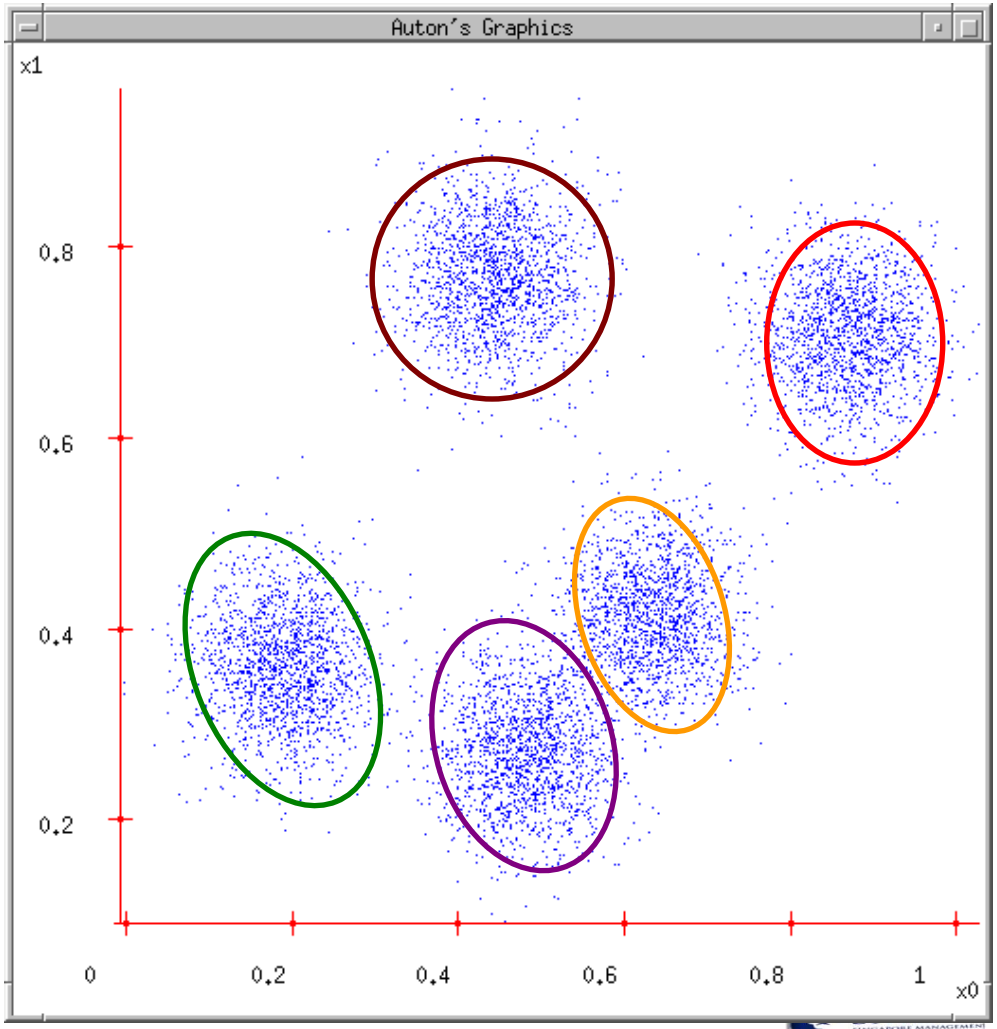
A Gaussian Mixture Model for Clustering

- Assume that data are generated from a mixture of Gaussian distributions

- For each Gaussian distribution
  - Center: μ_i
  - Variance: Σ_i (ignore)

- For each data point
  - Determine membership

$$z_{ij} : \text{ if } x_i \text{ belongs to j-th cluster}$$

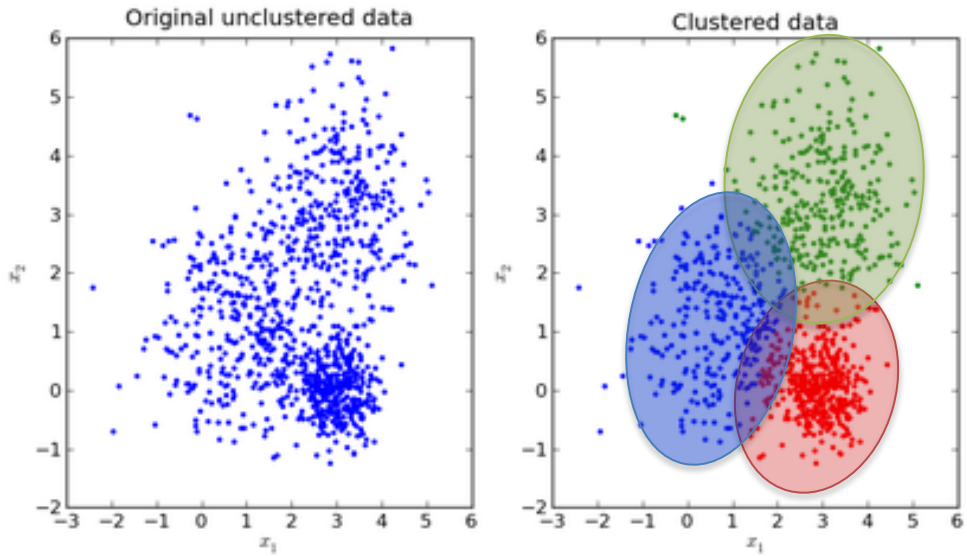


Learning a Gaussian Mixture
(with known covariance)

- Probability of one sample

$$p(x = x_i)$$

$$= \sum_{\mu_j} p(x = x_i, \mu = \mu_j)$$



$$= \sum_{\mu_j} p(\mu = \mu_j) p(x = x_i \mid \mu = \mu_j)$$

$$= \sum_{\mu_j} p(\mu = \mu_j) \frac{1}{\left(2\pi\sigma^2\right)^{d/2}} \exp\left(-\frac{\left\| x_i - \mu_j \right\|_2}{2\sigma^2} \right)$$

Learning a Gaussian Mixture
(with known covariance)

- Probability of one sample

$$p(x = x_i) = \sum_{\mu_j} p(\mu = \mu_j) \frac{1}{\left(2\pi\sigma^2\right)^{d/2}} \exp\left(-\frac{\left\| x_i - \mu_j \right\|_2}{2\sigma^2} \right)$$

- Log-likelihood of all data

$$\sum_i \log p(x = x_i) = \sum_i \log \left[\sum_{\mu_j} p(\mu = \mu_j) \frac{1}{\left(2\pi\sigma^2\right)^{d/2}} \exp\left(-\frac{\left\| x_i - \mu_j \right\|_2}{2\sigma^2} \right) \right]$$

- Apply MLE to find optimal parameters $\left\{ p(\mu = \mu_j), \mu_j \right\}_j$

EM algorithm for a Gaussian Mixture
(with known covariance)

- **Expectation step: "E-Step"**

Estimate membership

$$E[z_{ij}] = p(\mu = \mu_j \mid x = x_i)$$

(Bayes Theorem)

$$= \frac{p(x = x_i \mid \mu = \mu_j)\, p(\mu = \mu_j)}{\sum_{n=1}^{k} p(x = x_i \mid \mu = \mu_n)\; p(\mu = \mu_n)}$$

$$= \frac{e^{-\frac{1}{2\sigma^2}(x_i - \mu_j)^2}\, p(\mu = \mu_j)}{\sum_{n=1}^{k} e^{-\frac{1}{2\sigma^2}(x_i - \mu_n)^2}\, p(\mu = \mu_n)}$$

Learning a Gaussian Mixture
(with known covariance)

- **Maximization step: "M-Step"**

Estimate cluster centers

$$\mu_j \leftarrow \frac{1}{\sum_{i=1}^{m} E[z_{ij}]} \sum_{i=1}^{m} E[z_{ij}] x_i$$

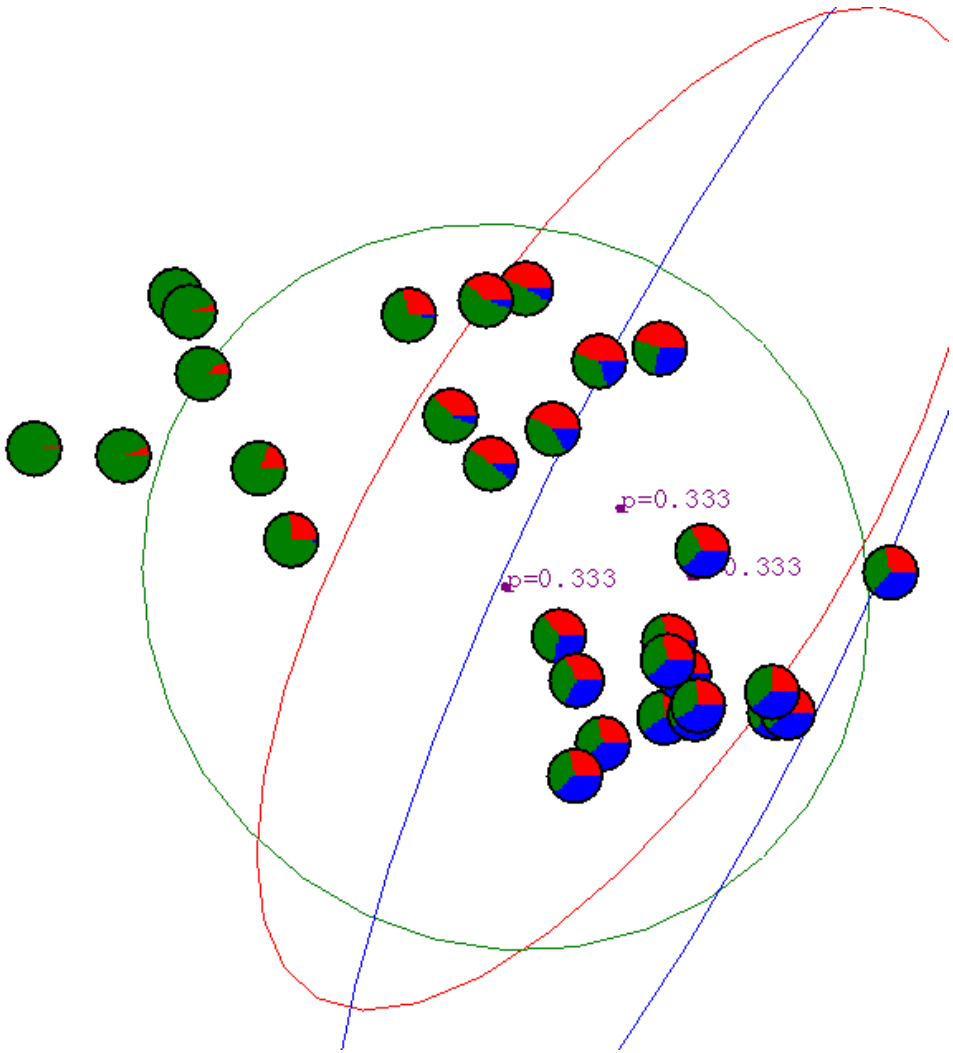
$$p(\mu = \mu_j) \leftarrow \frac{1}{m} \sum_{i=1}^{m} E[z_{ij}]$$

Learning a Gaussian Mixture
(with known covariance)

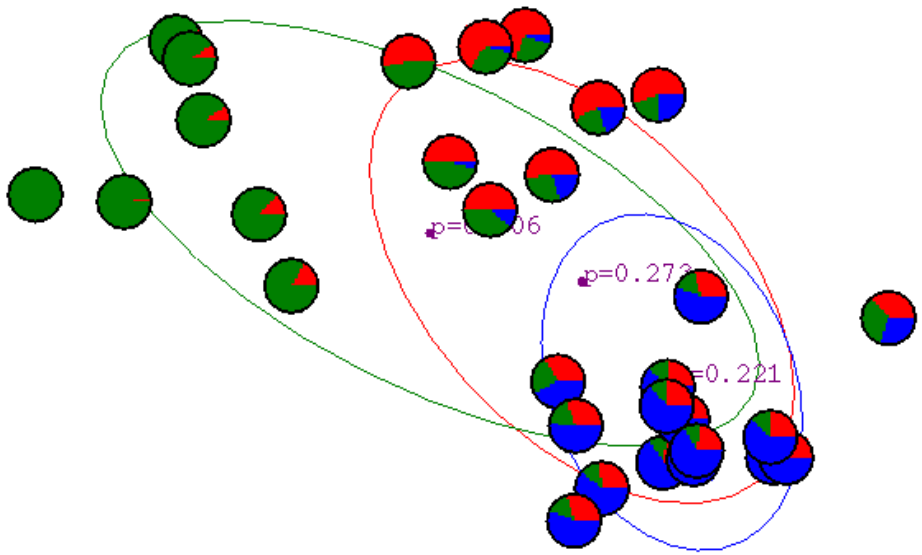
- EM Algorithm for Clustering
- Repeat
  - **"E-Step": Estimate** membership of each data points
  - **"M-Step"**: **Estimate** the cluster centers (and prior)

  Until convergence

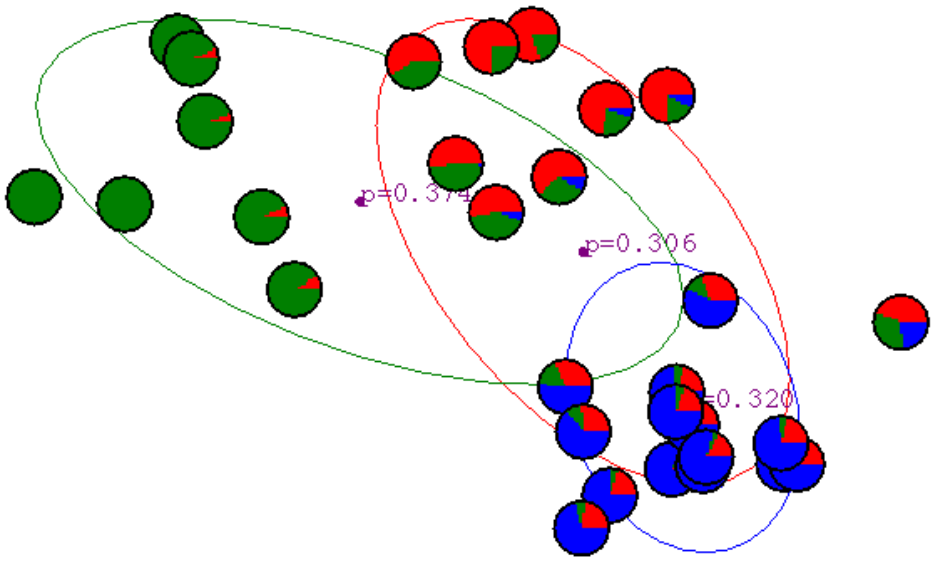
Gaussian Mixture Example: Start



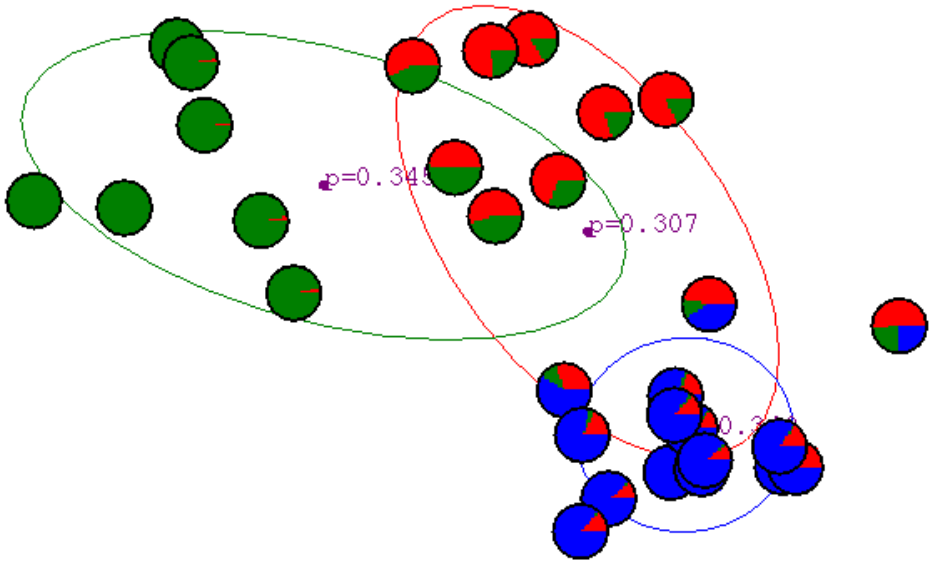
After First Iteration



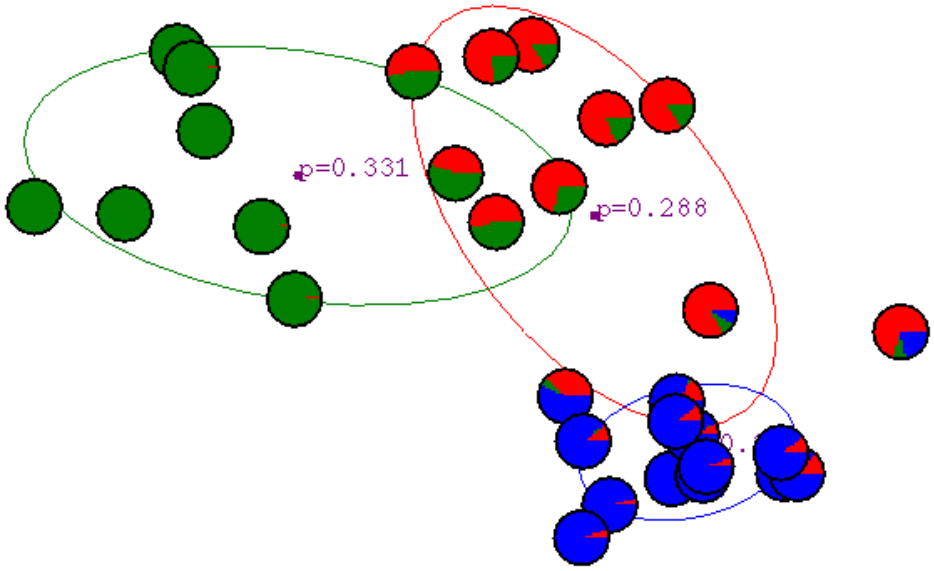
After 2nd Iteration



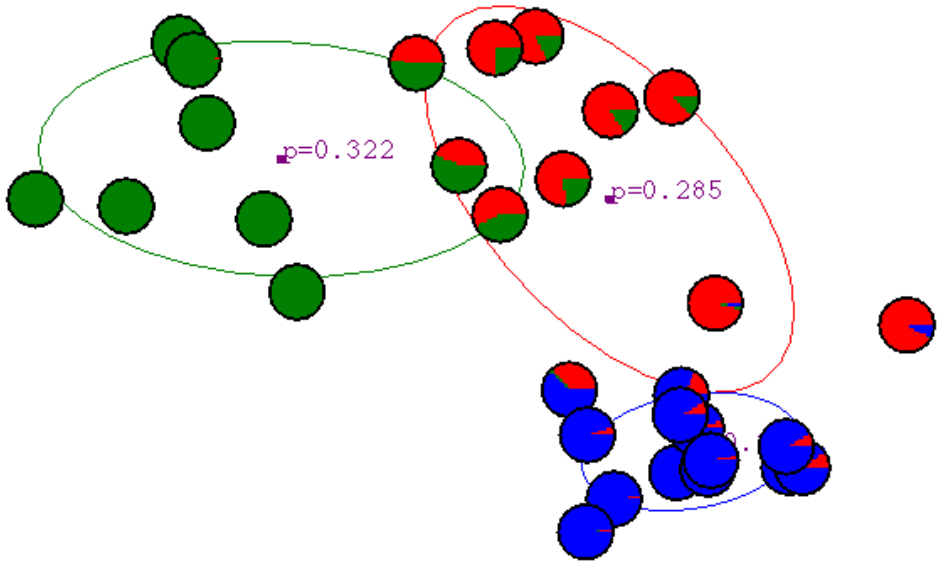
After 3rd Iteration



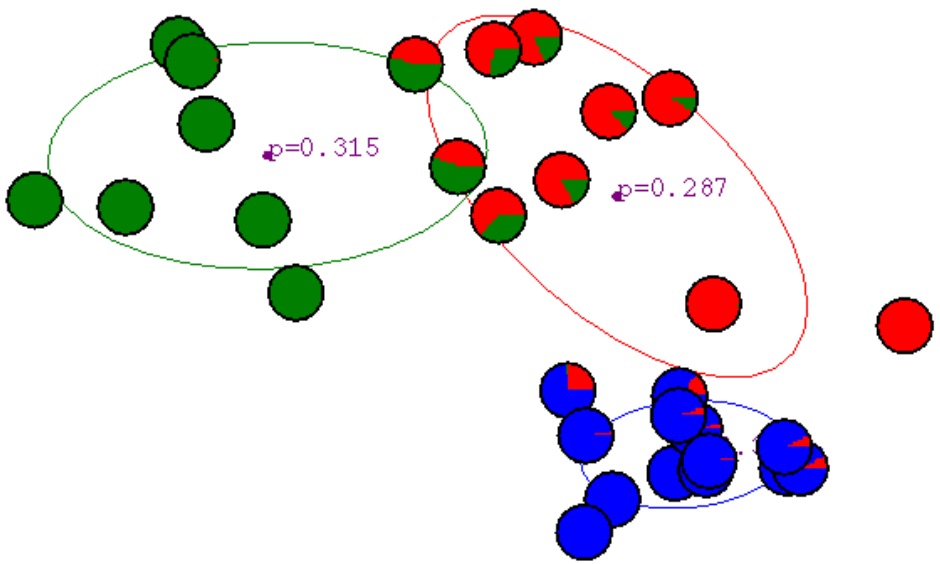
After 4th Iteration



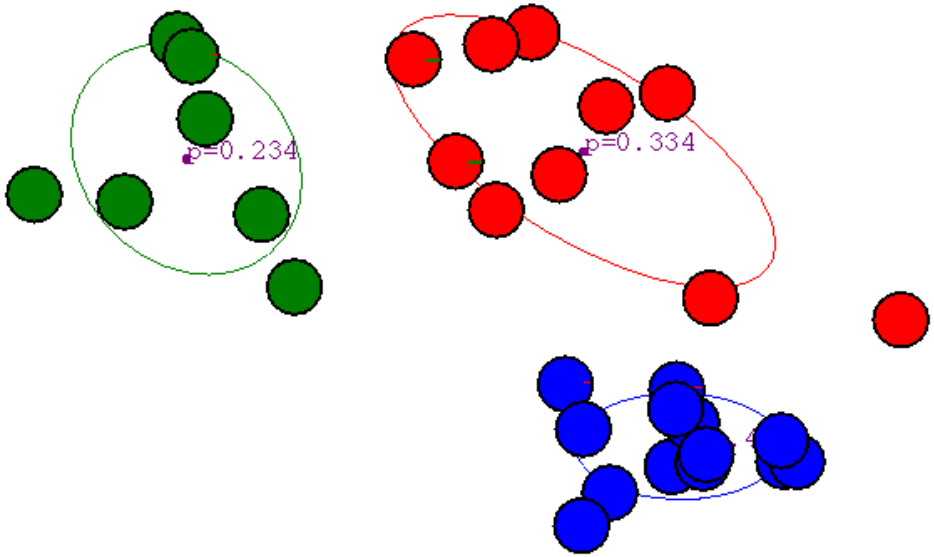
After 5th Iteration



After 6th Iteration



After 20th Iteration



EM (GMM) vs K-means

K-means	EM (GMM)
Spherical clusters (identical covariance matrices)	Beyond spherical clusters
Hard assignments	Probabilistic assignments
A special case of EM	More general

SMU
SINGAPORE MANAGEMENT UNIVERSITY

Dimension Reduction

Dimension Reduction

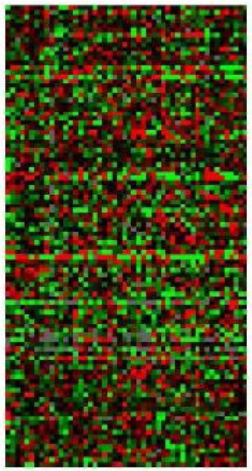
- Lots of high-dimensional data

face images

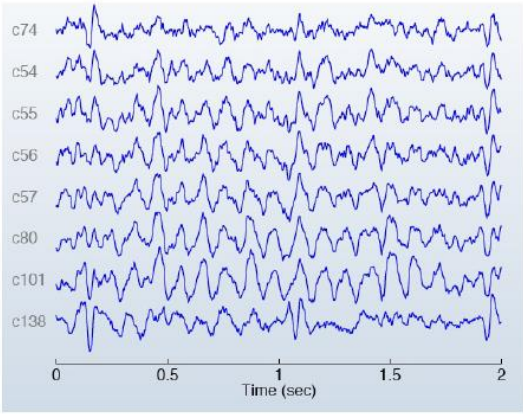
Zambian President Levy Mwanawasa has won a second term in office in an election his challenger Michael Sata accused him of rigging, official results showed on Monday.

According to media reports, a pair of hackers said on Saturday that the Firefox Web browser, commonly perceived as the safer and more customizable alternative to market leader Internet Explorer, is critically flawed. A presentation on the flaw was shown during the ToorCon hacker conference in San Diego.

text documents



gene expression data



MEG readings

Dimension Reduction

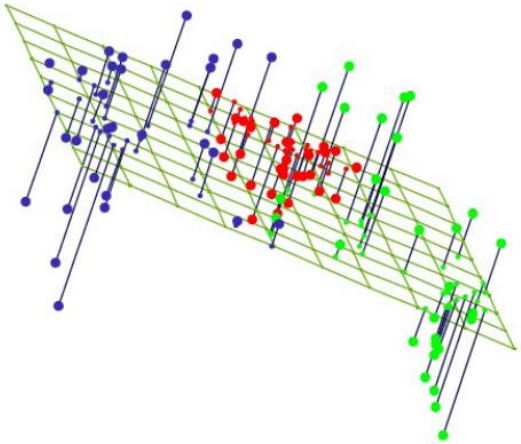
- Why dimension reduction?
  - Computational:

    compress data → time/space efficiency
  - Visualization: understand structure of data
  - Statistical: fewer dimensions → better generalization
  - Anomaly detection: describe normal data, detect outliers

Basic Idea of Dimension Reduction

- Represent each face as a high-dimensional vector

$$\mathbf{x} \in \mathbb{R}^{361}$$



$$\mathbf{x} \in \mathbb{R}^{361}$$

$$\downarrow \; \mathbf{z} = \mathbf{U}^\top \mathbf{x}$$

$$\mathbf{z} \in \mathbb{R}^{10}$$

How do we choose $\mathbf{U}$?

Dimension Reduction: Setting

- Given n data points in d dimensions: $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^d$

$$\mathbf{X} = \begin{pmatrix} | & & | \\ \mathbf{x}_1 & \cdots\cdots & \mathbf{x}_n \\ | & & | \end{pmatrix} \in \mathbb{R}^{d \times n}$$

- Want to reduce dimensionality from d to k
- Choose k directions $\mathbf{u}_1, \ldots, \mathbf{u}_k$

$$\mathbf{U} = \begin{pmatrix} | & & | \\ \mathbf{u}_1 & \cdots & \mathbf{u}_k \\ | & & | \end{pmatrix} \in \mathbb{R}^{d \times k}$$

- For each $\mathbf{u}_j$, compute "similarity" $z_j = \mathbf{u}_j^\top \mathbf{x}$
- Project $\mathbf{x}$ down to $\mathbf{z} = (z_1, \ldots, z_k)^\top = \mathbf{U}^\top \mathbf{x}$
- **How to choose $\mathbf{U}$?**

Principal Component Analysis (PCA)

- $\mathbf{U}$ serves two functions
  - Encode: $\mathbf{z} = \mathbf{U}^\top \mathbf{x}, \quad z_j = \mathbf{u}_j^\top \mathbf{x}$
  - Decode: $\tilde{\mathbf{x}} = \mathbf{U}\mathbf{z} = \sum_{j=1}^{k} z_j \mathbf{u}_j$
- Want reconstruction error $\|\mathbf{x} - \tilde{\mathbf{x}}\|$ to be small
- Objective of PCA:
  - Minimize total squared reconstruction error

$$\min_{\mathbf{U} \in \mathbb{R}^{d \times k}} \sum_{i=1}^{n} \|\mathbf{x}_i - \mathbf{U}\mathbf{U}^\top \mathbf{x}_i\|^2$$

PCA algorithm (sample covariance matrix)

- Given data $\{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$, compute covariance matrix Σ

$$\boxed{\Sigma = \frac{1}{n}\sum_{i=1}^{n}(\mathbf{x}_i - \bar{\mathbf{x}})(\mathbf{x} - \bar{\mathbf{x}})^T}$$

where

$$\boxed{\bar{\mathbf{x}} = \frac{1}{n}\sum_{i=1}^{n}\mathbf{x}_i}$$

- **PCA** basis vectors = the eigenvectors of Σ

- Larger eigenvalue $\Rightarrow$ more important eigenvectors

PCA algorithm

PCA algorithm($\mathbf{X}$, k): top k eigenvalues/eigenvectors

% $\underline{\mathbf{X}}$ = d × n data matrix,
% … each data point $\mathbf{x}_i$ = column vector, i=1..n

- $\underline{\mathbf{x}} = \frac{1}{n}\sum_{i=1}^{n}\mathbf{x}_i$
- $\mathbf{X} \leftarrow$ subtract mean $\underline{\mathbf{x}}$ from each column vector $\mathbf{x}_i$ in $\underline{\mathbf{X}}$
- $\Sigma \leftarrow \mathbf{X}\mathbf{X}^\top$ … covariance matrix of $\mathbf{X}$
- $\{\lambda_i, \mathbf{u}_i\}_{i=1..d}$ = eigenvectors/eigenvalues of Σ
  … $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_d$
- Return $\{\lambda_i, \mathbf{u}_i\}_{i=1..k}$
  % top k principal components

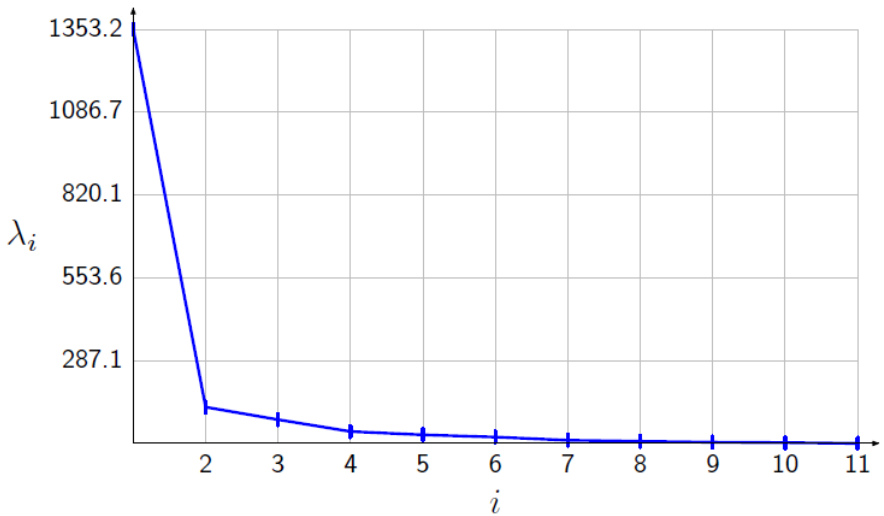
$$\mathbf{z}_i = \mathbf{U}^\top(\mathbf{x}_i - \underline{\mathbf{x}})$$

Number of principal components

- Choosing the number of principal components
  - Similar to question of “How many clusters?“

- Magnitude of eigenvalues indicate fraction of variance captured.

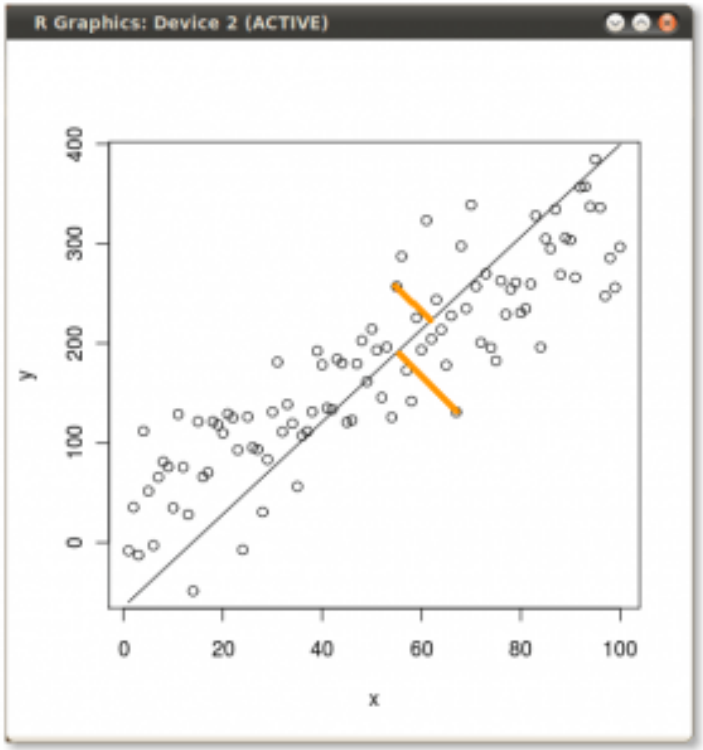
- Example of Eigenvalues on a face image dataset:

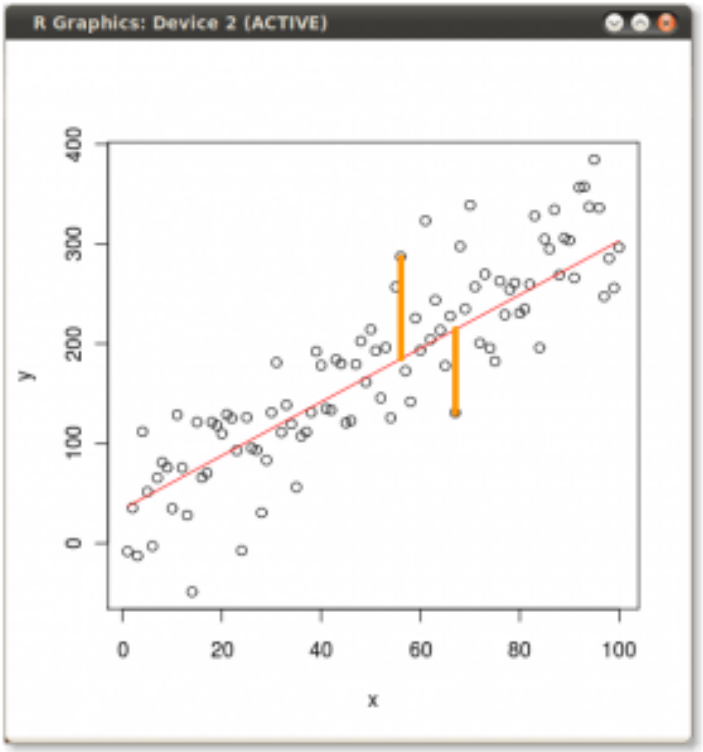


- Eigenvalues typically drop off sharply, so don't need that many.

PCA vs Linear Regression

- PCA minimizes the error orthogonal (perpendicular) to the model line.
- Linear Regression minimizes the error between the dependent and the model.



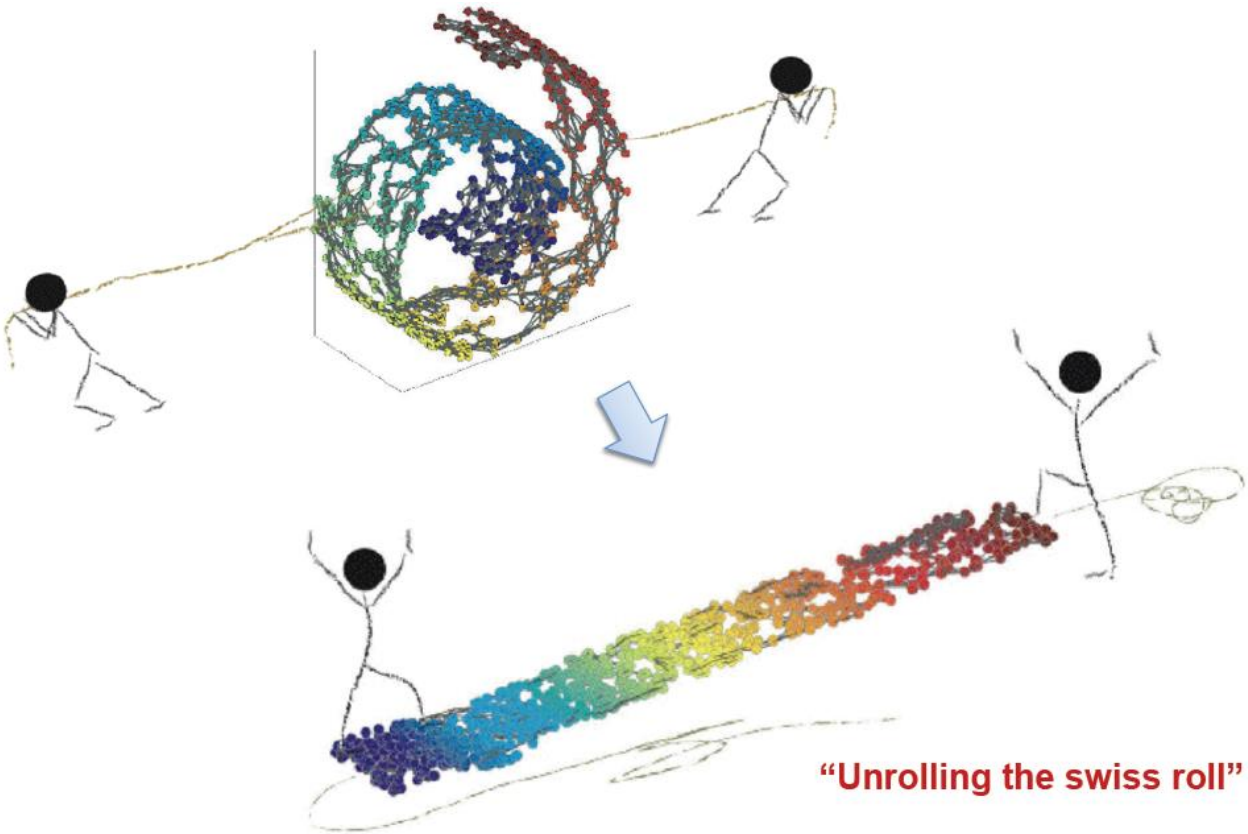


PCA Applications

- Compression
  - Reduce memory/disk needed to store data
  - Speed up learning algorithms
- Visualization
  - Project to 2D or 3D space
- Other usages
  - Denoising
  - To prevent overfitting (not the ideal way)

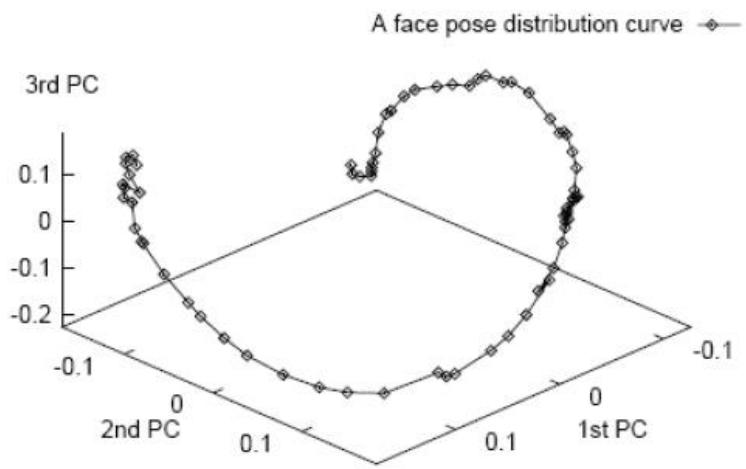
Nonlinear Dimension Reduction

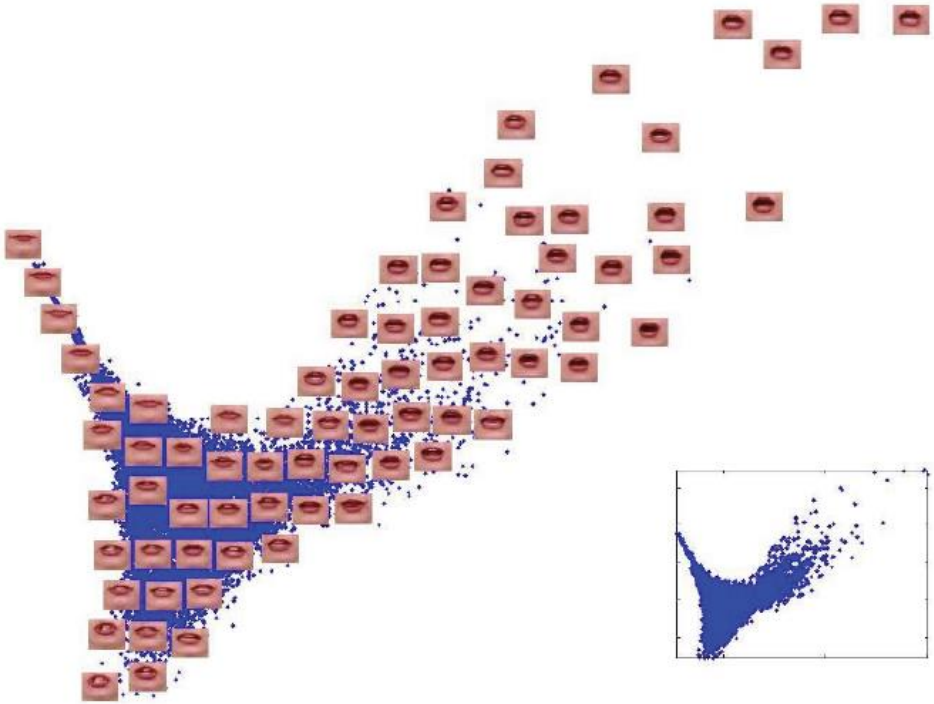
- Data often lies on or near a nonlinear low-dimensional curve (a.k.a “manifold”)



Nonlinear Dimension Reduction

- Example data sets





Nonlinear Dimension Reduction

- Manifold learning
  - Locally Linear Embedding (Sam T. Roweis & Lawrence K. Saul, Science 2000)
  - Isomap (J. B. Tenenbaum, V. de Silva, J. C. Langford, Science 2000)
  - Lapalcian Eigenmap (Mikhail Belkin and Partha Niyogi, NIPS'01)

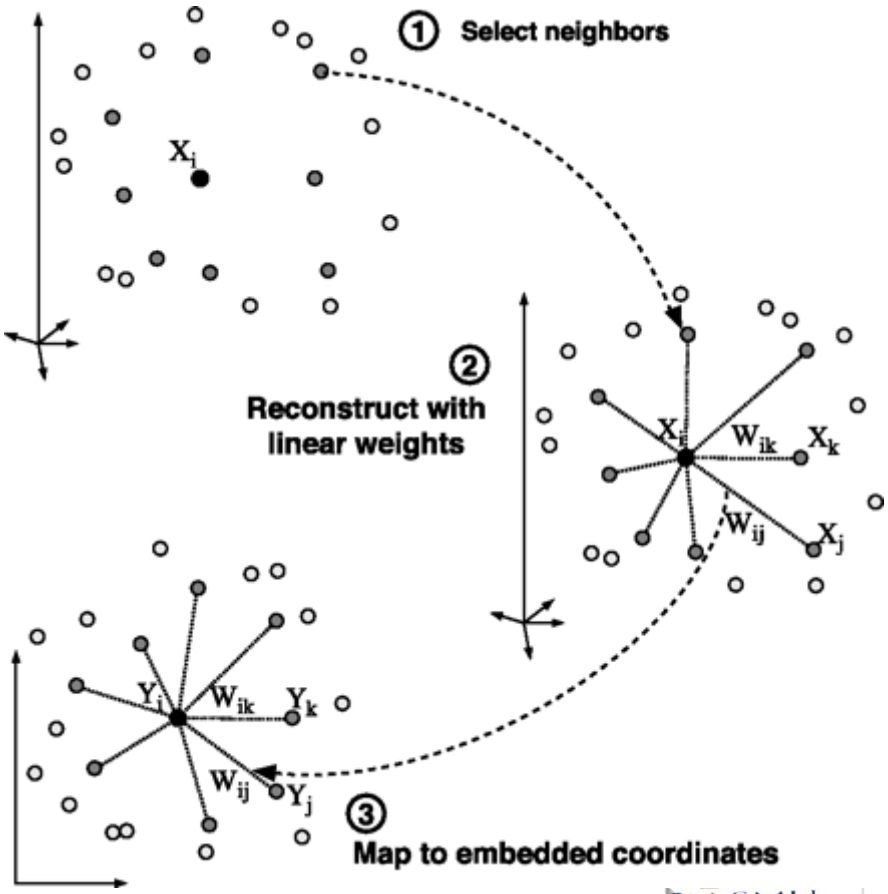
Locally Linear Embedding (LLE)

- Principle of Local method:
  - Points nearby should be mapped nearby, while points far away should impose no constraint.
- Optimization
  - **Local Fitting** (Step 2):
    Find reconstruction weight W

$$\mathcal{E}(W) = \sum_i \left| \vec{X}_i - \sum_j W_{ij} \vec{X}_j \right|^2, \qquad (1)$$

  - **Global Alignment** (Step 3):
    Find Y reconstructed by W

$$\Phi(Y) = \sum_i \left| \vec{Y}_i - \sum_j W_{ij} \vec{Y}_j \right|^2. \qquad (2)$$



LLE Algorithm

1. Compute the neighbors of each data point X

2. Compute the weights **W** that best reconstruct each data point X from its neighbors, minimizing the cost in Eq. (1) by constrained linear fits

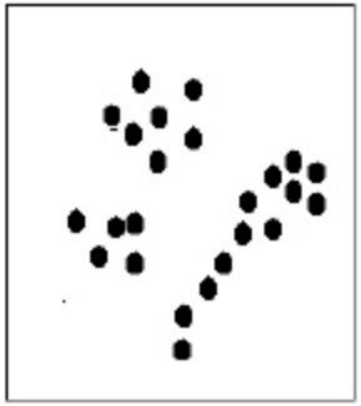
3. Compute the vector **Y** best reconstructed by the weight W, minimizing the quadratic from in Eq. (2) by its bottom nonzero eigenvectors

PCA versus LLE

PCA	LLE
Linear approach	Nonlinear approach
Global approach	Local approach
Global error reconstruction, global eigen-decomposition	local PCA for local fitting and global eigen-decomposition

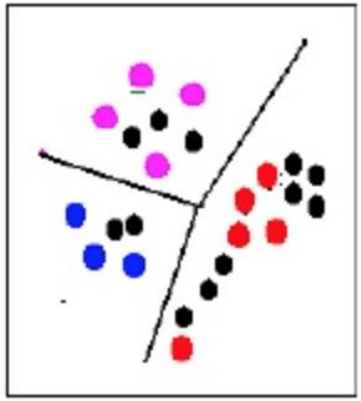
One more thing

Unsupervised Learning



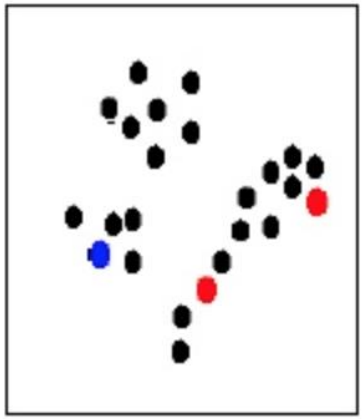
No label info

Supervised Learning



Sufficient labeled data

Semi-Supervised Learning



Limited labeled data

Summary

- Unsupervised Learning
- Clustering
  - K-means clustering
  - EM Clustering
- Dimension Reduction
  - Linear: PCA
  - Nonlinear: LLE

Appendix

- More about PCA
- More dimension reduction methods
- More clustering methods

Another Objective of PCA

- Empirical distribution: uniform over $\mathbf{x}_1, \ldots, \mathbf{x}_n$
- Expectation (think sum over data points):

$$\hat{\mathbb{E}}[f(\mathbf{x})] = \frac{1}{n}\sum_{i=1}^{n} f(\mathbf{x}_i)$$

- Variance (think sum of squares if centered):

$$\widehat{\mathrm{var}}[f(\mathbf{x})] + (\hat{\mathbb{E}}[f(\mathbf{x})])^2 = \hat{\mathbb{E}}[f(\mathbf{x})^2] = \frac{1}{n}\sum_{i=1}^{n} f(\mathbf{x}_i)^2$$

- Assume data is centered: $\hat{\mathbb{E}}[\mathbf{x}] = 0$ (what's $\hat{\mathbb{E}}[\mathbf{U}^\top \mathbf{x}]$?)
- Objective: maximize variance of projected data

$$\max_{\mathbf{U}\in\mathbb{R}^{d\times k}, \mathbf{U}^\top\mathbf{U}=I} \hat{\mathbb{E}}[\|\mathbf{U}^\top\mathbf{x}\|^2]$$

Equivalence between two Objectives

- Intuition

$$\underbrace{\text{variance of data}}_{\text{fixed}} = \underbrace{\text{captured variance}}_{\text{want large}} + \underbrace{\text{reconstruction error}}_{\text{want small}}$$

$$\hat{\mathbb{E}}[||\mathbf{x}||^2] = \hat{\mathbb{E}}[||\mathbf{U}^\top \mathbf{x}||^2] + \hat{\mathbb{E}}[||\mathbf{x} - \mathbf{U}\mathbf{U}^\top \mathbf{x}||^2]$$

- **Maximize captured variance**
  - PCA finds vectors **u** such that projections on to the vectors capture **maximum variance** in the data
- **Minimize reconstruction error**
  - PCA finds vectors **u** such that projection on to the vectors yields **minimum squared reconstruction error**

Finding one principal component

- Objective: maximize variance of projected data

$$= \max_{\|\mathbf{u}\|=1} \hat{\mathbb{E}}[(\mathbf{u}^\top \mathbf{x})^2]$$

$$= \max_{\|\mathbf{u}\|=1} \frac{1}{n} \sum_{i=1}^{n} (\mathbf{u}^\top \mathbf{x}_i)^2$$

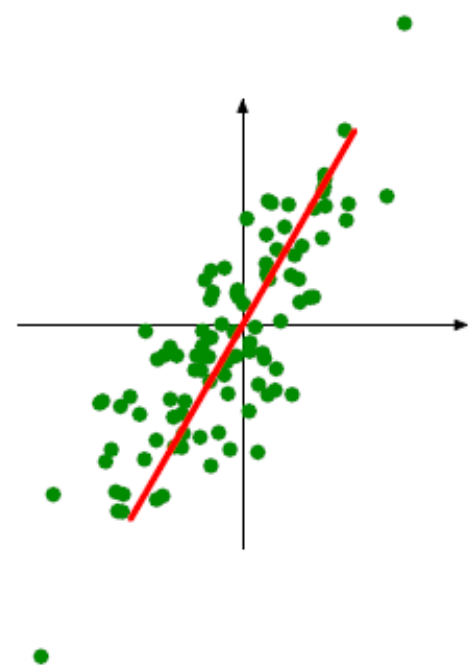
$$= \max_{\|\mathbf{u}\|=1} \frac{1}{n} \|\mathbf{u}^\top \mathbf{X}\|^2$$

$$= \max_{\|\mathbf{u}\|=1} \mathbf{u}^\top \left(\frac{1}{n}\mathbf{X}\mathbf{X}^\top\right) \mathbf{u}$$

$$= \text{largest eigenvalue of } C \stackrel{\text{def}}{=} \frac{1}{n}\mathbf{X}\mathbf{X}^\top$$

(C is covariance matrix of data)

$$\mathbf{X} = \begin{pmatrix} | & & | \\ \mathbf{x}_1 & \dots & \mathbf{x}_n \\ | & & | \end{pmatrix}$$



Methods for computing PCA

- **Method 1: Eigen-decomposition**

  $\mathbf{U}$ are eigenvectors of covariance matrix $C = \frac{1}{n}\mathbf{X}\mathbf{X}^\top$

  Computing C already takes $O(nd^2)$ time (very expensive)

- **Method 2: singular value decomposition (SVD)**

  Find $\mathbf{X} = \mathbf{U}_{d\times d}\Sigma_{d\times n}\mathbf{V}^\top_{n\times n}$

  where $\mathbf{U}^\top\mathbf{U} = I_{d\times d}$, $\mathbf{V}^\top\mathbf{V} = I_{n\times n}$, Σ is diagonal

  Computing top k singular vectors takes only $O(ndk)$

- Relationship between eigendecomposition and SVD:
  - Left singular vectors are principal components $C = \mathbf{U}\Sigma^2\mathbf{U}^\top$

Detailed LLE Algorithm

1. Construct a neighborhood graph $G = (V, E, W)$
2. Local fitting: X → W

Pick up a point x_i and its neighbors $\mathbb{N}_i$

Compute the local fitting weights

$$\min_{\sum_{j \in \mathbb{N}_i} w_{ij} = 1} \|x_i - \sum_{j \in \mathbb{N}_i} w_{ij}(x_j - x_i)\|^2$$

3. Global alignment W → Y

Define a n-by-n weight matrix W:

$$W_{ij} = \begin{cases} w_{ij}, & j \in \mathcal{N}_i \\ 0, & otherwise \end{cases}$$

Compute the global embedding d-by-n embedding matrix Y,

$$\min_Y \sum_i \|y_i - \sum_{j=1}^{n} W_{ij} y_j\|^2 = trace(Y(I-W)^T(I-W)Y^T)$$

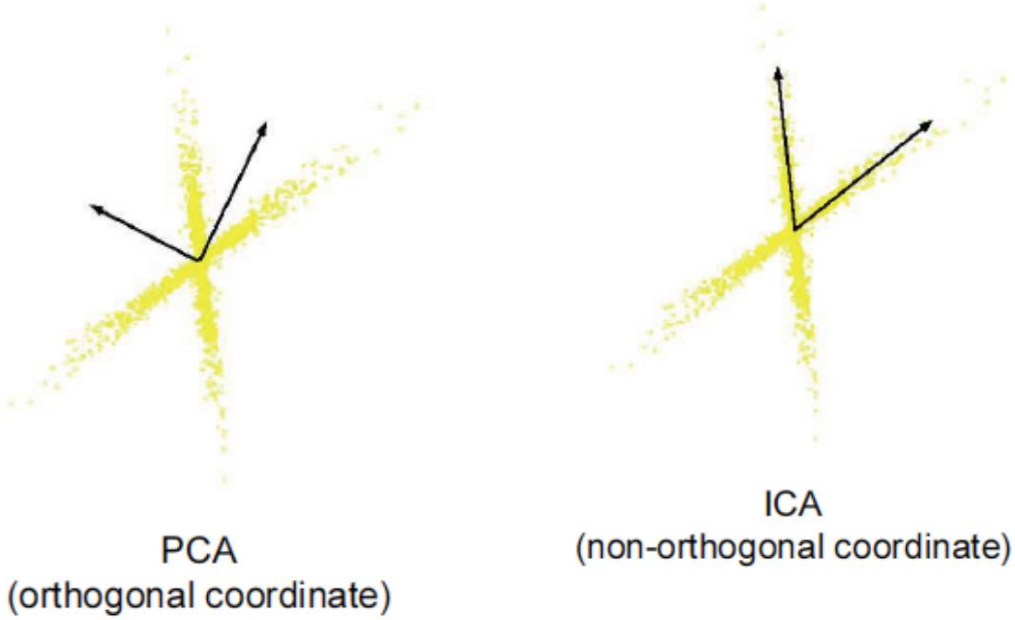
$$B = (I-W)^T(I-W)$$

Find top (d+1) smallest eigenvectors of B

$$Y = [v_1/\sqrt{(\lambda_1)}, \ldots, v_d/\sqrt{\lambda_d}]^T$$

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



Multi-Dimensional Scaling (MDS)

- MDS is a mathematical dimension reduction technique that maps the distances between observations from the original (high) dimensional space into a lower (for example, two) dimensional space.
- MDS attempts to retain inter-point distances (typically pairwise Euclidean) in the low-dimensional space .

Multi-Dimensional Scaling (MDS)

Goal: Find projection that best preserves inter-point distances

x_i Point in d dimensions

y_i Corresponding point in $r < d$ dimensions

δ_{ij} Distance between x_i and x_j

d_{ij} Distance between y_i and y_j

- Define (e.g.) $$E(\mathbf{y}) = \sum_{i,j} \left(\frac{d_{ij} - \delta_{ij}}{\delta_{ij}} \right)^2$$
- Find y_i's that minimize E by gradient descent
- Invariant to translations, rotations and scalings

MDS Algorithm

- Given the squared distance matrix $D^{n \times n}$, which is symmetric matrix,
- (1) Compute $B = -\frac{1}{2} H \cdot D \cdot H^T$, where H is a centering matrix. $H = I - \frac{1}{n} \cdot \mathbf{1} \cdot \mathbf{1}^T$

$$B = -\frac{1}{2} H \cdot D \cdot H^T = \tilde{X}^T \tilde{X}. \qquad \tilde{X} = X - \frac{1}{n} X \cdot \mathbf{1} \cdot \mathbf{1}^T$$

- (2) Do eigen-decomposition of B
- (3) Choose the top-k eigenvectors (e.g. by SVD)
- (4) Project them onto space spanned by them

Multi-Dimensional Scaling (MDS)

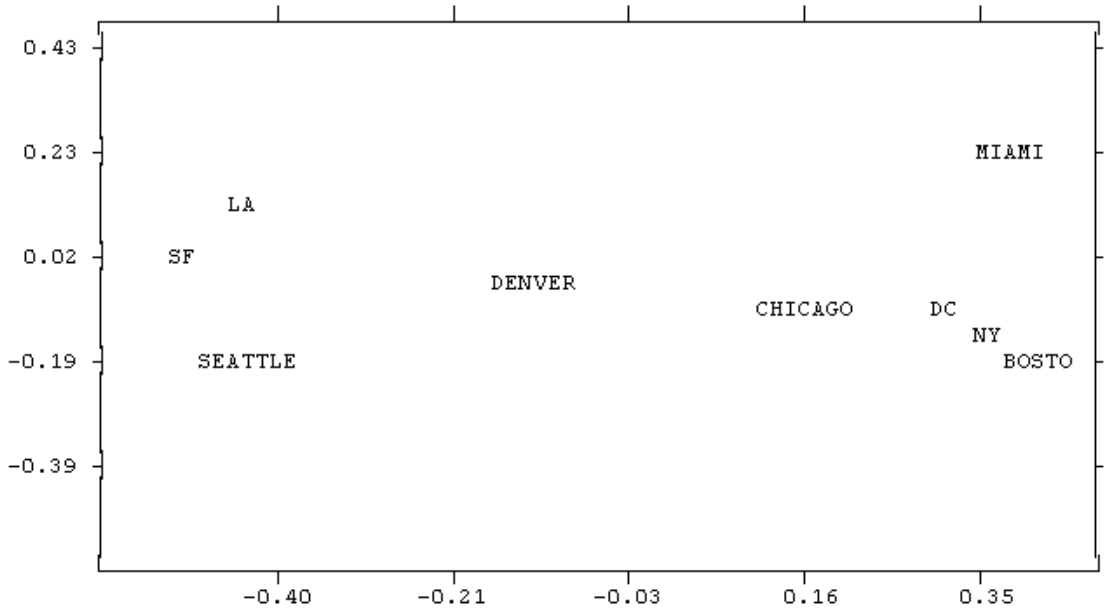
- When using Euclidean distance, it can be proved that MDS is identical to PCA
- For PCA, we need the original data points X
- For MDS, we only need the distance D, original data points are not necessary

MDS: Example

Input:

		1	2	3	4	5	6	7	8	9
		BOST	NY	DC	MIAM	CHIC	SEAT	SF	LA	DENV
1	BOSTON	0	206	429	1504	963	2976	3095	2979	1949
2	NY	206	0	233	1308	802	2815	2934	2786	1771
3	DC	429	233	0	1075	671	2684	2799	2631	1616
4	MIAMI	1504	1308	1075	0	1329	3273	3053	2687	2037
5	CHICAGO	963	802	671	1329	0	2013	2142	2054	996
6	SEATTLE	2976	2815	2684	3273	2013	0	808	1131	1307
7	SF	3095	2934	2799	3053	2142	808	0	379	1235
8	LA	2979	2786	2631	2687	2054	1131	379	0	1059
9	DENVER	1949	1771	1616	2037	996	1307	1235	1059	0

Output:



ISOMAP

Goal: Find projection onto *nonlinear* manifold

1. Construct neighborhood graph G:
   For all x_i, x_j
   If distance$(x_i, x_j) < \epsilon$
   Then add edge (x_i, x_j) to G
2. Compute shortest distances along graph $\delta_G(x_i, x_j)$ (e.g., by Floyd's algorithm)
3. Apply multidimensional scaling to $\delta_G(x_i, x_j)$

Dijkstra's algorithm ($O(kn^2 \log n)$) and Floyd's Algorithm ($O(n^3)$)

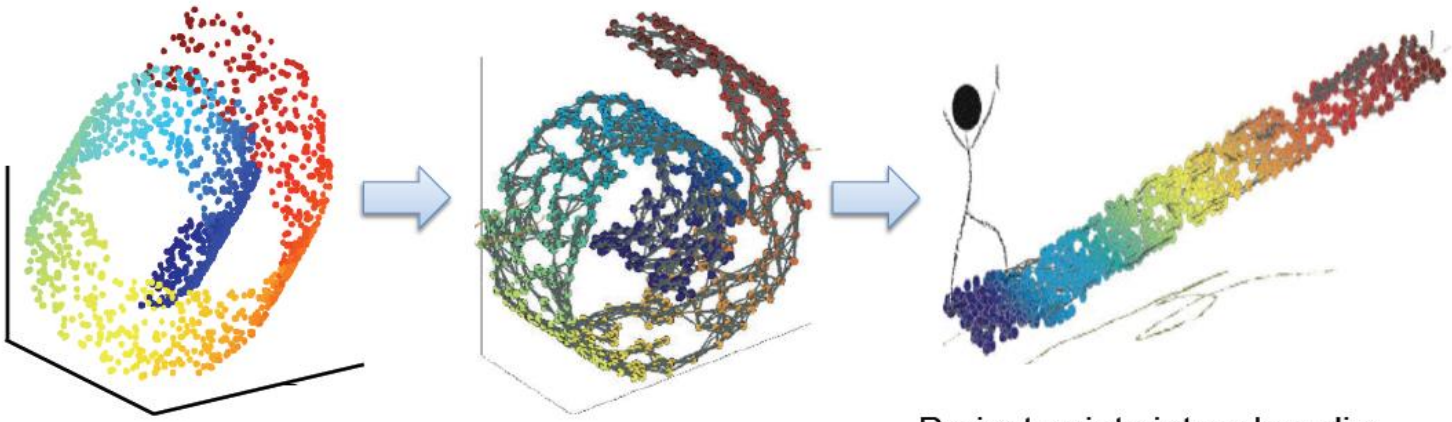
MDS on "geodesic" distance matrix

ISOMAP versus LLE

ISOMAP	LLE
MDS on geodesic distance matrix	local PCA and global eigen-decomposition
Global approach	Local approach
might not work for nonconvex manifolds with holes	ok with nonconvex manifolds with holes

Laplacian Eigenmaps

- Linear methods
  - Lower-dimensional linear projection that preserves distances between all points
- Laplacian Eigenmaps
  - key idea: preserve **local** information only



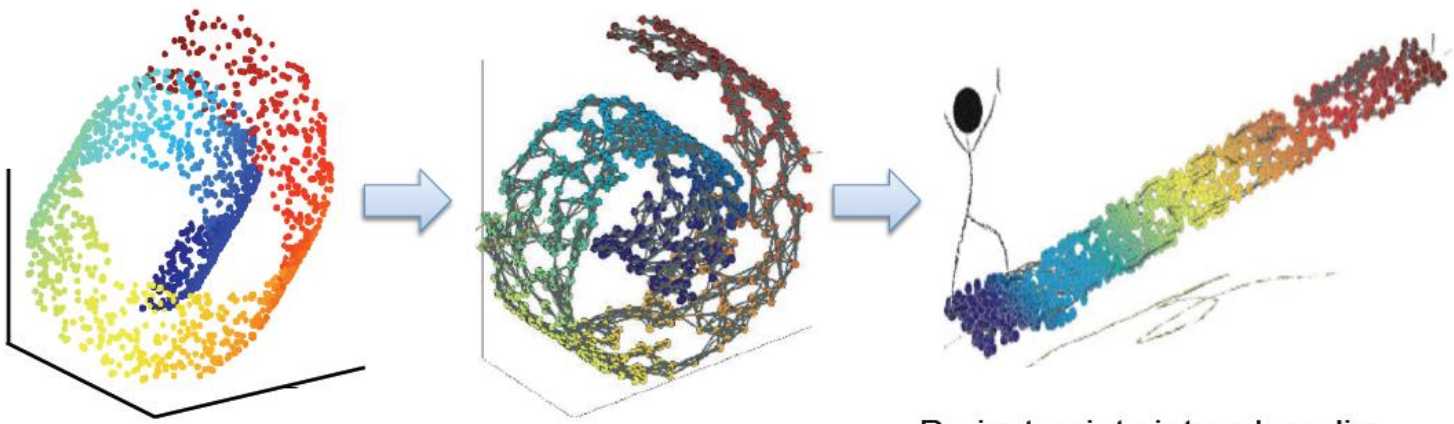
Construct graph from data points (capture local information)

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

(Belkin, Mikhail, and Partha Niyogi. 2001)

Laplacian Eigenmaps

- Step 1 – Adjacency Graph Construction
- Step 2 – Choosing the Weights
- Step 3 – Computing Eigenmaps



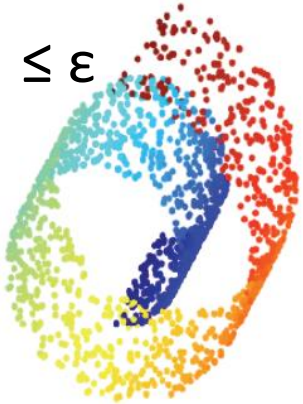
Construct graph from data points (capture local information)

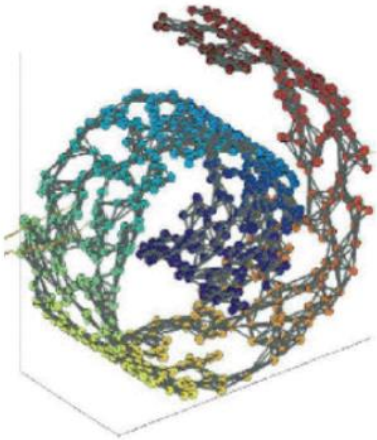
Project points into a low-dim space using "eigenvectors of the graph"

Step 1 – Adjacency Graph Construction

- G(V, E, W)
  - V: vertices (data points), E: edges, W: weights
- Methods to construct adjacency graph
  - (1) **ε – neighborhood graph**

    E – Edge (i,j) presents if $||x_i - x_j|| \leq \varepsilon$





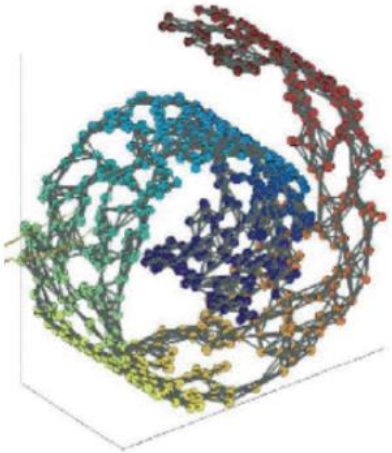
  - (2) **k-NN graph**

    E – Edge (i,j) presents if either i or j is among the other's k-nearest neighbors

Step 2 – Choosing the Weights

- G(V, E, W)
  - V: vertices (data points), E: edges, W: weights
- Methods for choosing the weights (W)
  - (1) parameter-free:
    Wij = 1 if the edge (i,j) presents, 0 otherwise
  - (2) Gaussion kernel function / heat kernel

$$W_{ij} = e^{-\frac{\|x_i - x_j\|^2}{2\sigma^2}}$$



Objective of Laplacian Eigenmaps

- **Original Representation → Transformed Representation**

data point		projections
xi	→	(f1(i), …, fd(i))
(D-dimensional vector)		(d-dimensional vector)

- Basic Idea:
  - Find **f** such that, if xi is close to xj in the adjacency graph (i.e. Wij is large), then the projections of the points f(i) and f(j) are close in the embedding space
- **Objective**

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

Step 3 – Computing Eigenmaps

- Define **L** as the graph Laplacian: **L = D – W**
  where D – Diagonal matrix = $\mathrm{diag}(D_{11}, \ldots, D_{nn})$ $\quad D_{ii} = \sum_j W_{ij}$
- The objective can be re-formulated as

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

  – Similar to PCA except that $\mathbf{X}\mathbf{X}^\top$ replaced by L
- The solution is equivalent to finding eigenvectors of the Graph Laplacian

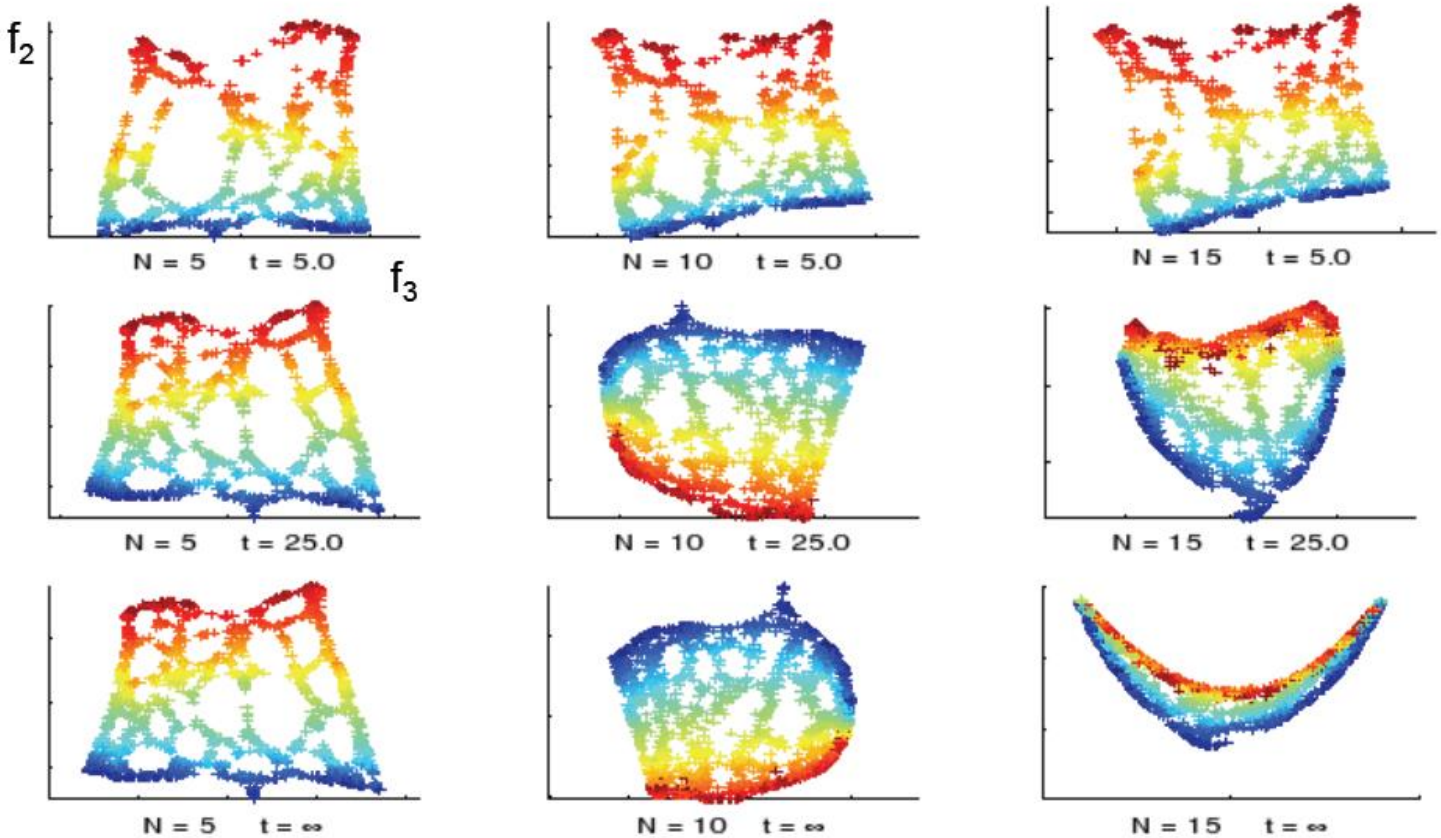
$$\boxed{\mathbf{L}\mathbf{f} = \lambda \mathbf{f}}$$

Ordered eigenvalues $0 = \lambda 0 \leq \lambda 1 \leq \lambda 2 \leq \lambda 3 \leq \ldots \leq \lambda n$

- To embed data points in d--dim space, project data points onto eigenvectors associated with λ1, λ2, …, λd

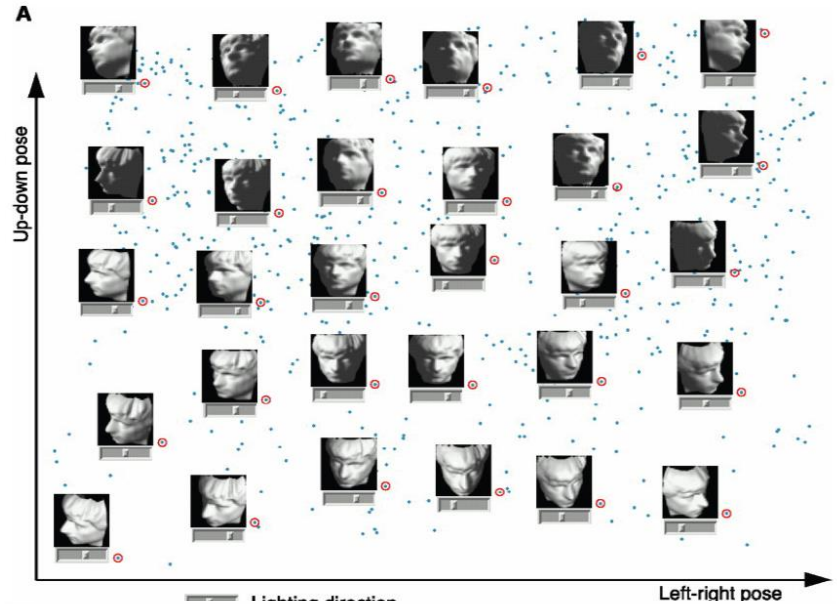
$$\mathbf{x}_i \rightarrow (\mathbf{f}_1(i), \ldots, \mathbf{f}_d(i))$$

Example: unrolling the swiss roll



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

More examples on real data

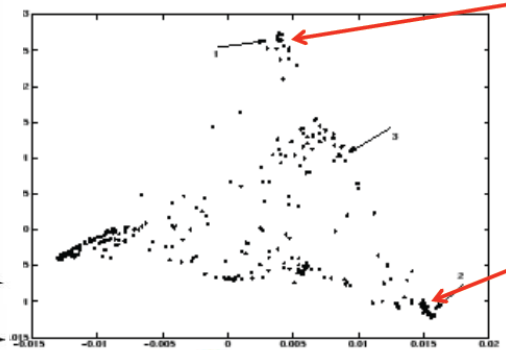


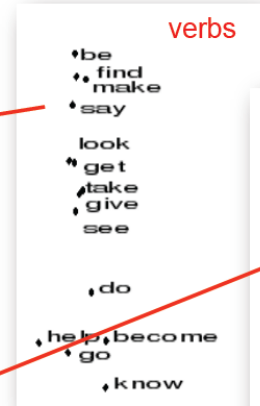
Understanding syntactic structure of words

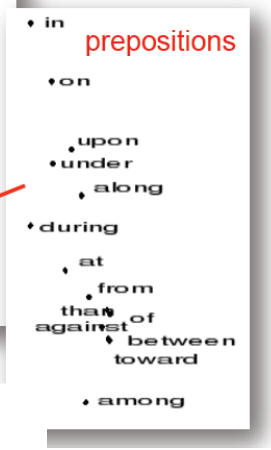
300 most frequent words of Brown corpus

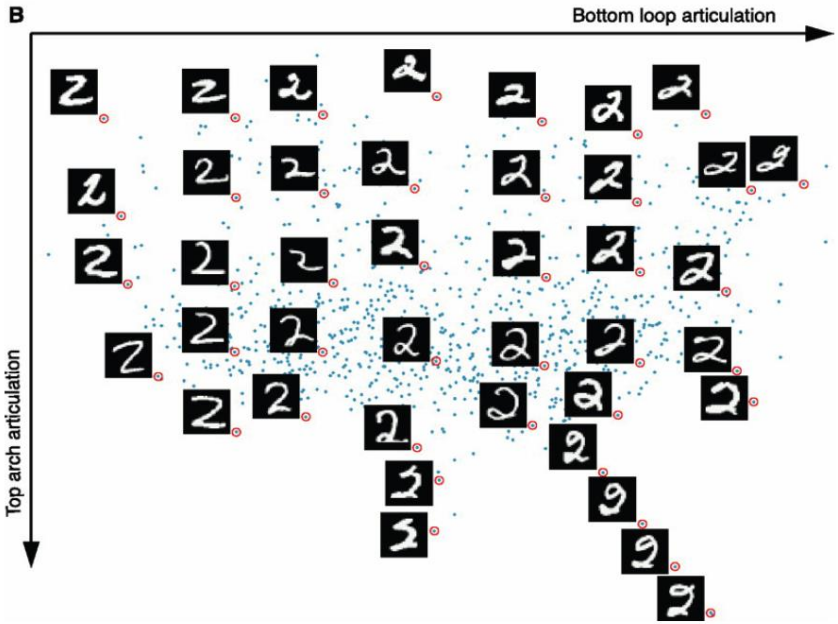
Information about the frequency of its left and right neighbors (600 Dimensional space.)

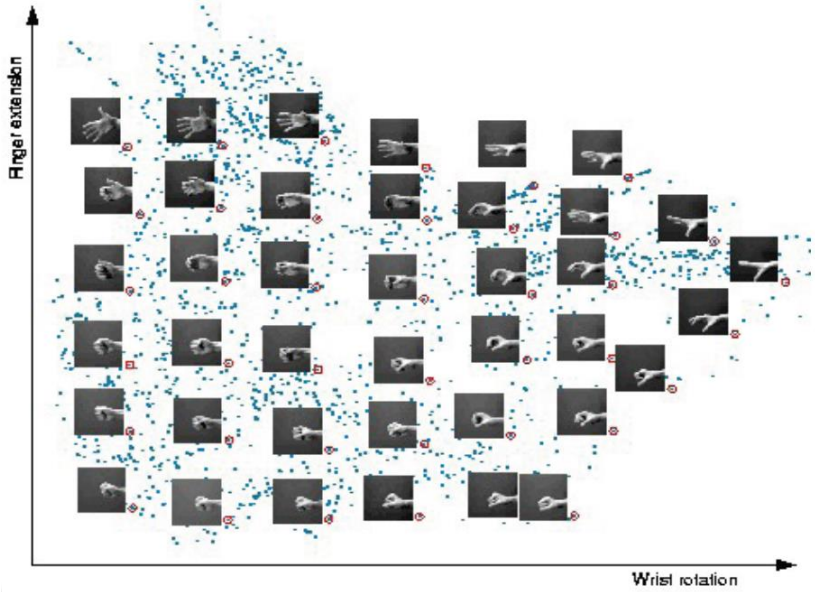
The algorithm run with N = 14, t = 1









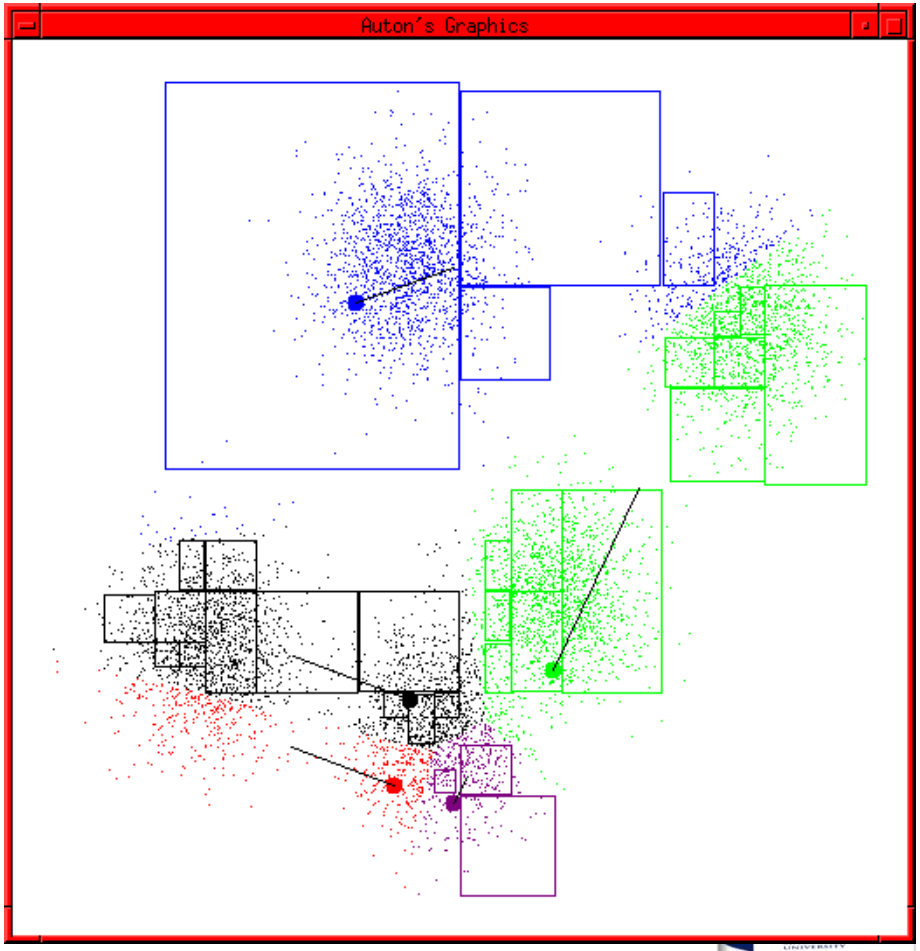


PCA vs. Laplacian Eigenmaps

PCA	Laplacian Eigenmaps
Linear Embedding	Nonlinear Embedding
Based on largest eigenvectors of DxD correlation matrix $\Sigma = \mathbf{X}\mathbf{X}^\top$ between original features	Based on smallest eigenvectors Of n x n Laplacian matrix L = D – W between data points
Eigenvectors give latent feature space: embedding of points are obtained by projecting features onto the latent subspace $\mathbf{z} = \mathbf{U}^\top \mathbf{x}$	Eigenvectors directly output embedding of data Points $\mathbf{x}_i \rightarrow (\mathbf{f}_1(i), \ldots, \mathbf{f}_d(i))$

Improving K-means Efficiency

- Speed up K-means by efficient data structure
  - Group points by region
    - KD tree
    - SR tree
  - Key difference
    - Find the closest center for each rectangle
    - Assign all the points within a rectangle to one cluster



Mixture Model for Doc Clustering

- A set of language models
  - $\Theta = \{\theta_1, \theta_2, \ldots, \theta_K\}$

    $\theta_i = \{p(w_1 \mid \theta_i), p(w_2 \mid \theta_i), \ldots, p(w_V \mid \theta_i)\}$

Mixture Model for Doc Clustering

- A set of language models
  - $\Theta = \{\theta_1, \theta_2, \ldots, \theta_K\}$
    $\theta_i = \{p(w_1 \mid \theta_i), p(w_2 \mid \theta_i), \ldots, p(w_V \mid \theta_i)\}$

- Probability $p(d = d_i)$

$$p(d = d_i) = \sum_{\theta_j} p(d = d_i, \theta = \theta_j)$$

Mixture Model for Doc Clustering

- A set of language models
  - $\Theta = \{\theta_1, \theta_2, \ldots, \theta_K\}$

    $\theta_i = \{p(w_1 \mid \theta_i), p(w_2 \mid \theta_i), \ldots, p(w_V \mid \theta_i)\}$

- Probability $p(d = d_i)$

$$p(d = d_i) = \sum_{\theta_j} p(d = d_i, \theta = \theta_j)$$

$$= \sum_{\theta_j} p(\theta = \theta_j) p(d = d_i \mid \theta = \theta_j)$$

Mixture Model for Doc Clustering

- A set of language models
  - $\Theta = \{\theta_1, \theta_2, \ldots, \theta_K\}$
    $\theta_i = \{p(w_1 \mid \theta_i), p(w_2 \mid \theta_i), \ldots, p(w_V \mid \theta_i)\}$

- Probability $p(d = d_i)$

$$p(d = d_i) = \sum_{\theta_j} p(d = d_i, \theta = \theta_j)$$

$$= \sum_{\theta_j} p(\theta = \theta_j) p(d = d_i \mid \theta = \theta_j)$$

$$\propto \sum_{\theta_j} p(\theta = \theta_j) \prod_{k=1}^{V} \left[p(w_k \mid \theta_j) \right]^{tf(w_k, d_i)}$$

Introduce hidden variable z_{ij}
z_{ij}: document d_i is generated by the j-th language model θ_j.

Learning a Mixture Model

E-Step

$$E[z_{ij}] = p(\theta = \theta_j \mid d = d_i)$$

$$= \frac{p(d = d_i \mid \theta = \theta_j)\, p(\theta = \theta_j)}{\sum_{n=1}^{K} p(d = d_i \mid \theta = \theta_n)\, p(\theta = \theta_n)}$$

$$= \frac{\prod_{m=1}^{V} \left[p(w_m \mid \theta_j) \right]^{tf(w_k, d_i)} p(\theta = \theta_j)}{\sum_{n=1}^{K} \prod_{m=1}^{V} \left[p(w_m \mid \theta_n) \right]^{tf(w_k, d_i)} p(\theta = \theta_n)}$$

K: number of language models

Learning a Mixture Model

M-Step

$$p(w_i \mid \theta_j) \leftarrow \frac{\sum_{k=1}^{N} E[z_{ij}]\, tf(w_i, d_k)}{\sum_{k=1}^{N} E[z_{ij}]\, |d_k|}$$

$$p(\theta = \theta_j) \leftarrow \frac{1}{N} \sum_{i=1}^{N} E[z_{ij}]$$

N: number of documents

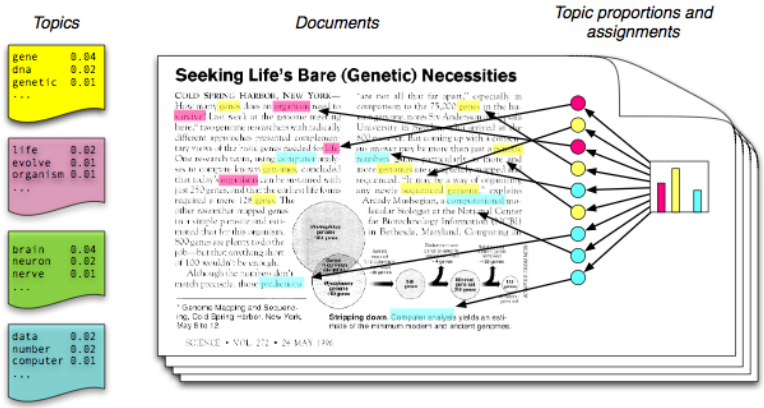
Examples of Mixture Models

"segment 1"	"segment 2"	"matrix 1"	"matrix 2"	"line 1"	"line 2"
imag	speaker	robust	manufactur	constraint	alpha
SEGMENT	speech	MATRIX	cell	LINE	redshift
texture	recogni	eigenvalu	part	match	LINE
color	signal	uncertainti	MATRIX	locat	galaxi
tissue	train	plane	cellular	imag	quasar
brain	hmm	linear	famili	geometr	absorp
slice	source	condition	design	impos	high
cluster	speakerind.	perturb	machinepart	segment	ssup
mri	SEGMENT	root	format	fundament	densiti
volume	sound	suffici	group	recogn	veloc

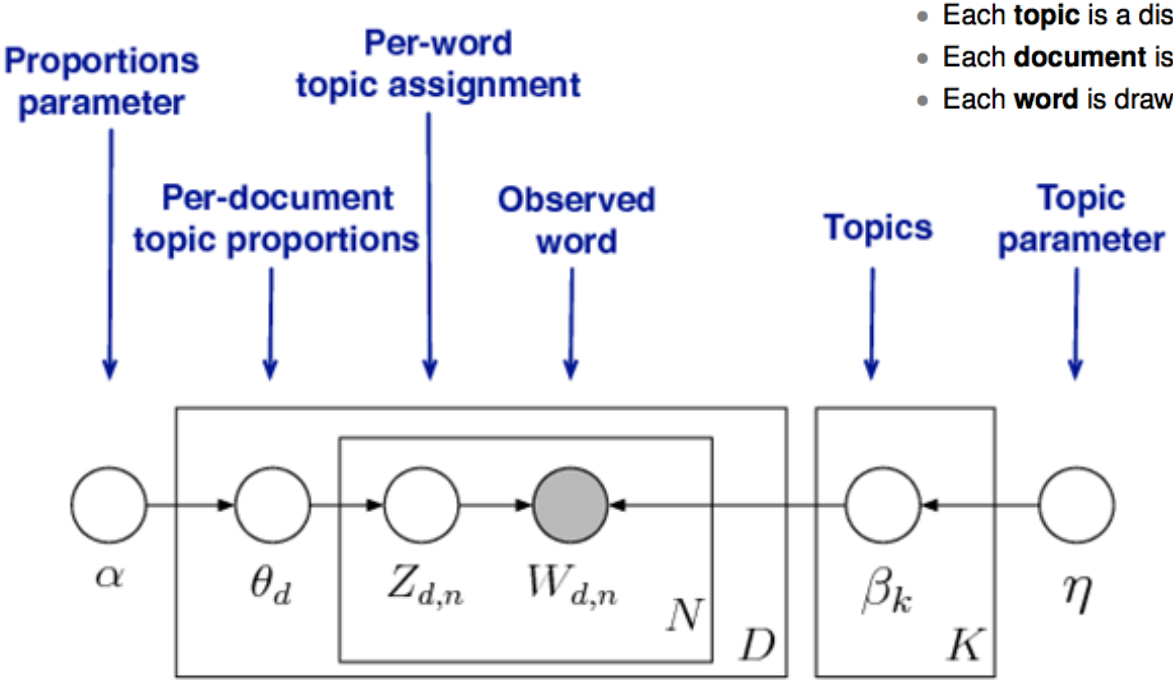
Other Mixture Models

- Probabilistic latent semantic index (PLSI)
- Latent Dirichlet Allocation (LDA)

LDA as a graphical model



- Each **topic** is a distribution over words
- Each **document** is a mixture of corpus-wide topics
- Each **word** is drawn from one of those topics



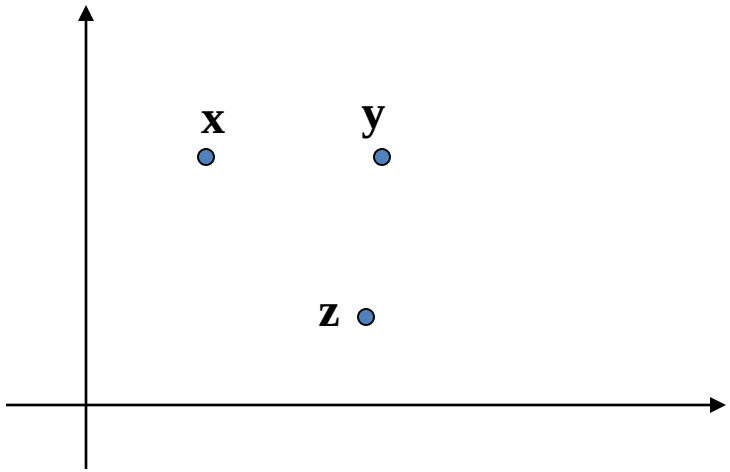
$$p(\beta,\theta,\mathbf{z},\mathbf{w}) = \left(\prod_{i=1}^{K} p(\beta_i|\eta)\right)\left(\prod_{d=1}^{D} p(\theta_d|\alpha)\prod_{n=1}^{N} p(z_{d,n}|\theta_d)p(w_{d,n}|\beta_{1:K},z_{d,n})\right)$$

Spectral Clustering: Problem (I)

- Both k-means and mixture models need to compute centers of clusters and explicit distance measurement
  - Given strange distance measurement, the center of clusters can be hard to compute

E.g.,

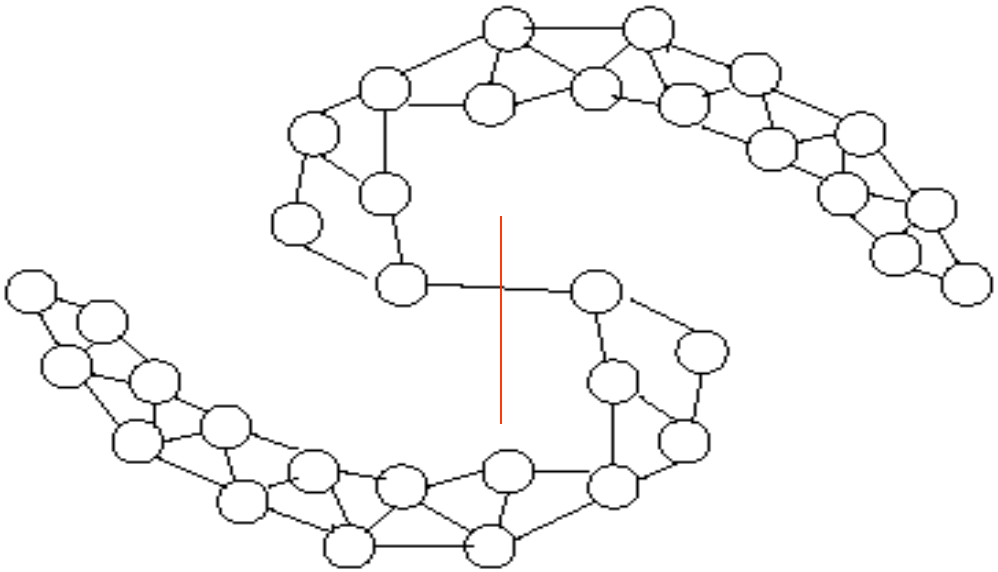
$$\|\vec{x}-\vec{x}'\|_\infty = \max\left(\left|x_1-x_1'\right|,\left|x_2-x_2'\right|,\ldots,\left|x_n-x_n'\right|\right)$$



$$\|\mathbf{x}-\mathbf{y}\|_\infty = \|\mathbf{x}-\mathbf{z}\|_\infty$$

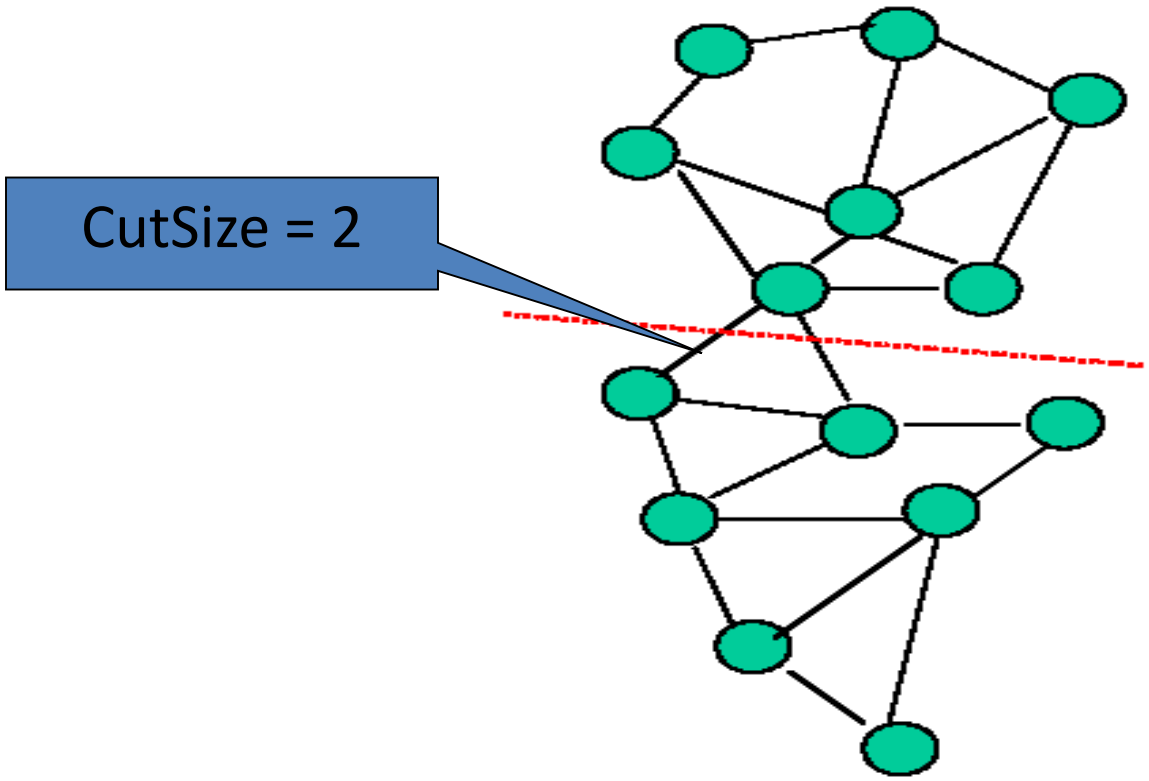
Problem: Motivation (I)

- Both k-means and mixture models look for compact clustering structures
  - In some cases, connected clustering structures are more desirable



Graph Partition

- MinCut: bipartite graphs with minimal number of cut edges



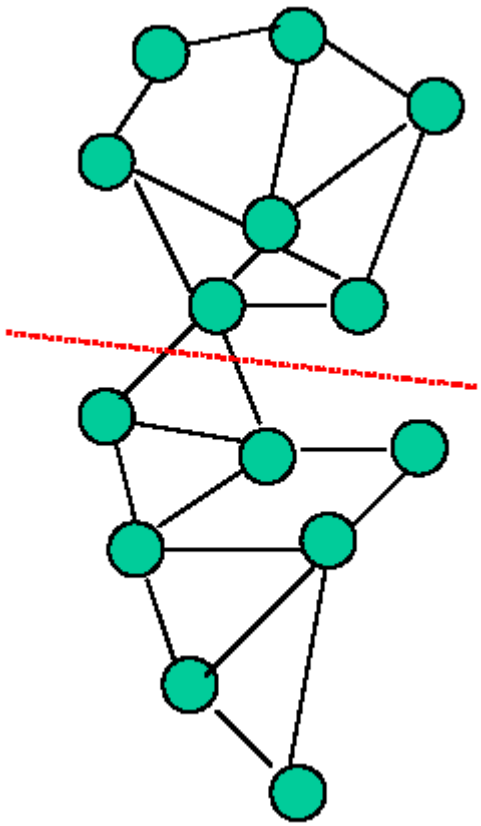
2-way Spectral Graph Partitioning

- Weight matrix **W**
  - $w_{i,j}$: the weight between two vertices i and j
- Membership vector **q**

$$q_i = \begin{cases} 1 & i \in \text{Cluster } A \\ -1 & i \in \text{Cluster } B \end{cases}$$

$$\mathbf{q} = \arg\min_{\mathbf{q} \in [-1,1]^n} CutSize$$

$$CutSize = J = \frac{1}{4} \sum_{i,j} \left(q_i - q_j \right)^2 w_{i,j}$$



Solving the Optimization Problem

- Directly solving the above problem requires combinatorial search → exponential complexity

- How to reduce the computation complexity?

$$\mathbf{q} = \arg\min_{\mathbf{q}\in[-1,1]^n} \frac{1}{4}\sum_{i,j}\left(q_i - q_j\right)^2 w_{i,j}$$

Relaxation Approach

- Key difficulty: q_i has to be either –1, 1
  - Relax q_i to be any real number
  - Impose constraint $\sum_{i=1}^{n} q_i^2 = n$

$$d_i \equiv \sum_j w_{i,j}$$

$$D \equiv \left[d_i \delta_{i,j} \right]$$

$$J = \frac{1}{4}\sum_{i,j}\left(q_i - q_j\right)^2 w_{i,j} = \frac{1}{4}\sum_{i,j}\left(q_i^2 + q_j^2 - 2q_i q_j\right) w_{i,j}$$

$$= \frac{1}{4}\sum_i 2q_i^2 \left(\sum_j w_{i,j}\right) - \frac{1}{4}\sum_{i,j} 2q_i q_j w_{i,j}$$

$$= \frac{1}{2}\sum_i q_i^2 d_i - \frac{1}{2}\sum_{i,j} q_i q_j w_{i,j} = \frac{1}{2}\sum_i q_i \left(d_i \delta_{i,j} - w_{i,j}\right) q_j$$

$$J = \mathbf{q}^T (\mathbf{D} - \mathbf{W}) \mathbf{q}$$

Relaxation Approach

$$\mathbf{q}^* = \arg\min_{\mathbf{q}} J = \arg\min_{\mathbf{q}} \mathbf{q}^T (\mathbf{D} - \mathbf{W})\mathbf{q}$$

$$\text{subject to } \sum_k q_k^2 = n$$

Relaxation Approach

$$\mathbf{q}^* = \arg\min_{\mathbf{q}} J = \arg\min_{\mathbf{q}} \mathbf{q}^T (\mathbf{D} - \mathbf{W})\mathbf{q}$$

$$\text{subject to } \sum_k q_k^2 = n$$

- Solution: the second minimum eigenvector for **D**-**W**

$$(D - W)\mathbf{q} = \lambda_2 \mathbf{q}$$

Graph Laplacian

- **L** is semi-positive definitive matrix
  - For Any **x,** we have $\mathbf{x}^T\mathbf{L}\mathbf{x} \geq 0$, why?
  - Minimum eigenvalue $\lambda_1 = 0$ (what is the eigenvector?)
  - $$\mathbf{L} = \mathbf{D} - \mathbf{W}: \ \mathbf{W} = \left[w_{i,j} \right], \ \mathbf{D} = \left[\delta_{i,j} \left(\sum_j w_{i,j} \right) \right]$$
  - The second minimum eigenvalue λ_2 gives the best bipartite graph

$$0 = \lambda_1 < \lambda_2 < \lambda_3 \ldots < \lambda_k$$

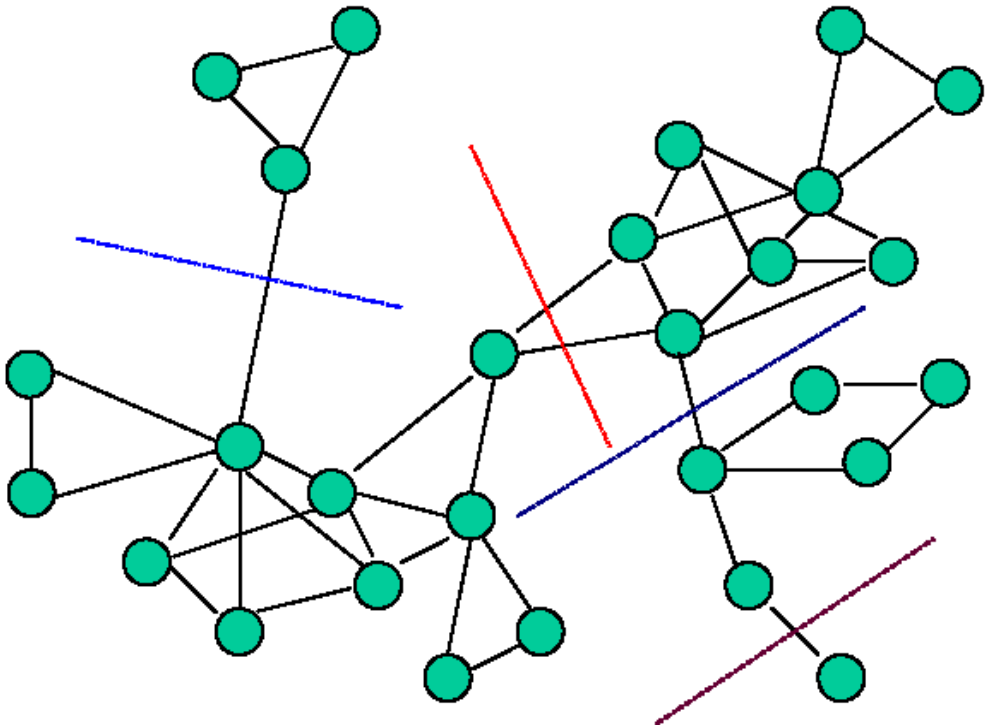
Recovering Partitions

- Due to the relaxation, **q** can be any number (not just –1 and 1)
- How to construct partition based on the eigenvector?

- Simple strategy: $A = \{i \mid q_i < 0\},\ B = \{i \mid q_i \geq 0\}$

Spectral Clustering

- Minimum cut does not balance the size of bipartite graphs



Normalized Cut (Shi & Malik, 1997)

- Minimize the similarity between clusters and meanwhile maximize the similarity within clusters

$$s(A,B) \equiv \sum_{i \in A} \sum_{j \in B} w_{i,j},\ d_A = \sum_{i \in A} d_i,\ d_B = \sum_{i \in B} d_i$$

$$d \equiv \sum_j d_j$$

$$J = \frac{s(A,B)}{d_A} + \frac{s(A,B)}{d_B}$$

$$J = \frac{s(A,B)}{d_A} + \frac{s(A,B)}{d_B} = \sum_{i \in A} \sum_{j \in B} w_{i,j} \frac{d_B + d_A}{d_A d_B}$$

$$= \sum_{i \in A} \sum_{j \in B} w_{i,j} \frac{(d_B + d_A)^2}{d_A d_B d}$$

$$= \sum_i \sum_j w_{i,j} \left(q_i - q_j\right)^2$$

$$q(i) = \begin{cases} \sqrt{d_B / d_A d} & \textbf{if } i \in A \\ -\sqrt{d_A / d_B d} & \textbf{if } i \in B \end{cases}$$

Normalized Cut

$$J = \sum_i \sum_j w_{i,j} \left(q_i - q_j \right)^2 = \mathbf{q}^T (\mathbf{D} - \mathbf{W}) \mathbf{q}$$

$$q_i = \begin{cases} \sqrt{d_B / d_A d} & \text{if } i \in A \\ -\sqrt{d_A / d_B d} & \text{if } i \in B \end{cases}$$

Normalized Cut

- Relax **q** to real value under the constraint

$$J = \sum_i \sum_j w_{i,j} \left(q_i - q_j \right)^2 = \mathbf{q}^T (\mathbf{D} - \mathbf{W}) \mathbf{q}$$

$$q_i = \begin{cases} \sqrt{d_B / d_A d} & \text{if } i \in A \\ -\sqrt{d_A / d_B d} & \text{if } i \in B \end{cases}$$

$$q^T D q = 1, q^T D e = 0$$

- Solution: $(D - W) q = \lambda D q$

Image Segmentation