

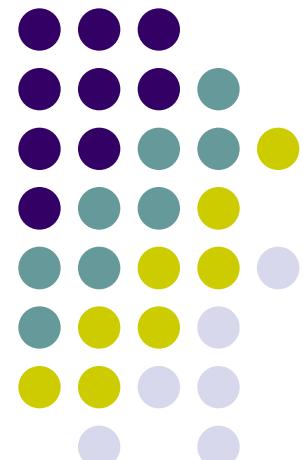
# Distance and Similarity

Hongxin Zhang

[zhx@cad.zju.edu.cn](mailto:zhx@cad.zju.edu.cn)

State Key Lab of CAD&CG, ZJU

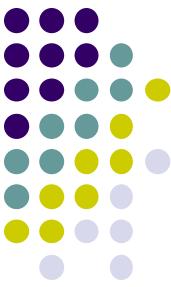
2020-03-10





# In the last lesson

- Data driven decomposition:
  - Data driven curve fitting
  - Singular value decomposition (the power of orthogonal basis)
  - A sort of spectral analysis
- PCA and its related techniques are very useful

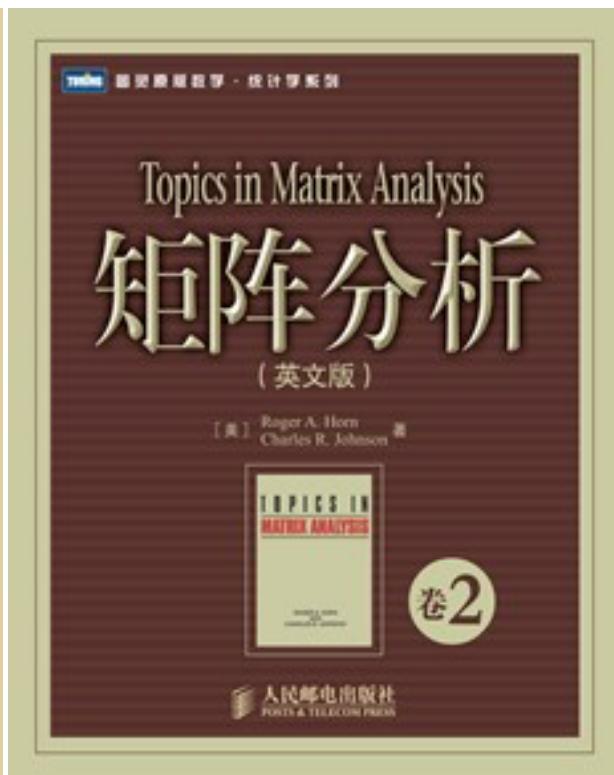
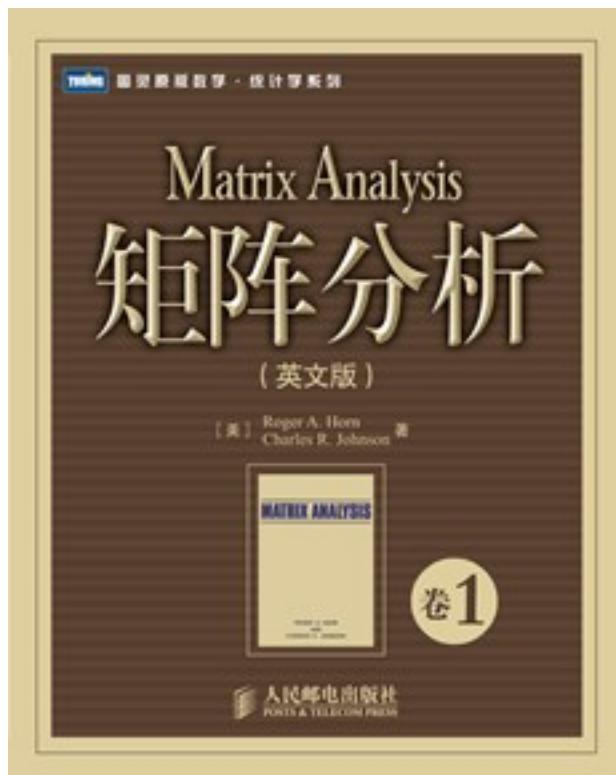


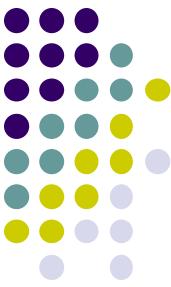
# In the last lesson

- Mathematical concepts and techniques
  - Least squares (LSQ)
  - Curve fitting
  - Norm (范数) and inner product (内积)
  - Singular value decomposition
  - Eigen vectors and eigen-values
  - Low rank matrix approximation and decomposition



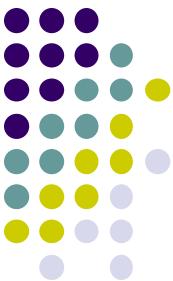
# Recommended textbooks





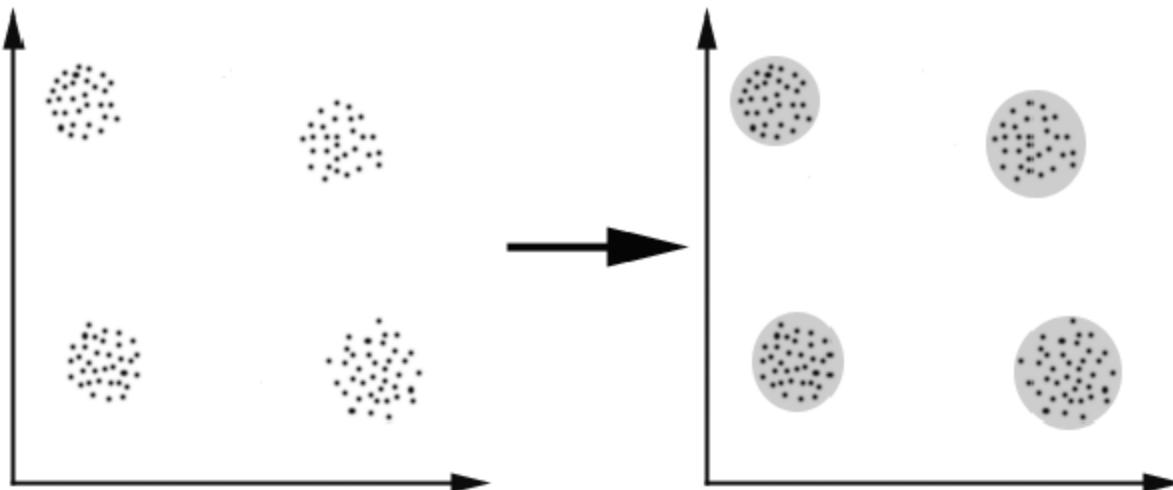
# Today's Talk

- Pre-processing: Distance / metric learning
  - ISO-map
  - LLE
- What is similarity? How to Clustering
  - Spectral based
  - E-M based
    - MOG and K-means
  - Mean shift

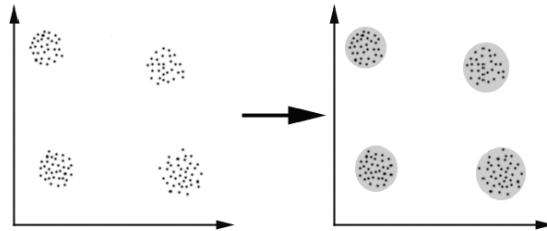


# Clustering

- Given set of data points, group them, find the overall structure
- **Unsupervised** learning
- Learn the similarity. Which patient are similar? (or customers, faces, earthquakes, ...)



# Distance



- Given  $n$ -dimensional vector  $\mathbf{x}, \mathbf{y}$
- Euclidian ( $L^2$  distance)

$$dist(\mathbf{x}, \mathbf{y}; 2) = \left( \sum_{i=1}^n (x_i - y_i)^2 \right)^{1/2}$$

- $L^1$  distance

$$dist(\mathbf{x}, \mathbf{y}; 1) = \sum_{i=1}^n |x_i - y_i|$$

- $L^p$  distance (Minkowsky)

$$dist(\mathbf{x}, \mathbf{y}; p) = \left( \sum_{i=1}^n (x_i - y_i)^p \right)^{1/p}$$

$$dist(\mathbf{x}, \mathbf{y}; \infty) = \max_i |x_i - y_i| \quad \text{切比雪夫}$$

# Distance, Norm and inner product



- Distance (of 2 points)

$$dist(\mathbf{x}, \mathbf{y}; 2) = \left( \sum_{i=1}^n (x_i - y_i)^2 \right)^{1/2}$$

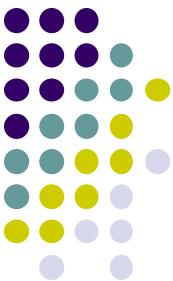
- Norm (of vector)

$$norm(\mathbf{x}; 2) := \|\mathbf{x}\|_2 = \left( \sum_{i=1}^n x_i^2 \right)^{1/2}$$

- Inner product (of 2 vectors)

$$\mathbf{x} \cdot \mathbf{y} = \sum_{i=1}^n x_i y_i$$

$$\begin{aligned} norm(\mathbf{x} - \mathbf{y}; 2) &= dist(\mathbf{x}, \mathbf{y}; 2) & dist(x, y; 2) &= (\mathbf{x} \cdot \mathbf{x} + \mathbf{y} \cdot \mathbf{y} - 2\mathbf{x} \cdot \mathbf{y})^{1/2} \\ & & &= \left( \|\mathbf{x}\|_2^2 + \|\mathbf{y}\|_2^2 - 2\mathbf{x} \cdot \mathbf{y} \right)^{1/2} \end{aligned}$$



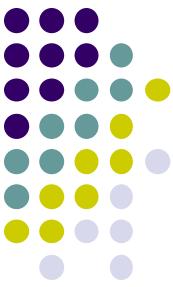
# Dimensional aware distances

- Along dimension  $j$ :

$$\bar{x}_j = \frac{1}{N} \sum_{i=1}^N x_{i,j}$$
$$R_j = \max_i x_{i,j} - \min_i x_{i,j}$$
$$S_j = \left( \frac{1}{N-1} \sum_{i=1}^N (x_{i,j} - \bar{x}_j)^2 \right)^{1/2}$$

- Normalized data:

$$x'_{i,j} = \frac{x_{i,j} - \bar{x}_j}{R_j}$$
$$x'_{i,j} = \frac{x_{i,j} - \bar{x}_j}{S_j}$$



# M-distance

- Consider the dependency of different dimensions

$$dist(\mathbf{x}, \mathbf{y}; \mathbf{M}) = (\mathbf{x} - \mathbf{y})^T \mathbf{M}^{-1} (\mathbf{x} - \mathbf{y})$$

- M is the covariance matrix of data
- Transform invariance



# Data whitening

设有均值为零的随机信号向量  $x$ ，其自相关矩阵为

$$R_x = E[xx^T] \neq I$$

很明显， $R_x$  是对称矩阵，且是非负定的（所有特征值都大于或等于0）。

现在，寻找一个线性变换  $B$  对  $x$  进行变换，即  $y = Bx$ ，使得

$$R_y = BE[xx^T]B^T = I$$

上式的含义是： $y$  的各分量是不相关的，即  $E[y_i y_j] = \delta_{ij}$ 。通常将这个过程称为“空间解相关”、“空间白化”或“球化”。 $B$  称为空间解相关矩阵（空间白化矩阵、球化矩阵）。

由  $R_x$  的性质可知，其存在特征值分解：

$$R_x = Q\Sigma Q^T$$

$Q$  是正交矩阵， $\Sigma$  是对角矩阵，其对角元素是  $R_x$  特征值。

令

$$B = \Sigma^{-1/2} Q^T \quad (1)$$

$$R_y = BE[xx^T]B^T = I$$

上式的含义是： $y$ 的各分量是不相关的，即  $E[y_iy_j] = \delta_{ij}$ 。通常将这个过程称为“空间解相关”、“空间白化”或“球化”。 $B$  称为空间解相关矩阵（空间白化矩阵、球化矩阵）。

由  $R_x$  的性质可知，其存在特征值分解：

$$R_x = Q\Sigma Q^T$$

$Q$  是正交矩阵， $\Sigma$  是对角矩阵，其对角元素是  $R_x$  特征值。

令

$$B = \Sigma^{-1/2}Q^T \quad (1)$$

则有

$$R_y = (\Sigma^{-1/2}Q^T)Q\Sigma Q^T(\Sigma^{-1/2}Q^T)^T = I$$

因此，通过矩阵  $B$  线性变换后， $y$  的各个分量变得不相关了。

对于  $R_x$  来说，特征值分解和奇异值分解是等价的，而奇异值分解的数值算法比特征值分解的数值算法具有更好的稳定性，因此一般都用奇异值分解来构造空间解相关矩阵  $B$ 。

应该注意到，“空间解相关”不能保证各分量信号之间的“独立性”，但它能够简化盲分离算法或改善分离算法的性能。

# More complex method for distance computing



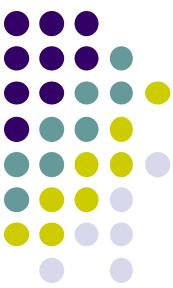
- PCA ...
- structure aware
  - Main idea:
    - Find a suitable mapping
    - Compute distance in mapped space
  - Available techniques
    - MDS + global geodesic distance: ISO-MAP(Science 2000)
    - Local distance approximation: LLE (Science 2000)
    - ...

# Classical Multi-dimensional Scaling



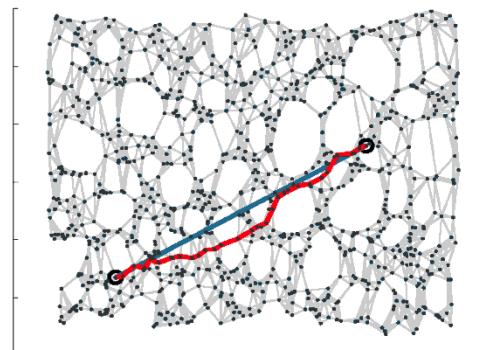
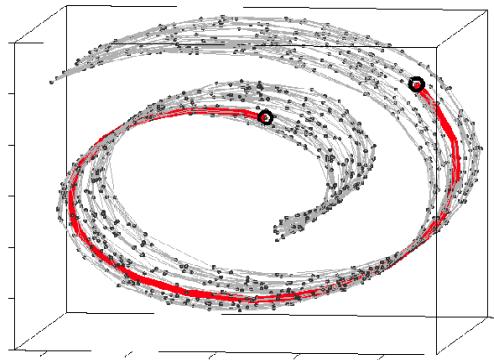
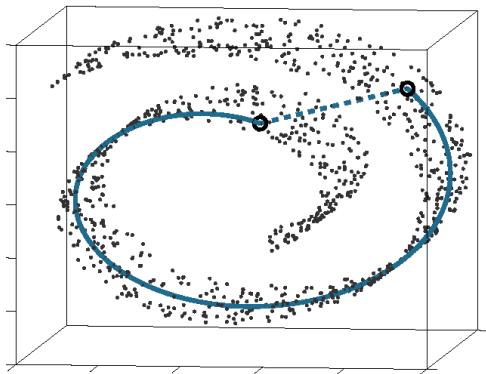
- MDS: 多维标度法
- Main idea:
  - Compute (match) distance between samples
  - Use SVD to find similarity

# Isomap: (Science 2000) Isometric feature mapping



- Preserve the **intrinsic geometry** of the data.
- Use the **geodesic distances** on manifold between all pairs.

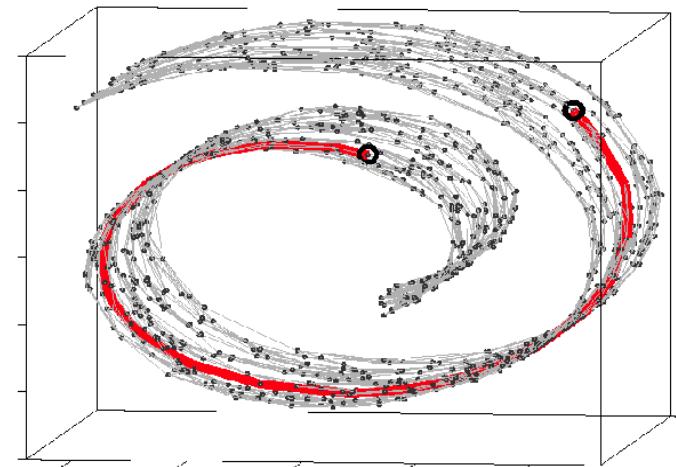
Three steps algorithm



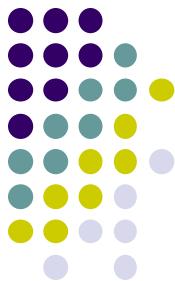
# Isomap: Construct Neighborhood Graph



- Determine which points are neighbors, based on the distances  $d(i,j)$ .
  - K nearest neighbors
  - $\epsilon$ -radius
- Create a graph  $G$ , with edges between neighbors and distance weights.



# Isomap: Compute Shortest Paths



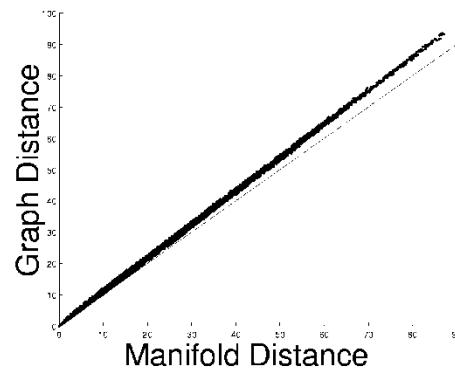
- Estimate the geodesic distances.
- Compute all-pairs shortest paths in  $G$ .
- Can be done using Floyd's algorithm,  $O(N^2 \ln N)$ .

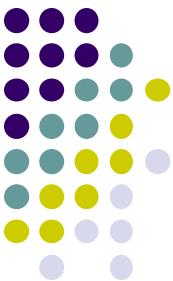
$d_G(i, j) = d(i, j)$  neighboring  $i, j$

$d_G(i, j) = \infty$  otherwise

for  $k = 1, 2, \dots, N$

$$d_G(i, j) = \min \{d_G(i, j), d_G(i, k) + d_G(k, j)\}$$





# Isomap: Construct d-dimensional Embedding

Classical MDS with  $d_G(i,j)$ ,  
minimize the cost function:

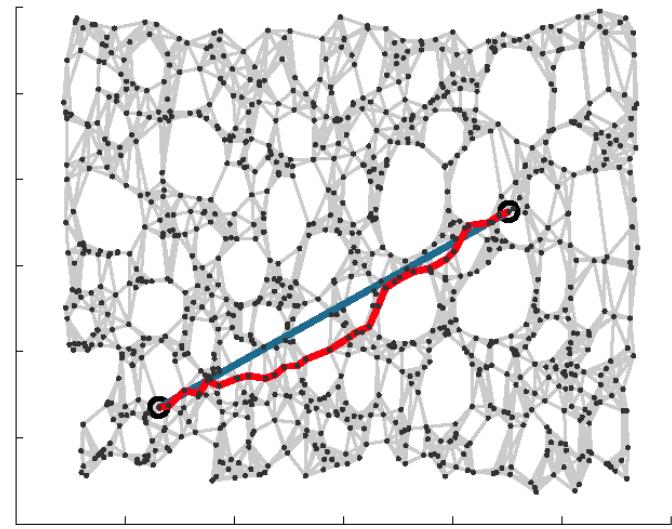
$$E = \|\tau(D_G) - \tau(D_Y)\|_{L^2}$$

where  $D_Y(i, j) = \|y_i - y_j\|$

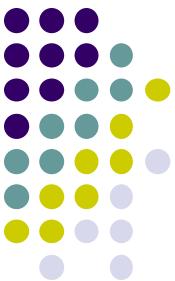
$$D_G(i, j) = d_G(i, j)$$

and

$$\tau(D) = \frac{-1}{2} \left( I - \frac{1}{N} \right) D^{-2} \left( I - \frac{1}{N} \right)$$



Solution: take top d  
eigenvectors of the  
matrix  $\tau(D_G)$



# Isomap: Classical Multi-dimensional Scaling

$$\mathbf{X}'\mathbf{X} = -\frac{1}{2}\mathbf{JEJ} \quad E: \text{Euclidian distance matrix}$$

$$\mathbf{B} = -\frac{1}{2}\mathbf{JMJ} \quad M: \text{Manifold distance matrix}$$

$$L(\hat{\mathbf{X}}) = \left\| -\frac{1}{2}\mathbf{J}(\mathbf{E} - \mathbf{M})\mathbf{J} \right\| \\ = \left\| \hat{\mathbf{X}}\hat{\mathbf{X}}' - \mathbf{B} \right\|.$$

$$\mathbf{B} = \mathbf{Q}\Lambda\mathbf{Q}' \quad \hat{\mathbf{X}} = \mathbf{Q}_+\Lambda_+^{\frac{1}{2}}$$

$$c_i = \sum_{a=1}^m x_{ia}^2$$

$$d_{ij}^2 = \sum_{a=1}^m (x_{ia} - x_{ja})^2$$

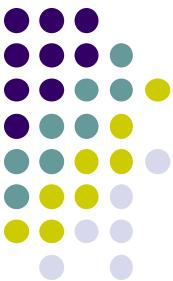
$$\mathbf{E} = \mathbf{c}\mathbf{1}' + \mathbf{1}\mathbf{c}' - 2\mathbf{XX}'$$

$$\mathbf{J} = \mathbf{I} - \frac{1}{n}\mathbf{1}\mathbf{1}'$$

$$\mathbf{B} = -\frac{1}{2}\mathbf{J}(\mathbf{c}\mathbf{1}' + \mathbf{1}\mathbf{c}' - 2\mathbf{XX}')\mathbf{J}$$

$$= -\frac{1}{2}\mathbf{J}\mathbf{c}\mathbf{0}' - \frac{1}{2}\mathbf{0}\mathbf{c}'\mathbf{J} + \mathbf{J}\mathbf{XX}'\mathbf{J} \\ = \mathbf{XX}'.$$

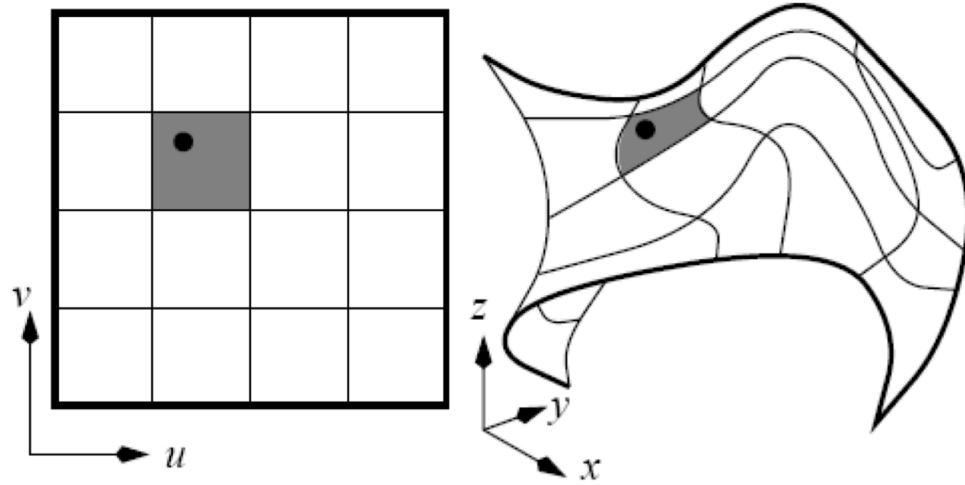
Eigen-structure analysis again



# Isomap: Classical Multi-dimensional Scaling (2D)

```
J = eye(n) - ones(n)./n;
B = -0.5 * J * M * J;
% Find largest eigenvalues+their eigenvectors:
[Q,L] = eigs(B, 2, 'LM');
% Extract the coordinates:
newy = sqrt(L(1,1)).*Q(:,1);
newx = sqrt(L(2,2)).*Q(:,2);
```

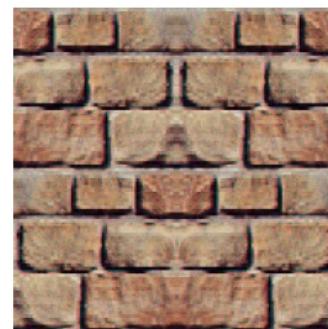
# Isomap: application texture mapping



(a)



(b)



(a)



(b)

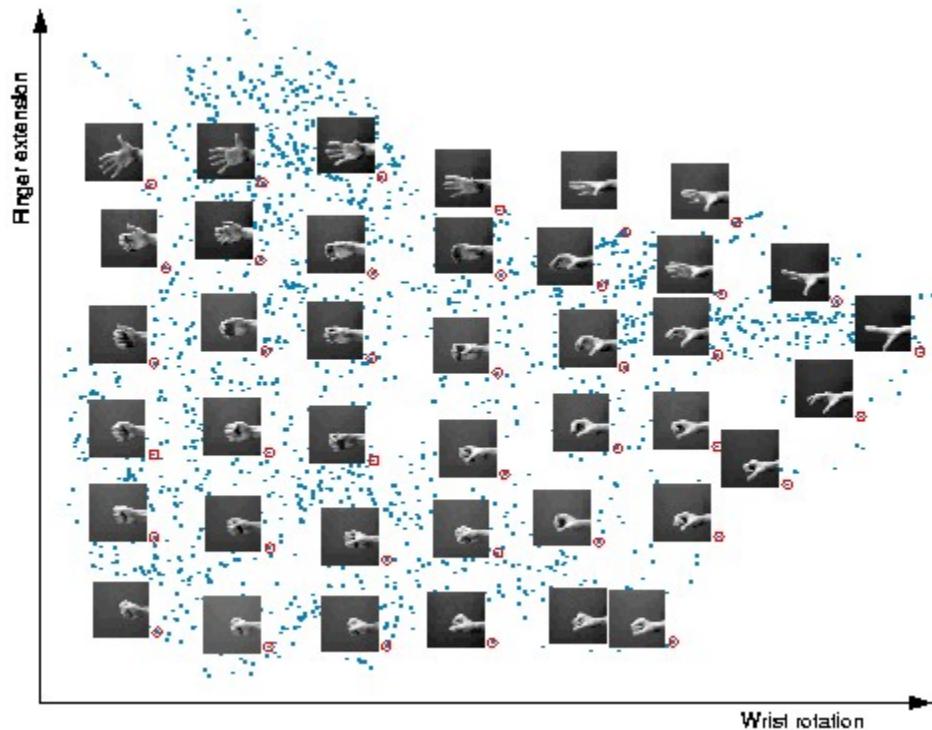
Fig. 3. An example of a face flattening. (a) A 3D reconstruction of a face. (b) The flattened texture image of the face.



# Isomap: Examples



- $N=2000$  images  
 $64 \times 64$  pixels  $K=6$



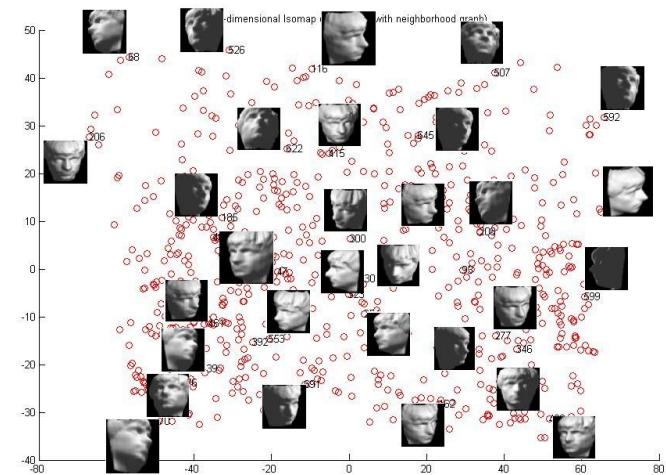
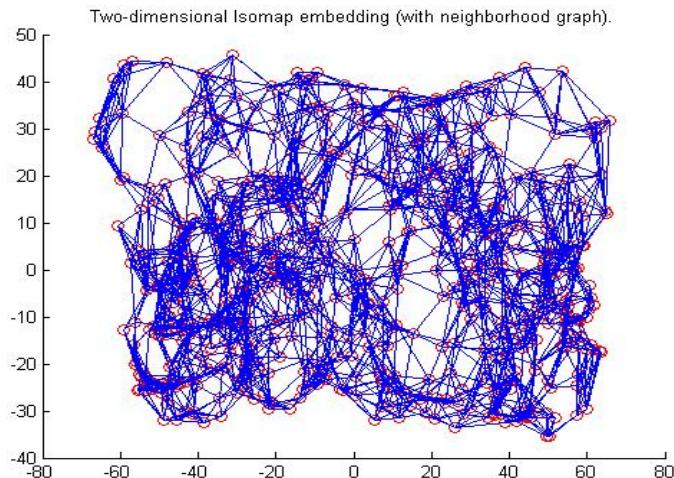
# Isomap: More Results

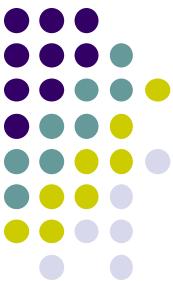


Input: 698  
images of 64x64  
 $K=7, d=2$



Outputs:

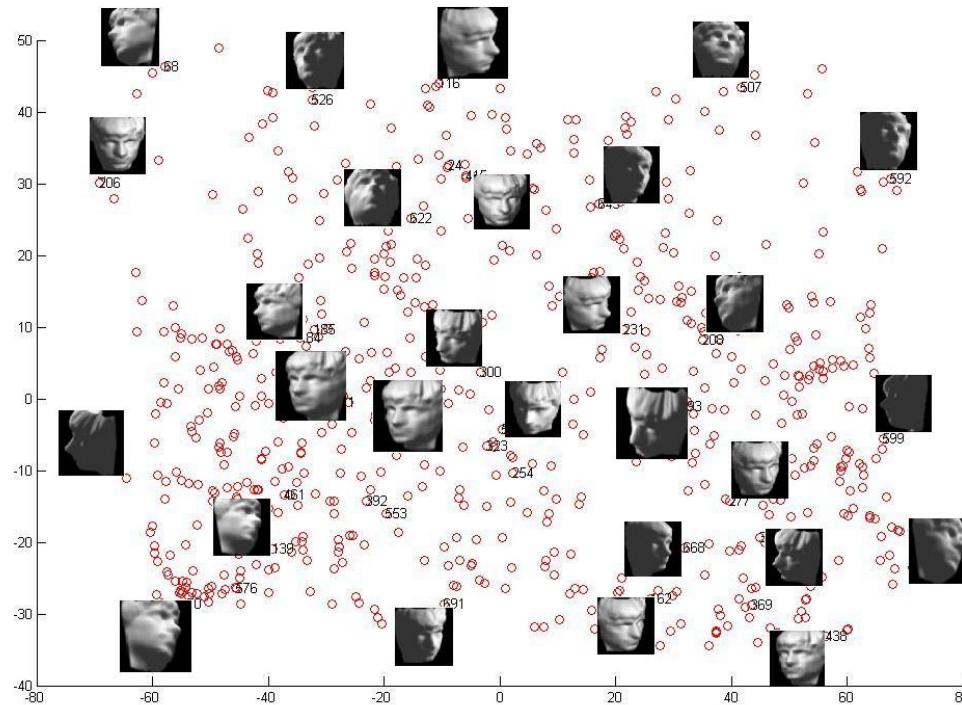




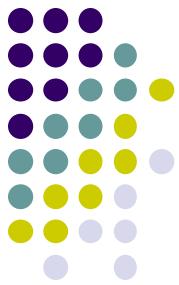
# Isomap: More Results

- Same inputs, but this time with  $d=3$

698 images of  $64 \times 64$   $K=7$



# Locally Linear Embedding (LLE)



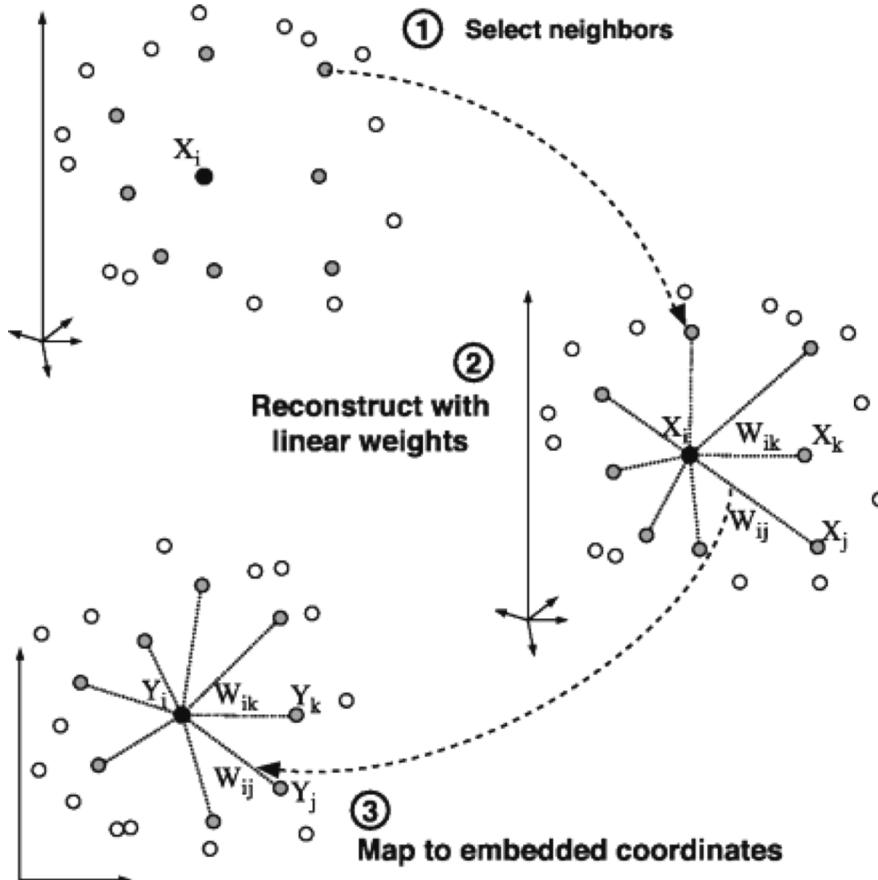
- Recovers global nonlinear structure from locally linear fits.
- Each data point and it's neighbors is expected to lie on or close to a locally linear patch.
- Each data point is constructed by it's neighbors:

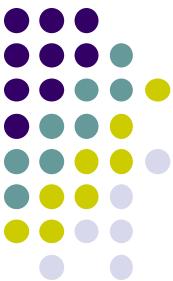
$$\hat{\vec{X}}_i = \sum_j W_{ij} \vec{X}_j$$

$W_{ij} = 0$  if  $\vec{X}_j$  is not a neighbor of  $\vec{X}_i$



# LLE – main idea





# LLE: Getting the Reconstruction Weights

- We want to minimize the error function:

$$\varepsilon(W) = \sum_i \left| \vec{X}_i - \sum_j W_{ij} \vec{X}_j \right|^2$$

- With the constraints:

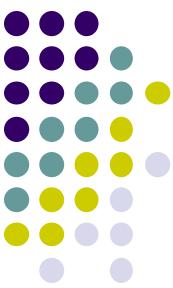
$$W_{ij} = 0 \quad \text{if } \vec{X}_j \text{ is not a neighbor of } \vec{X}_i$$

$$\sum_j W_{ij} = 1$$

- Solution (using Lagrange multipliers):

$$W_j = \sum_k C_{jk}^{-1} (\vec{X} \vec{\eta}_k + \lambda)$$

$$\lambda = 1 - \sum_{jk} C_{jk}^{-1} (\vec{X} \vec{\eta}_k) / \sum_{jk} C_{jk}^{-1}$$



# LLE:

# Find Embedded Coordinates

- Choose d-dimensional

coordinates,  $Y$ , to minimize:

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

Under:  $\sum_i \vec{Y}_i = \vec{0}$ ,  $\frac{1}{N} \sum_i \vec{Y} \vec{Y}^T = I$

Quadratic form:

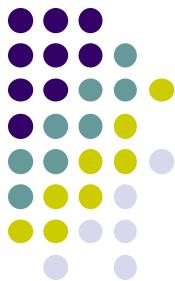
$$\phi(Y) = \sum_{ij} M_{ij} (\vec{Y}_i \vec{Y}_j)$$

where:

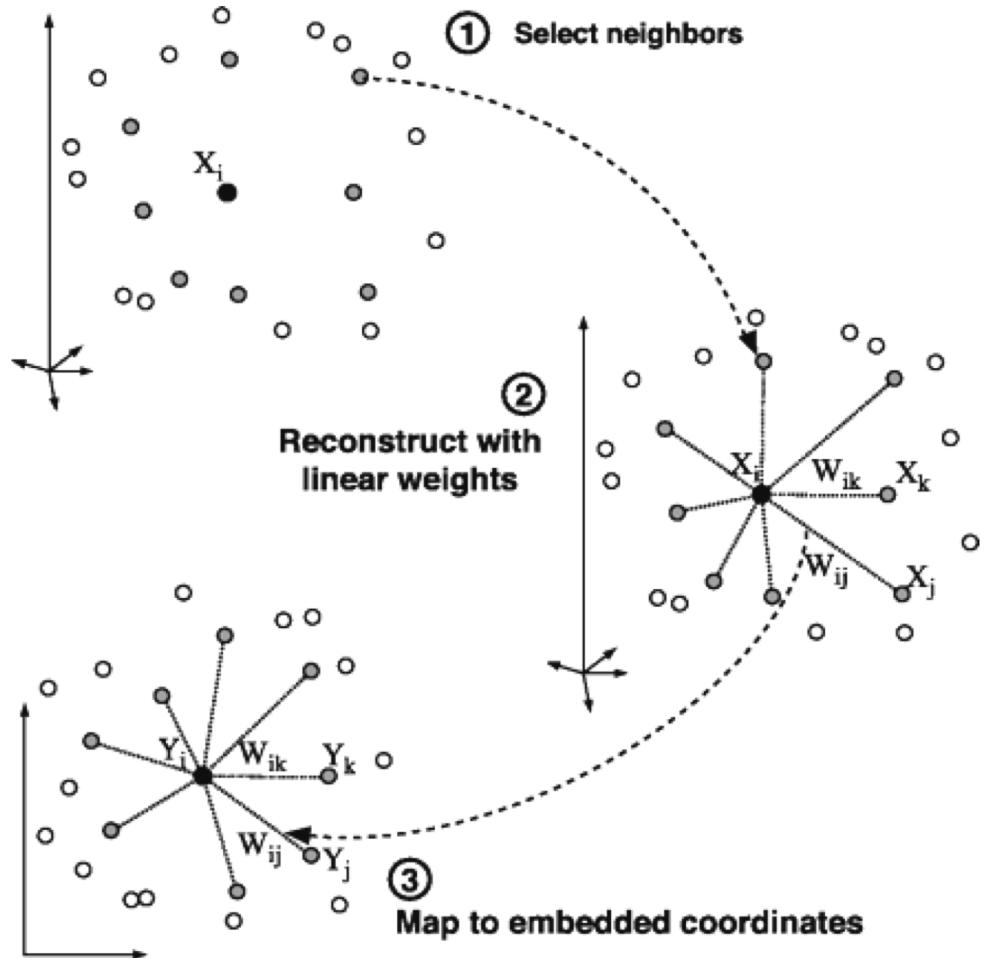
$$M = (I - W)^T (I - W)$$

- Solution: compute bottom  $d+1$  eigenvectors of  $M$ .  
(discard the last one)

# LLE: Summary



- Input: N data items in D dimension (X).
- Output:  $d < D$  dimensional embedding coordinates (Y) for the input points.





# LLE: Algorithm Pseudocode (I)

Find neighbors in X space

For i=1:N

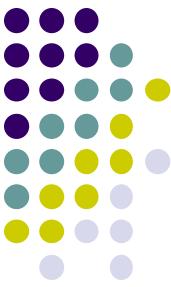
    compute the distance from  $X_i$  to every other point  $X_j$

    find the K smallest distances

    assign the corresponding points to be neighbors of  $X_i$

end

<http://www.cs.toronto.edu/~roweis/lle/algorith.html>



# LLE:

## Algorithm Pseudocode (II)

Solve for reconstruction weights W.

for i=1:N

    create matrix Z consisting of all neighbors of  $X_i$

    subtract  $X_i$  from every column of Z

    compute the local covariance  $C = Z^*Z$

    solve linear system  $C^*w = 1$  for w

    set  $W_{ij} = 0$  if j is not a neighbor of i

    set the remaining elements in the ith row of W equal to  $w/sum(w);$

end

# LLE:

## Algorithm Pseudocode (III)



Compute embedding coordinates  $Y$  using weights  $W$ .

create sparse matrix  $M = (I-W)^*(I-W)$

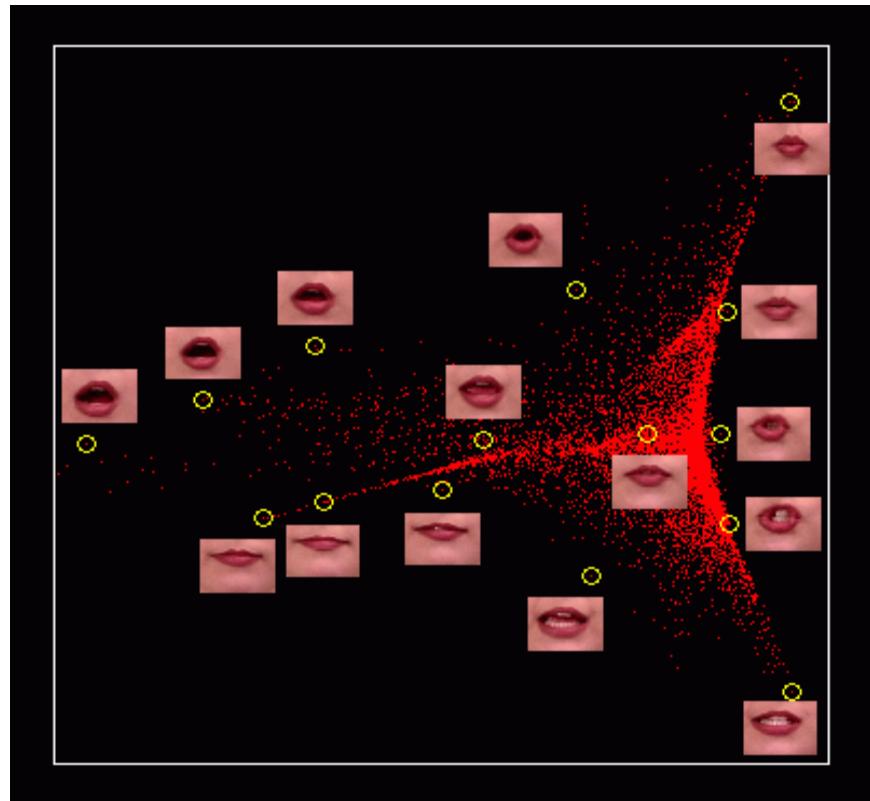
find bottom  $d+1$  eigenvectors of  $M$  (corresponding to the  $d+1$  smallest eigenvalues)

set the  $q$ -th ROW of  $Y$  to be the  $q+1$  smallest eigenvector (discard the bottom eigenvector  $[1, 1, 1, 1\dots]$  with eigenvalue zero)

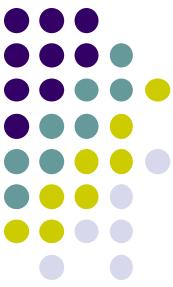
# LLE: Example



- N=8588 (RGB) images of lips of size 108x84.  
 $D=27216$
- Num of neighbors  $K=16$



3333333333333333333333333333

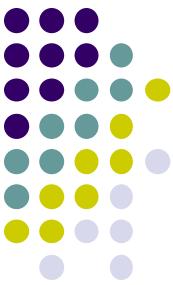


# Reference

- <https://cs.nyu.edu/~roweis/lle/publications.html>
- Nonlinear dimensionality reduction by locally linear embedding. Sam Roweis & Lawrence Saul.  
Science, v.290 [no.5500](#), Dec.22, 2000. pp.2323—2326
- The Manifold Ways of Perception. (Cognition Perspectives in same issue) H. Sebastian Seung & Daniel D. Lee.  
Science, v.290 no.5500 , Dec.22, 2000. pp.2268—2269
- An Introduction to Locally Linear Embedding.  
Lawrence Saul & Sam Roweis

# SNE

## Stochastic Neighbor Embedding



$$p_{j|i} = \frac{\exp\left(\frac{-|x_i - x_j|^2}{2\sigma_i^2}\right)}{\sum_{k \neq i} \exp\left(\frac{-|x_i - x_k|^2}{2\sigma_i^2}\right)}$$

$$p_{ij} = \frac{p_{i|j} + p_{j|i}}{2N}$$

$$q_{j|i} = \frac{e^{-|y_i - y_j|^2}}{\sum_{k \neq i} e^{-|y_i - y_k|^2}}$$

$$q_{ij} = \frac{q_{i|j} + q_{j|i}}{2N}$$

}

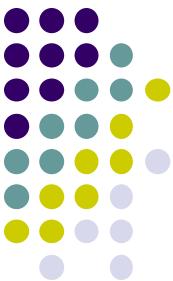
$$\min KL(P||Q) = \sum_{i,j} p_{ij} \log \frac{p_{ij}}{q_{ij}}$$



$$\frac{\partial C}{\partial y_i} = 2 \sum_j (p_{ij} - q_{ij})(y_i - y_j)$$

$$Y^t = Y^{t-1} + \eta \frac{\partial C}{\partial y_i} + \alpha(t)(Y^{t-1} - Y^{t-2})$$





# t-SNE

## t-Distributed Stochastic Neighbor Embedding

$$p_{j|i} = \frac{\exp\left(\frac{-|x_i - x_j|^2}{2\sigma_i^2}\right)}{\sum_{k \neq i} \exp\left(\frac{-|x_i - x_k|^2}{2\sigma_i^2}\right)}$$

$$q_{ij} = \frac{(1 + |y_i - y_j|)^{-1}}{\sum_{k \neq l} (1 + |y_k - y_l|)^{-1}}$$

$$q_{ij} = \frac{f(|y_i - y_j|^2)}{\sum_{k \neq l} f(|y_k - y_l|^2)} \quad f(x) = \frac{1}{1 + x^2}$$

t-Distribution

$$Y^t = Y^{t-1} + \eta \frac{\partial C}{\partial y_i} + \alpha(t)(Y^{t-1} - Y^{t-2})$$

$$p_{ij} = \frac{p_{i|j} + p_{j|i}}{2N}$$

$$q_{ij} = \frac{q_{i|j} + q_{j|i}}{2N}$$

}

$$\min KL(P||Q) = \sum_{i,j} p_{ij} \log \frac{p_{ij}}{q_{ij}}$$

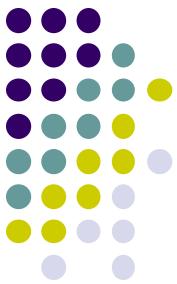


$$\frac{\partial C}{\partial y_i} = 4 \sum_j (p_{ij} - q_{ij})(y_i - y_j)(1 + |y_i - y_j|^2)^{-1}$$



# t-SNE

## t-Distributed Stochastic Neighbor Embedding



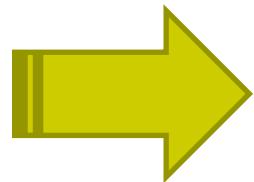
9

SIMPLE DEMO



# t-SNE

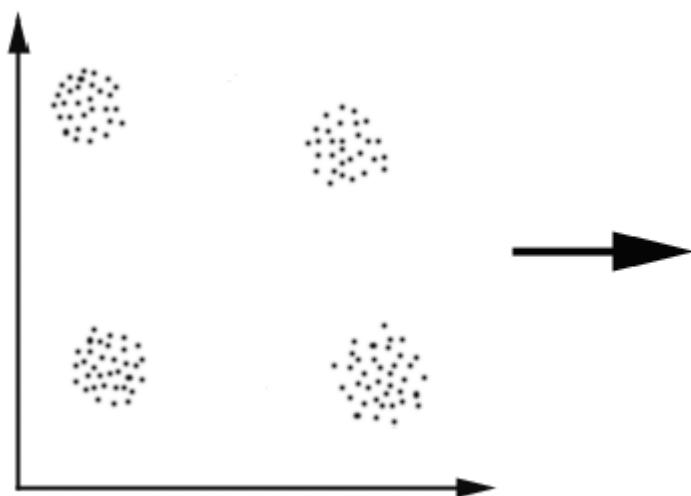
## t-Distributed Stochastic Neighbor Embedding

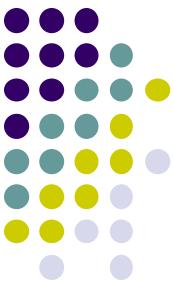


MORE ...

t-SNE visualization of CNN codes  
with 50,000 [ILSVRC 2012](#) validation images

# Clustering





# Spectral clustering

- The spectral clustering method we define relies on a random walk representation over the points. We construct this in three steps

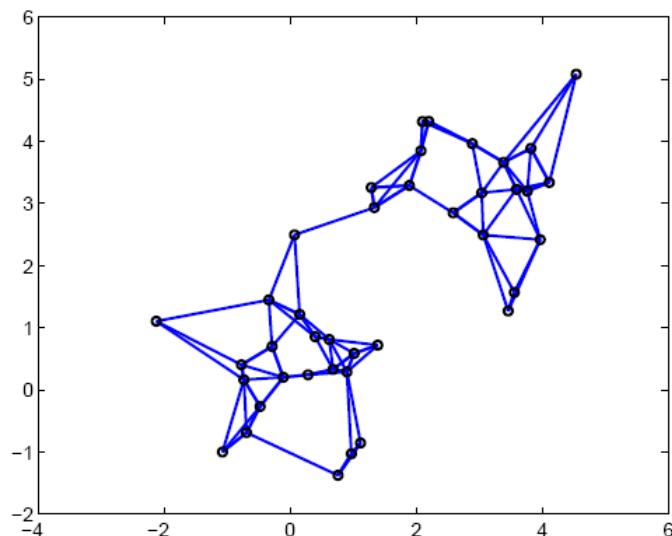
1. a nearest neighbor graph
2. similarity weights on the edges:

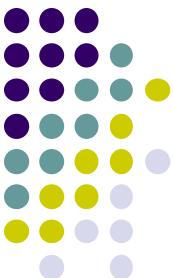
$$W_{ij} = \exp\{-\beta\|\mathbf{x}_i - \mathbf{x}_j\|\}$$

where  $W_{ii} = 1$  and the weight is zero for non-edges.

3. transition probability matrix

$$P_{ij} = W_{ij} / \sum_{j'} W_{ij'}$$





# Properties of the random walk

- If we start from  $i_0$ , the distribution of points  $i_t$  that we end up in after  $t$  steps is given by

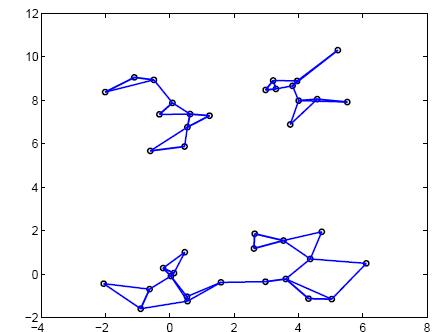
$$i_1 \sim P_{i_0 i_1}, \quad P_{ij} = \frac{W_{ij}}{W_{i \cdot}}, \text{ where } W_{i \cdot} = \sum_j W_{ij}$$

$$i_2 \sim \sum_{i_1} P_{i_0, i_1} P_{i_1, i_2} = [P^2]_{i_0 i_2},$$

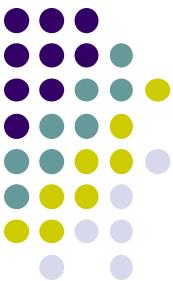
$$i_3 \sim \sum_{i_1} \sum_{i_2} P_{i_0, i_1} P_{i_1, i_2} P_{i_2, i_3} = [P^3]_{i_0 i_3},$$

...

$$i_t \sim [P^t]_{i_0 i_t}$$



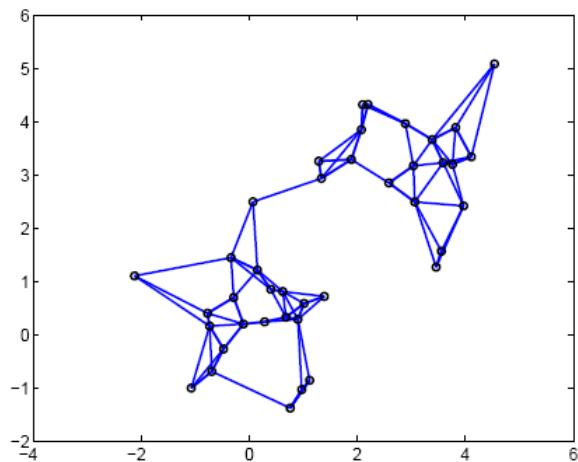
where  $P^t = PP\dots P$  ( $t$  matrix products) and  $[\cdot]_{ij}$  denotes the  $i, j$  component of the matrix.



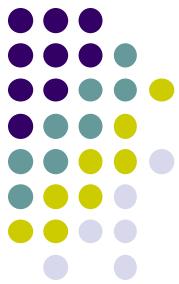
# Random walk and clustering

- The distributions of points we end up in after  $t$  steps converge as  $t$  increases. If the graph is connected, the resulting distribution is independent of the starting point

Even for large  $t$ , the transition probabilities  $[P^t]_{ij}$  have a slightly higher probability of transitioning within “clusters” than across; we want to recover this effect from eigenvalues/vectors



# Eigenvalues/vectors and spectral clustering



- Let  $W$  be the matrix with components  $W_{ij}$  and  $D$  a diagonal matrix such that  $D_{ii} = \sum_j W_{ij}$ . Then

$$P = D^{-1}W$$

- To find out how  $P^t$  behaves for large  $t$  it is useful to examine the eigen-decomposition of the following symmetric matrix

$$D^{-\frac{1}{2}}WD^{-\frac{1}{2}} = \lambda_1 \mathbf{z}_1 \mathbf{z}_1^T + \lambda_2 \mathbf{z}_2 \mathbf{z}_2^T + \dots + \lambda_n \mathbf{z}_n \mathbf{z}_n^T$$

where the ordering is such that  $|\lambda_1| \geq |\lambda_2| \geq \dots \geq |\lambda_n|$ .



# Eigenvalues/vectors cont'd

- The symmetric matrix is related to  $P^t$  since

$$(D^{-\frac{1}{2}}WD^{-\frac{1}{2}}) \cdots (D^{-\frac{1}{2}}WD^{-\frac{1}{2}}) = D^{\frac{1}{2}}(P \cdots P)D^{-\frac{1}{2}}$$

This allows us to write the  $t$  step transition probability matrix in terms of the eigenvalues/vectors of the symmetric matrix

$$\begin{aligned} P^t &= D^{-\frac{1}{2}} \left( D^{-\frac{1}{2}}WD^{-\frac{1}{2}} \right)^t D^{\frac{1}{2}} \\ &= D^{-\frac{1}{2}} \left( \lambda_1^t \mathbf{z}_1 \mathbf{z}_1^T + \lambda_2^t \mathbf{z}_2 \mathbf{z}_2^T + \dots + \lambda_n^t \mathbf{z}_n \mathbf{z}_n^T \right) D^{\frac{1}{2}} \end{aligned}$$

where  $\lambda_1 = 1$  and

$$P^\infty = D^{-\frac{1}{2}} \left( \mathbf{z}_1 \mathbf{z}_1^T \right) D^{\frac{1}{2}}$$

# Eigenvalues/vectors and spectral clustering



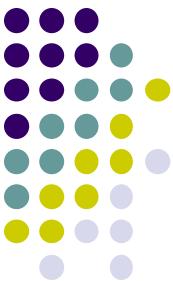
- We are interested in the largest correction to the asymptotic limit

$$P^t \approx P^\infty + D^{-\frac{1}{2}} \left( \lambda_2^t \mathbf{z}_2 \mathbf{z}_2^T \right) D^{\frac{1}{2}}$$

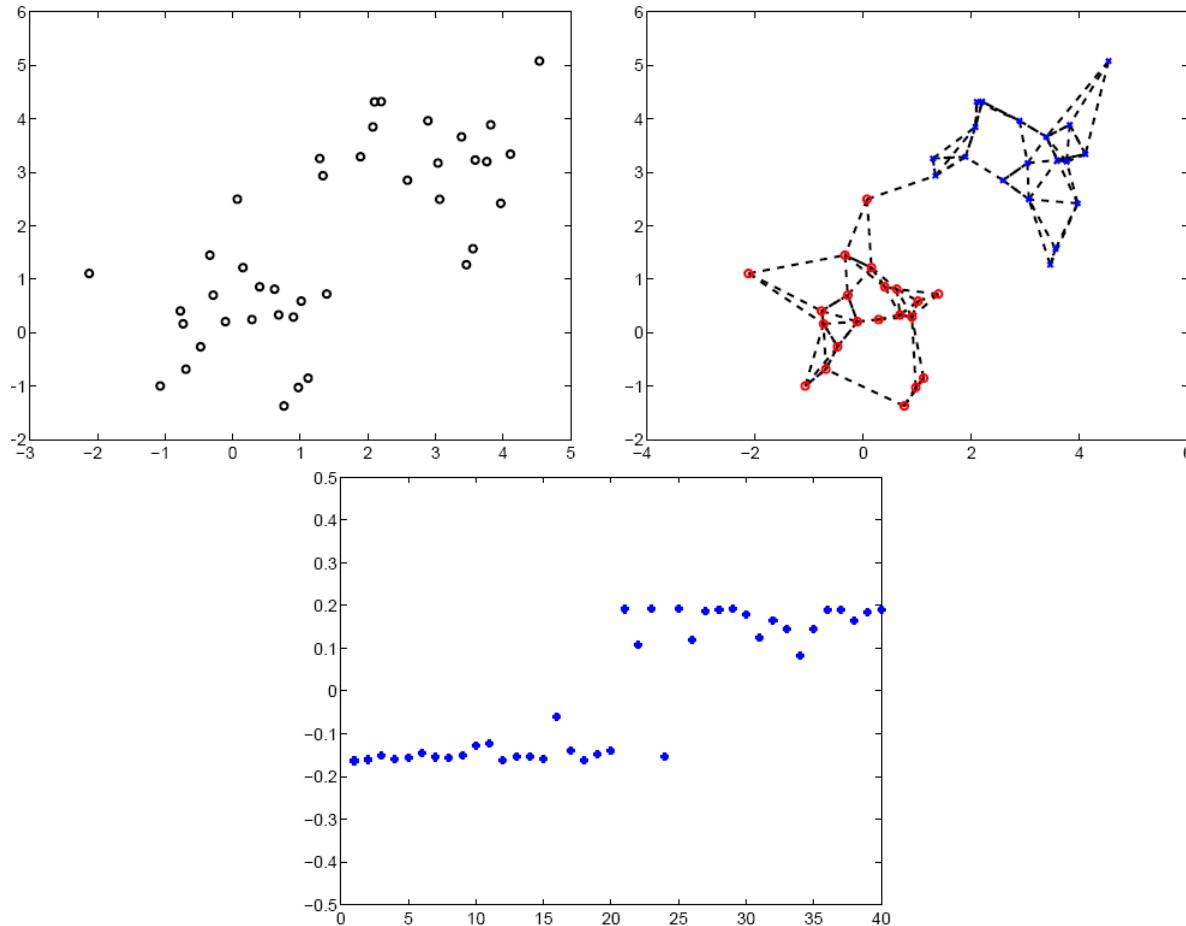
Note:  $[\mathbf{z}_2 \mathbf{z}_2^T]_{ij} = z_{2i} z_{2j}$  and thus the largest correction term increases the probability of transitions between points that share the same sign of  $z_{2i}$  and decreases transitions across points with different signs

- Binary spectral clustering: we divide the points into clusters based on the sign of the elements of  $\mathbf{z}_2$

$$z_{2j} > 0 \Rightarrow \text{cluster 1, otherwise cluster 0}$$



# Spectral clustering: example



Components of the eigenvector corresponding to the second largest eigenvalue



# Reference papers of SC

- A. Y. Ng, M. I. Jordan, and Y. Weiss, *On spectral clustering: Analysis and an algorithm*, NIPS, (2001)
- Y. Weiss, *Segmentation using eigenvectors: a unifying view*. ICCV, (1999)
- J. Shi and J. Malik, *Normalized cuts and image segmentation*, IEEE TPAMI, 22 (2000)
- And more about image segmentations ...
  - Graph cut
  - Mean-shift



# Classical methods on cluster distance



- A *linkage* method: we have to be able to measure distances between clusters of examples  $C_k$  and  $C_l$

a) Single linkage:

Nearest neighbor

$$d_{kl} = \min_{i \in C_k, j \in C_l} d(\mathbf{x}_i, \mathbf{x}_j)$$

b) Average linkage:

$$d_{kl} = \frac{1}{|C_l| |C_k|} \sum_{i \in C_k, j \in C_l} d(\mathbf{x}_i, \mathbf{x}_j)$$

c) Centroid linkage:

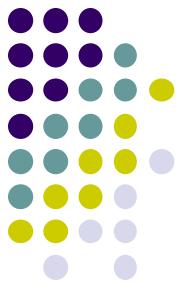
$$d_{kl} = d(\bar{\mathbf{x}}_k, \bar{\mathbf{x}}_l), \quad \bar{\mathbf{x}}_l = \frac{1}{|C_l|} \sum_{i \in C_l} \mathbf{x}_i$$

# Hierarchical (bottom-up) clustering

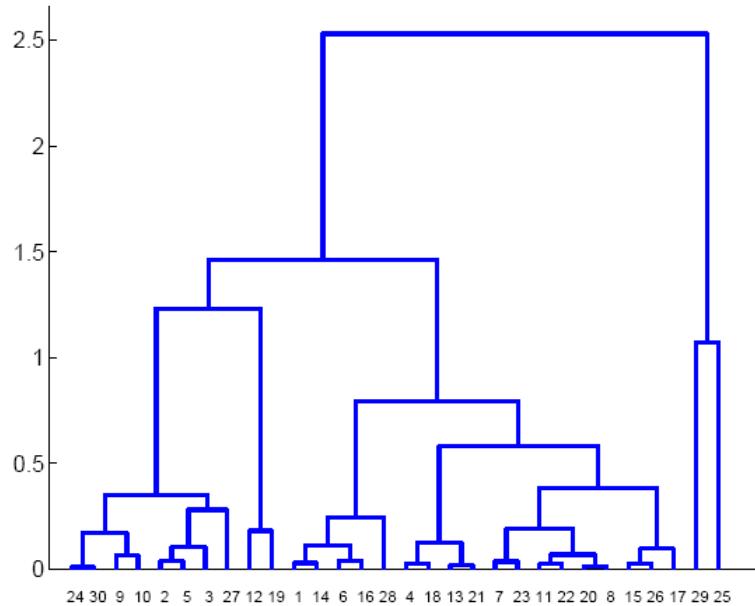


- Hierarchical agglomerative clustering: we sequentially merge the pair of “closest” points/clusters
- The procedure
  1. Find two closest points (clusters) and merge them
  2. Proceed until we have a single cluster (all the points)
- Two prerequisites:
  1. distance measure  $d(x_i, x_j)$  between two points
  2. distance measure between clusters (cluster linkage)

# Hierarchical (bottom-up) clustering

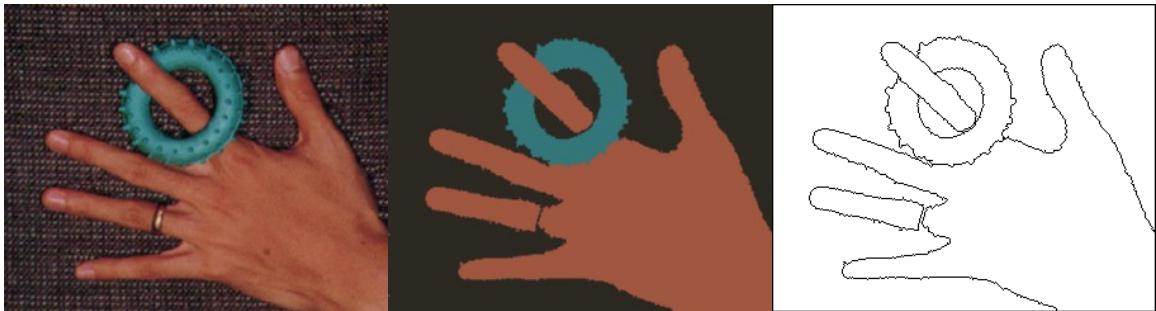
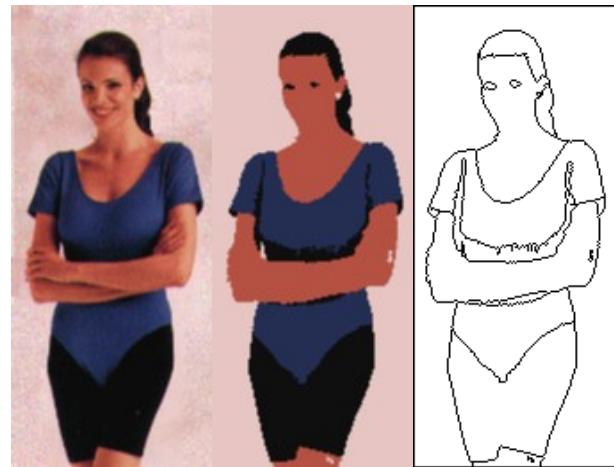


- A dendrogram representation of hierarchical clustering



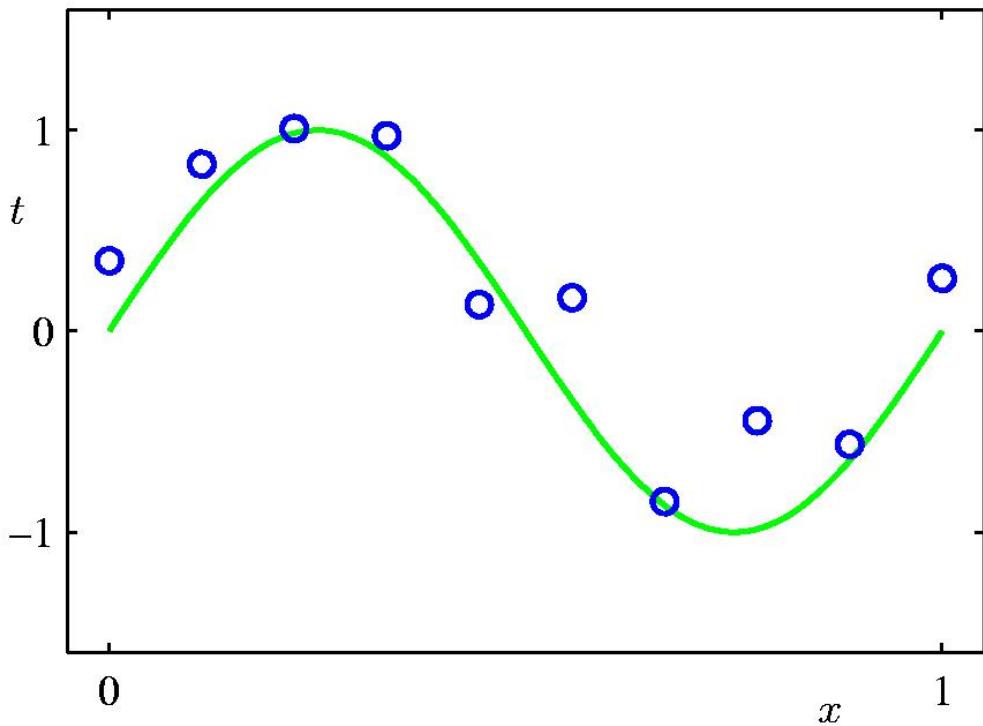
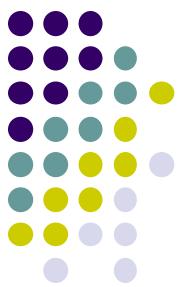
The height of each pair represents the distance between the merged clusters; the specific linear ordering of points is chosen for clarity

# Clustering and image segmentation



Mean-shift segmentation

# Regression revisit: Polynomial Curve Fitting



$$\mathbf{h}(x) = \begin{pmatrix} h_0(x) \\ h_1(x) \\ \vdots \\ h_M(x) \end{pmatrix} \quad \mathbf{w} = \begin{pmatrix} w_0 \\ w_1 \\ \vdots \\ w_M \end{pmatrix} \quad \mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}$$

$$y = \mathbf{h}(x) \cdot \mathbf{w}$$

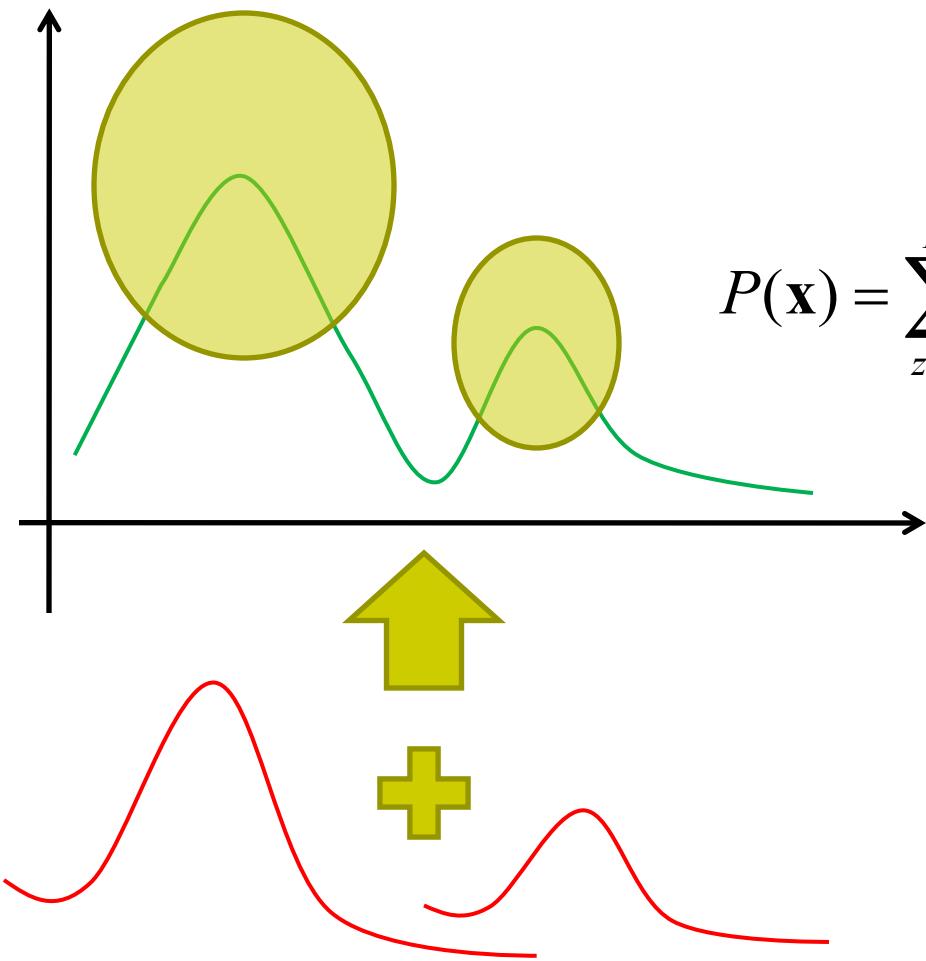
$$\mathbf{w} = (\mathbf{H}^T \mathbf{H})^{-1} \mathbf{H}^T \mathbf{y}$$

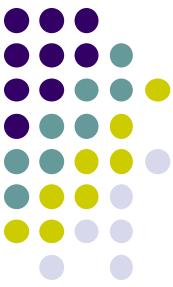
Normal equation

  $y(x, \mathbf{w}) = w_0 + w_1 x + w_2 x^2 + \dots + w_M x^M = \sum_{j=0}^M w_j x^j$

$y(x, \mathbf{w}) = w_0 + w_1 h_1(x) + w_2 h_2(x) + \dots + w_M h_M(x) = \sum_{j=0}^M w_j h_j(x)$  **Basis function**

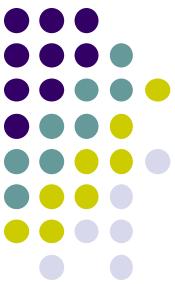
# Regression revisit: alternative approach





# Mixtures of Gaussians

- Mixture distribution:
  - Assume  $P(x)$  is a mixture of  $K$  different Gaussians
  - Assume each data point,  $x$  is generated by 2-step process
    - Choose one of the  $K$  Gaussians as label  $z$
    - Generate  $x$  according to the Gaussian  $N(\mu_z, \Sigma_z)$
- What object function shall we optimize?
  - Maximize data likelihood

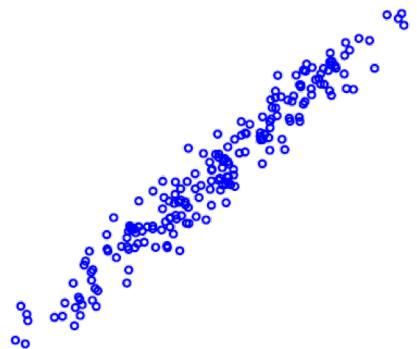


# Mixtures of Gaussians (cont.)

- Multivariate Gaussian model

$$p(\mathbf{x}|\mu, \Sigma) = \frac{1}{(2\pi)^{p/2}|\Sigma|^{1/2}} \exp\left\{-\frac{1}{2}(\mathbf{x} - \mu)^T \Sigma^{-1} (\mathbf{x} - \mu)\right\}$$

- How to generate it?
  - **numpy.random.multivariate\_normal**





# Mixtures of Gaussians (cont.)

## `numpy.random.multivariate_normal`

`numpy.random.multivariate_normal(mean, cov[, size, check_valid, tol])`

Draw random samples from a multivariate normal distribution.

The multivariate normal, multinormal or Gaussian distribution is a generalization of the one-dimensional normal distribution to higher dimensions. Such a distribution is specified by its mean and covariance matrix. These parameters are analogous to the mean (average or “center”) and variance (standard deviation, or “width,” squared) of the one-dimensional normal distribution.

**Parameters:** `mean : 1-D array_like, of length N`

Mean of the  $N$ -dimensional distribution.

`cov : 2-D array_like, of shape (N, N)`

Covariance matrix of the distribution. It must be symmetric and positive-semidefinite for proper sampling.

`size : int or tuple of ints, optional`

Given a shape of, for example, `(m, n, k)`,  $m * n * k$  samples are generated, and packed in an  $m$ -by- $n$ -by- $k$  arrangement. Because each sample is  $N$ -dimensional, the output shape is `(m, n, k, N)`. If no shape is specified, a single ( $N$ -D) sample is returned.

`check_valid : {‘warn’, ‘raise’, ‘ignore’}, optional`

Behavior when the covariance matrix is not positive semidefinite.

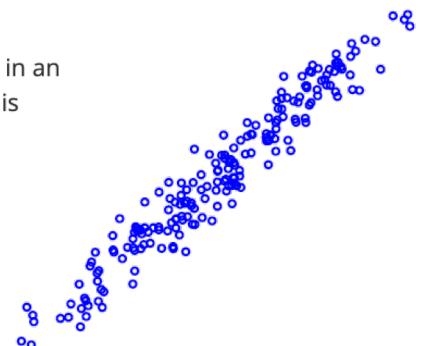
`tol : float, optional`

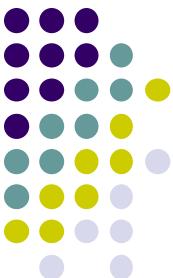
Tolerance when checking the singular values in covariance matrix.

**Returns:** `out : ndarray`

The drawn samples, of shape `size`, if that was provided. If not, the shape is `(N, )`.

In other words, each entry `out[i, j, ..., :]` is an  $N$ -dimensional value drawn from the distribution.





# Mixtures of Gaussians (cont.)

- Multivariate Gaussian model

$$p(\mathbf{x}|\mu, \Sigma) = \frac{1}{(2\pi)^{p/2} |\Sigma|^{1/2}} \exp\left\{-\frac{1}{2}(\mathbf{x} - \mu)^T \Sigma^{-1} (\mathbf{x} - \mu)\right\}$$

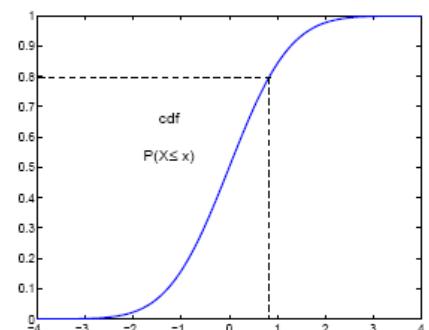
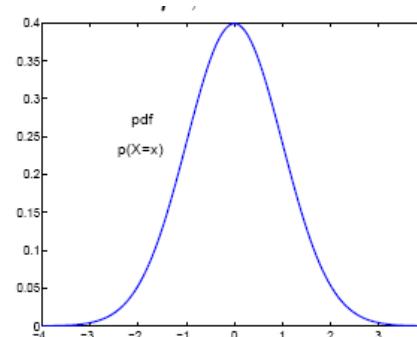
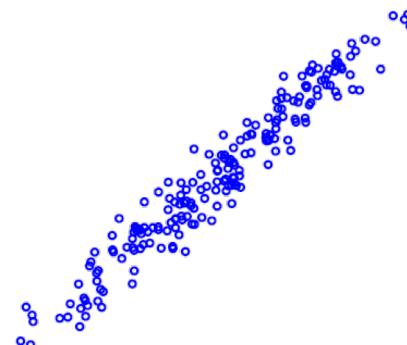
- How to generate it?

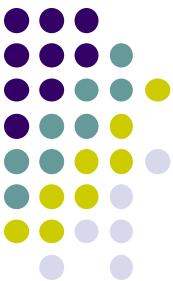
$$F_{\mu, \sigma^2}(x) = \int_{-\infty}^x p(z|\mu, \sigma^2) dz$$

$$u \sim \text{Uniform}(0, 1) \Rightarrow x = F_{\mu, \sigma^2}^{-1}(u) \sim p(x|\mu, \sigma^2)$$

$$z_i \sim p(z_i|\mu = 0, \sigma^2 = 1), \quad \mathbf{z} = [z_1, \dots, z_d]^T$$

$$\mathbf{x} = \Sigma^{1/2} \mathbf{z} + \mu$$



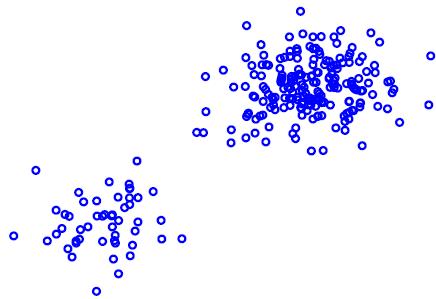


# Multi-variate density estimation

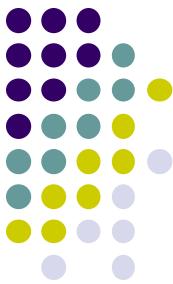
- A mixture of Gaussians model

$$p(\mathbf{x}|\theta) = \sum_{i=1}^k p_j p(\mathbf{x}|\mu_j, \Sigma_j)$$

where  $\theta = \{p_1, \dots, p_k, \mu_1, \dots, \mu_k, \Sigma_1, \dots, \Sigma_k\}$  contains all the parameters of the mixture model.  $\{p_j\}$  are known as *mixing proportions or coefficients*.



# Mixtures of Gaussians: Wishart distribution



- A mixture of Gaussian Model:

$$p(\mathbf{x}|\theta) = \sum_{i=1}^k p_j p(\mathbf{x}|\mu_j, \Sigma_j)$$

High dimensional  
parameters

$$\theta = \{p_1, \dots, p_k, \mu_1, \dots, \mu_k, \underline{\Sigma_1, \dots, \Sigma_k}\}$$

- Wishart prior

$$P(\Sigma|S, n') \propto \frac{1}{|\Sigma|^{n'/2}} \exp\left(-\frac{n'}{2} \text{Trace}(\Sigma^{-1} S)\right)$$

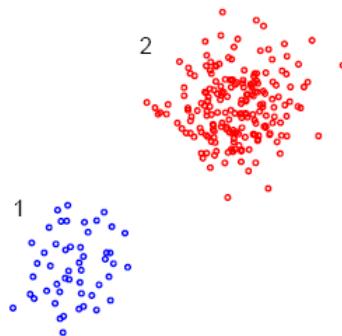
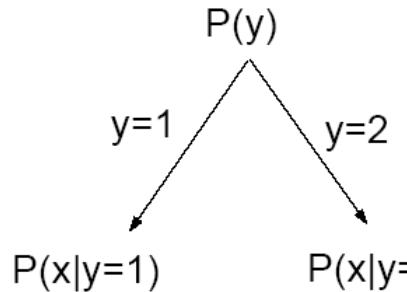
$S$  = “prior” covariance matrix

$n'$  = equivalent sample size



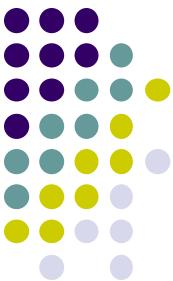
# Mixture density

- Data generation process:



$$\begin{aligned} p(\mathbf{x}|\theta) &= \sum_{j=1,2} P(y = j) \cdot p(\mathbf{x}|y = j) \quad (\text{generic mixture}) \\ &= \sum_{j=1,2} p_j \cdot p(\mathbf{x}|\mu_j, \Sigma_j) \quad (\text{mixture of Gaussians}) \end{aligned}$$

- Any data point  $\mathbf{x}$  could have been generated in two ways



# Mixture density

- If we are given just  $\mathbf{x}$  we don't know which mixture component this example came from

$$p(\mathbf{x}|\theta) = \sum_{j=1,2} p_j p(\mathbf{x}|\mu_j, \Sigma_j)$$

- We can evaluate the posterior probability that an observed  $\mathbf{x}$  was generated from the first mixture component

$$\begin{aligned} P(y = 1|\mathbf{x}, \theta) &= \frac{P(y = 1) \cdot p(\mathbf{x}|y = 1)}{\sum_{j=1,2} P(y = j) \cdot p(\mathbf{x}|y = j)} \\ &= \frac{p_1 p(\mathbf{x}|\mu_1, \Sigma_1)}{\sum_{j=1,2} p_j p(\mathbf{x}|\mu_j, \Sigma_j)} \end{aligned}$$

- This solves a *credit assignment* problem

# Mixture density: posterior sampling



- Consider sampling  $\mathbf{x}$  from the mixture density, then  $y$  from the posterior over the components given  $\mathbf{x}$ , and finally  $\mathbf{x}'$  from the component density indicated by  $y$ :

$$\mathbf{x} \sim p(\mathbf{x}|\theta)$$

$$y \sim P(y|\mathbf{x}, \theta)$$

$$\mathbf{x}' \sim p(\mathbf{x}'|y, \theta)$$

Is  $y$  a fair sample from the prior distribution  $P(y)$ ?

Is  $\mathbf{x}'$  a fair sample from the mixture density  $p(\mathbf{x}'|\theta)$ ?

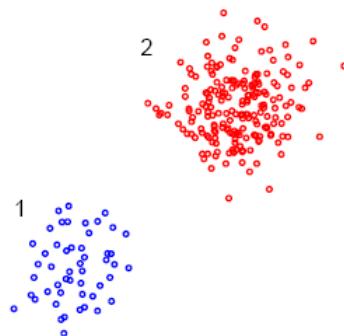


# Mixture density estimation

- Suppose we want to estimate a two component mixture of Gaussians model.

$$p(\mathbf{x}|\theta) = p_1 p(\mathbf{x}|\mu_1, \Sigma_1) + p_2 p(\mathbf{x}|\mu_2, \Sigma_2)$$

- If each example  $\mathbf{x}_i$  in the training set were labeled  $y_i = 1, 2$  according to which mixture component (1 or 2) had generated it, then the estimation would be easy.

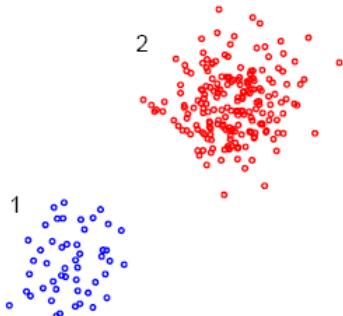


- Labeled examples  $\Rightarrow$  no credit assignment problem



# Mixture density estimation

When examples are already assigned to mixture components (labeled), we can estimate each Gaussian independently



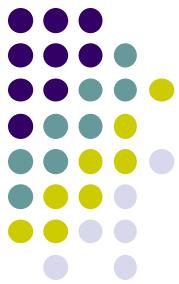
- If  $\hat{n}_j$  is the number of examples labeled  $j$ , then for each  $j = 1, 2$  we set

$$\hat{p}_j \leftarrow \frac{\hat{n}_j}{n}$$

$$\hat{\mu}_j \leftarrow \frac{1}{\hat{n}_j} \sum_{i:y_i=j} \mathbf{x}_i$$

$$\hat{\Sigma}_j \leftarrow \frac{1}{\hat{n}_j} \sum_{i:y_i=j} (\mathbf{x}_i - \hat{\mu}_j)(\mathbf{x}_i - \hat{\mu}_j)^T$$

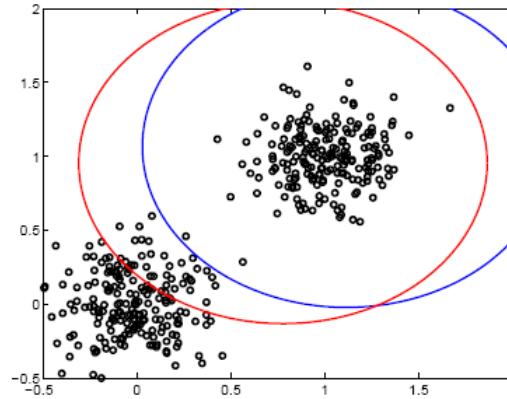
# Mixture density estimation: credit assignment



- Of course we don't have such labels ... but we can guess what the labels might be based on our current mixture distribution
- We get soft labels or posterior probabilities of which Gaussian generated which example:

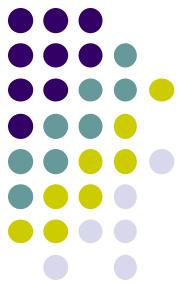
$$\hat{p}(j|i) \leftarrow P(y_i = j | \mathbf{x}_i, \theta)$$

where  $\sum_{j=1,2} \hat{p}(j|i) = 1$  for all  $i = 1, \dots, n$ .



- When the Gaussians are almost identical (as in the figure),  $\hat{p}(1|i) \approx \hat{p}(2|i)$  for almost any available point  $\mathbf{x}_i$ .  
Even slight differences can help us determine how we should modify the Gaussians.

# The Expectation-Maximization algorithm



**E-step:** softly assign examples to mixture components

$$\hat{p}(j|i) \leftarrow P(y_i = j | \mathbf{x}_i, \theta), \text{ for all } j = 1, 2 \text{ and } i = 1, \dots, n$$

**M-step:** re-estimate the parameters (separately for the two Gaussians) based on the soft assignments.

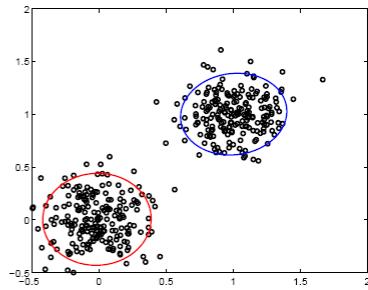
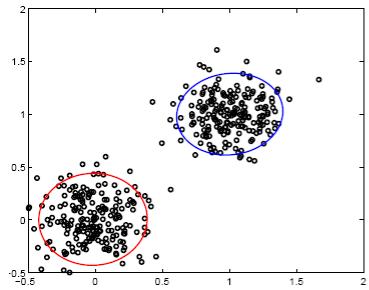
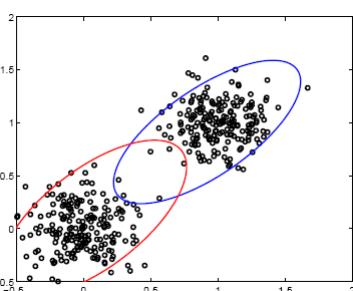
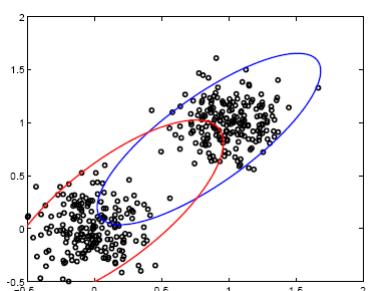
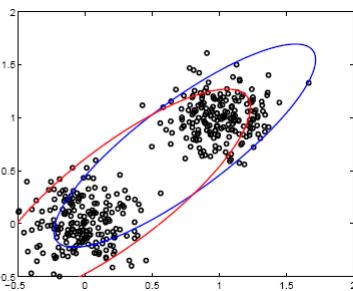
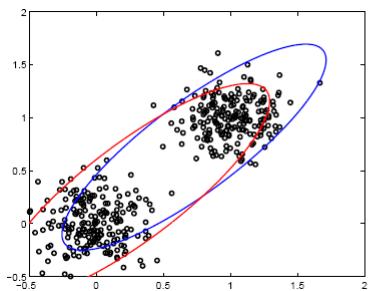
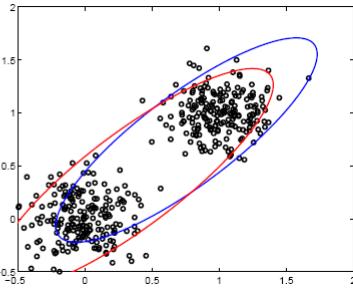
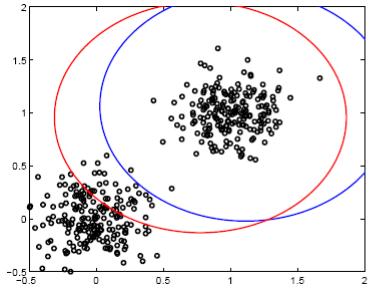
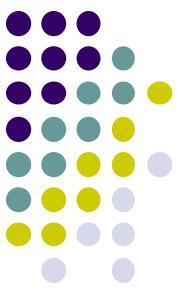
$$\hat{n}_j \leftarrow \sum_{i=1}^n \hat{p}(j|i) = \text{Soft \# of examples labeled } j$$

$$\hat{p}_j \leftarrow \frac{\hat{n}_j}{n}$$

$$\hat{\mu}_j \leftarrow \frac{1}{\hat{n}_j} \sum_{i=1}^n \hat{p}(j|i) \mathbf{x}_i$$

$$\hat{\Sigma}_j \leftarrow \frac{1}{\hat{n}_j} \sum_{i=1}^n \hat{p}(j|i) (\mathbf{x}_i - \hat{\mu}_j)(\mathbf{x}_i - \hat{\mu}_j)^T$$

# Mixture density estimation: example



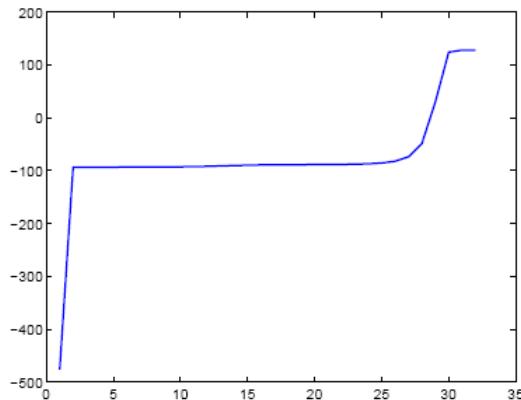


# The EM-algorithm

- Each iteration of the EM-algorithm *monotonically* increases the (log-)likelihood of the  $n$  training examples  $\mathbf{x}_1, \dots, \mathbf{x}_n$ :

$$\log p(\text{data} | \theta) = \sum_{i=1}^n \log \left( \overbrace{p_1 p(\mathbf{x}_i | \mu_1, \Sigma_1) + p_2 p(\mathbf{x}_i | \mu_2, \Sigma_2)}^{p(\mathbf{x}_i | \theta)} \right)$$

where  $\theta = \{p_1, p_2, \mu_1, \mu_2, \Sigma_1, \Sigma_2\}$  contains all the parameters of the mixture model.





# The EM algorithm

- The EM-algorithm finds a local maximum of  $l(\theta; D)$

**E-step:** evaluate the expected complete log-likelihood

$$\begin{aligned} J(\theta; \theta^{(t)}) &= \sum_{i=1}^n E_{j \sim P(j|\mathbf{x}_i, \theta^{(t)})} \log \left( p_j p(\mathbf{x}_i | \mu_j, \Sigma_j) \right) \\ &= \sum_{i=1}^n \sum_{j=1,2} P(j|\mathbf{x}_i, \theta^{(t)}) \log \left( p_j p(\mathbf{x}_i | \mu_j, \Sigma_j) \right) \end{aligned}$$

**M-step:** find the new parameters by maximizing the expected complete log-likelihood

$$\theta^{(t+1)} \leftarrow \arg \max_{\theta} J(\theta; \theta^{(t)})$$



# Regularized EM algorithm

- To maximize a penalized (regularized) log-likelihood

$$l'(\theta; D) = \sum_{i=1}^n \log p(\mathbf{x}_i | \theta) + \log p(\theta)$$

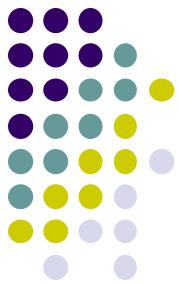
we only need to modify the M-step of the EM-algorithm.

Specifically, in the M-step, we find  $\theta$  that maximize a penalized expected complete log-likelihood:

$$\begin{aligned} J(\theta; \theta^{(t)}) &= \sum_{i=1}^n E_{j \sim P(j | \mathbf{x}_i, \theta^{(t)})} \log \left( p_j p(\mathbf{x}_i | \mu_j, \Sigma_j) \right) \\ &\quad + \log p(p_1, p_2) + \log p(\Sigma_1) + \log p(\Sigma_1) \end{aligned}$$

where, for example,  $p(p_1, p_2)$  could be a *Dirichlet* and each  $p(\Sigma_j)$  a *Wishart* prior.

# Selecting the number of components

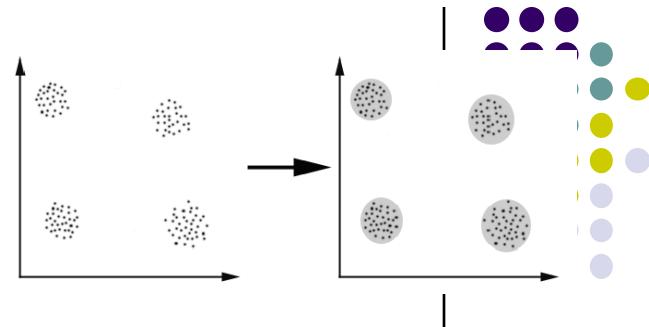


- As a simple strategy for selecting the appropriate number of mixture components, we can find  $k$  that minimize the following asymptotic approximation to the description length:

$$DL \approx -\log p(\text{data}|\hat{\theta}_k) + \frac{d_k}{2} \log(n)$$

where  $n$  is the number of training points,  $\hat{\theta}_k$  is the maximum likelihood parameter estimate for the  $k$ -component mixture, and  $d_k$  is the (effective) number of parameters in the  $k$ -mixture.

# K-means clustering



Given data  $\langle x_1 \dots x_n \rangle$ , and  $K$ , assign each  $x_i$  to one of  $K$  clusters,

$$C_1 \dots C_K, \text{ minimizing } J = \sum_{j=1}^K \sum_{x_i \in C_j} \|x_i - \mu_j\|^2$$

Where  $\mu_j$  is mean over all points in cluster  $C_j$

## K-Means Algorithm:

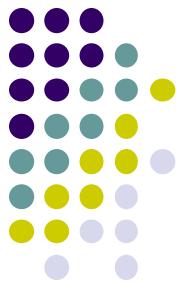
Initialize  $\mu_1 \dots \mu_K$  randomly

Repeat until convergence:

1. Assign each point  $x_i$  to the cluster with the closest mean  $\mu_j$
2. Calculate the new mean for each cluster

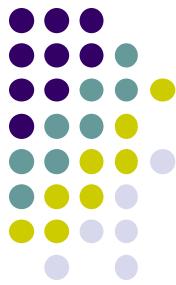
$$\mu_j \leftarrow \frac{1}{|C_j|} \sum_{x_i \in C_j} x_i$$

# K-Means vs. Mixture of Gaussians

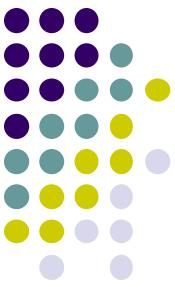


- Both are iterative algorithms to assign points to clusters
- Objective function
  - K Means: minimize  $J = \sum_{j=1}^K \sum_{x_i \in C_j} \|x_i - \mu_j\|^2$
  - MoG: maximize likelihood  $P(X|\theta)$
- MoG the more general formulation
  - Equivalent to K Means when  $\Sigma_k = \sigma I$ , and  $\sigma \rightarrow 0$

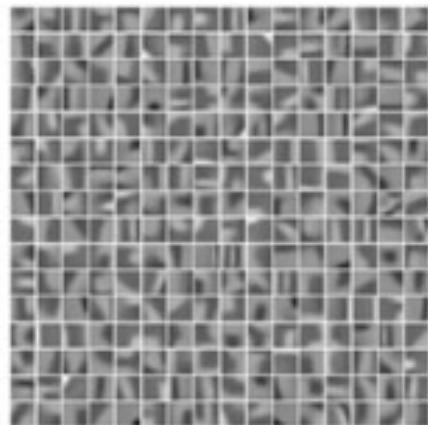
# Disadvantage of K-means and MOG



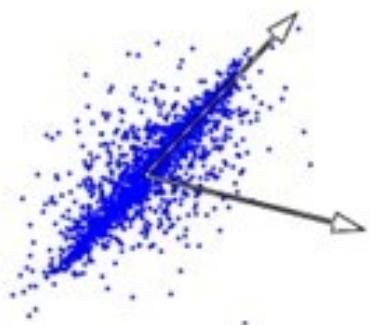
- The result is sensitive to the initial data
- How to determine the number of clusters



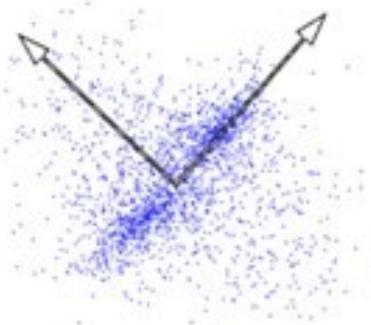
# K-means and whitening



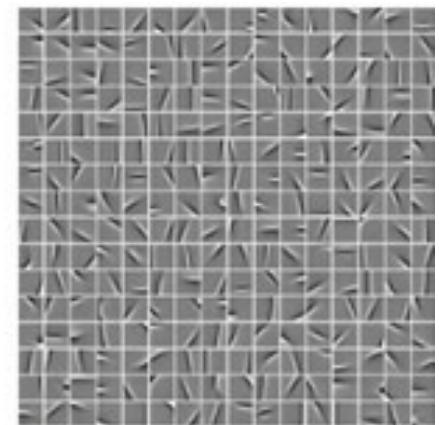
(a)



(b)



<http://blog.csdn.net/zouxy09>



(c)

Learning Feature Representations with K-means, Adam Coates and Andrew Y. Ng. In Neural Networks: Tricks of the Trade, Reloaded, Springer LNCS, 2012



# K-means and whitening

1. Normalize inputs:

$$x^{(i)} := \frac{x^{(i)} - \text{mean}(x^{(i)})}{\sqrt{\text{var}(x^{(i)}) + \epsilon_{\text{norm}}}}, \forall i$$

2. Whiten inputs:

$$[V, D] := \text{eig}(\text{cov}(x)); // \text{ So } VDV^\top = \text{cov}(x)$$

$$x^{(i)} := V(D + \epsilon_{\text{zca}}I)^{-1/2}V^\top x^{(i)}, \forall i$$

3. Loop until convergence (typically 10 iterations is enough):

$$s_j^{(i)} := \begin{cases} \mathcal{D}^{(j)\top} x^{(i)} & \text{if } j == \arg \max_l |\mathcal{D}^{(l)\top} x^{(i)}| \\ 0 & \text{otherwise.} \end{cases} \quad \forall j, i$$

$$\mathcal{D} := XS^\top + \mathcal{D}$$

$$\mathcal{D}^{(j)} := \mathcal{D}^{(j)}/\|\mathcal{D}^{(j)}\|_2 \quad \forall j$$

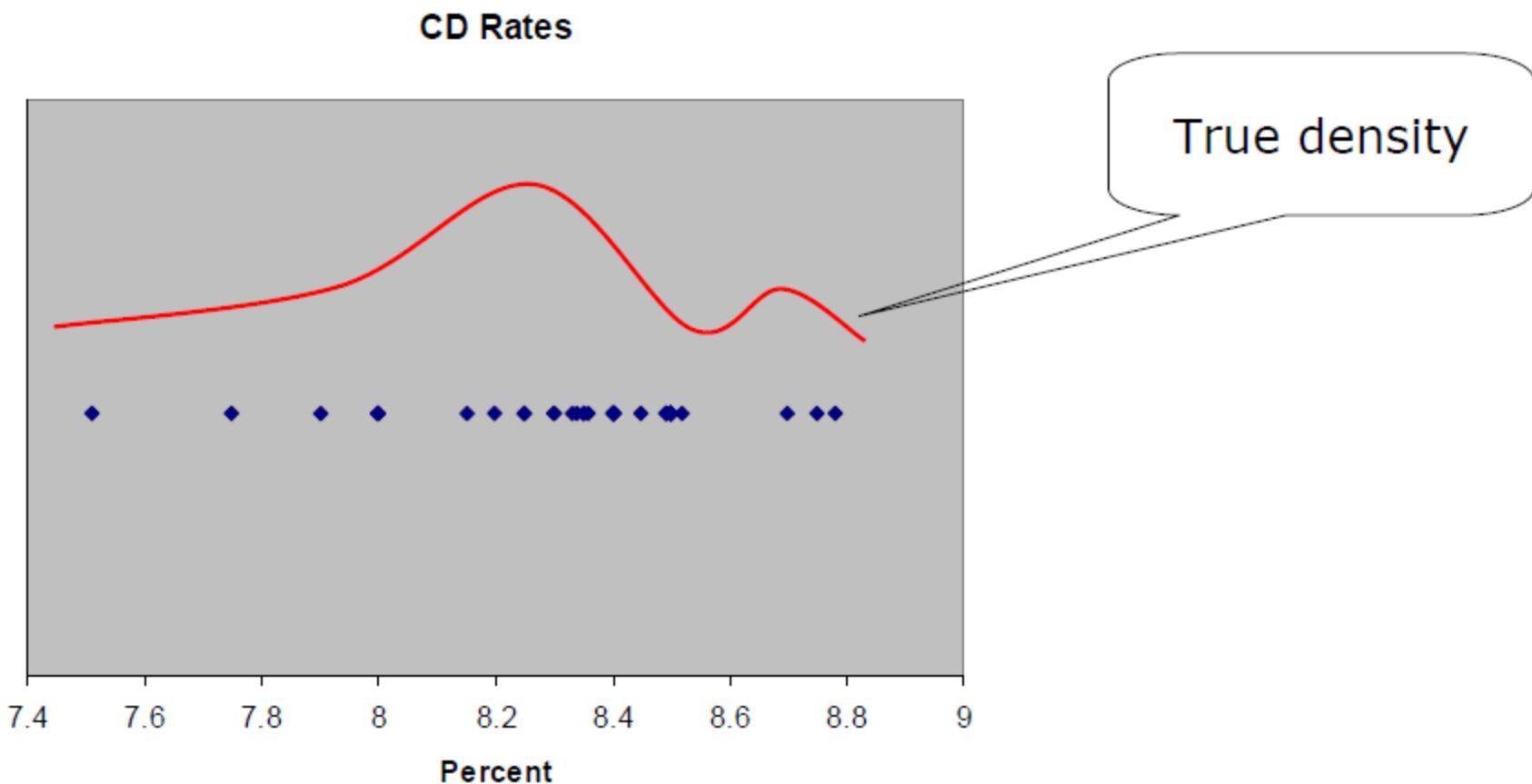


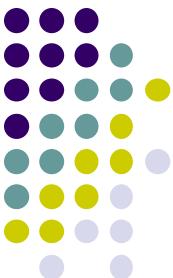
# Mean shift

- First proposed by Fukunaga in 1970's
- Widely used since 1998
  - In computer vision
  - And other areas
- The following several slides is mainly from:
  - <http://www.cs.cornell.edu/courses/cs664/2005fa/Lectures/lecture3.pdf>

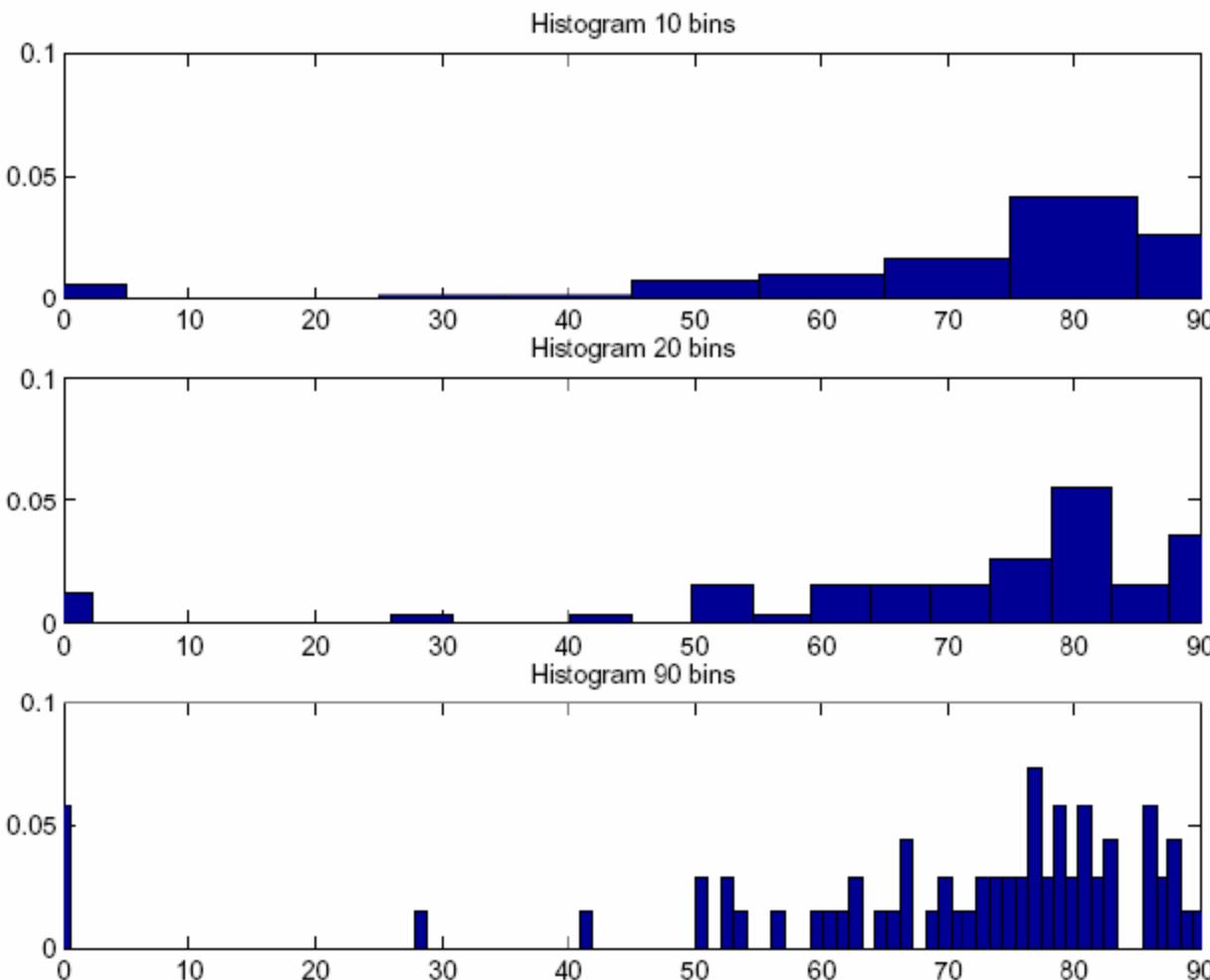


# Density estimation





# Histogram representation





# Histogram-based estimates

- You can use a variety of fitting techniques to produce a curve from a histogram
  - Lines, polynomials, splines, etc.
  - Also called regression/function approximation
  - Normalize to make this a density
- If you know quite a bit about the underlying density you can compute a good bin size
  - But that's rarely realistic in vision
  - And defeats the whole purpose of the non-parametric approach



# Nearest-neighbor estimate

- To estimate the density, count the number of nearby data points
  - Like histogramming with sliding bins
  - Avoid bin-placement artifacts

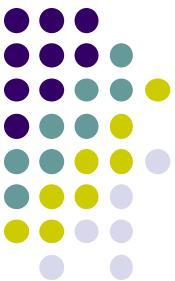
$$\hat{p}(x) = \frac{\#\{x_i \mid \|x_i - x\| \leq \varepsilon\}}{N}$$

- We can fix  $\varepsilon$  and compute this quantity, or we can fix the quantity and compute  $\varepsilon$

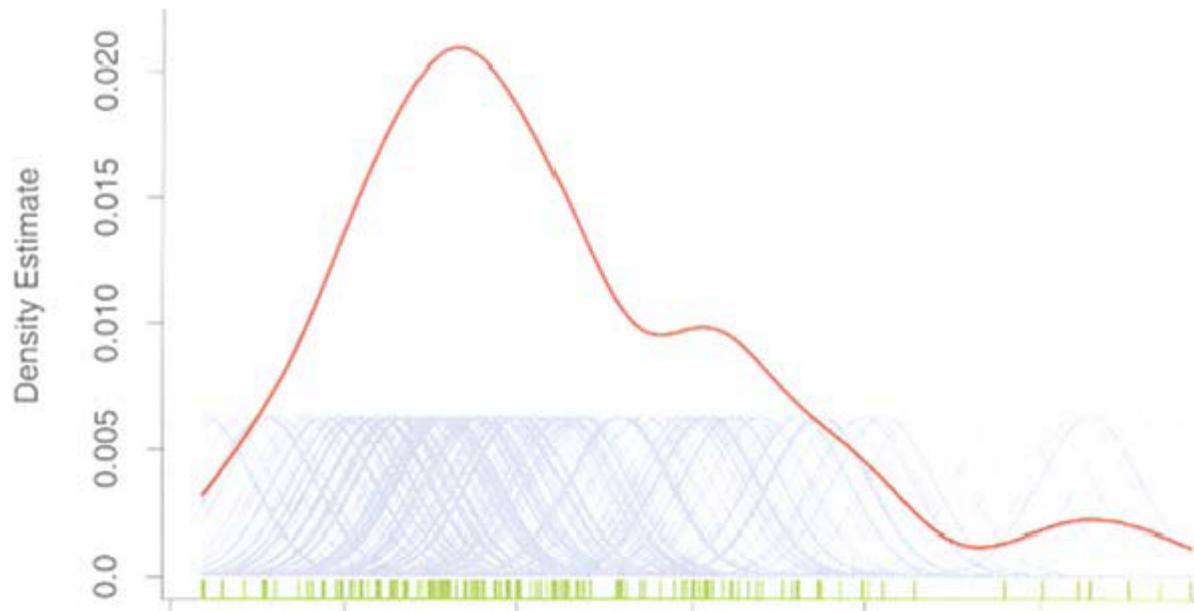


# Parzen estimation

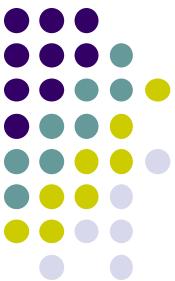
- Each observed data increases our estimate of the probability nearby
  - Simplest case: raise the probability uniformly within a fixed radius
    - Place a fixed-height “box” at each data point, add them up to get the density estimate
  - This is nearest neighbor with fixed  $\epsilon$
- More generally, you can use some slowly decreasing function (such as a Gaussian)
  - Called Kernel function



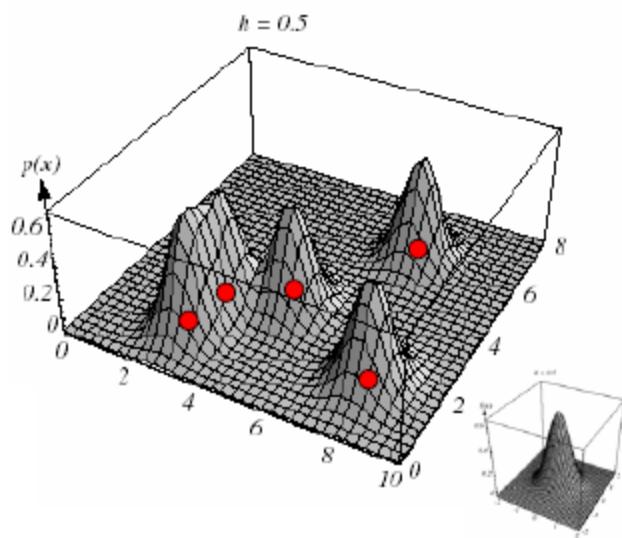
# Parzen example



from Hastie *et al.*



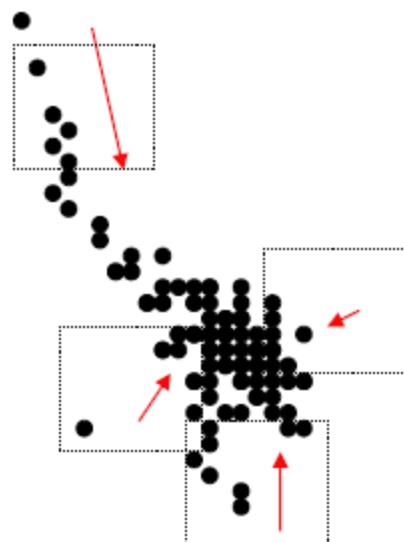
# Importance of scale





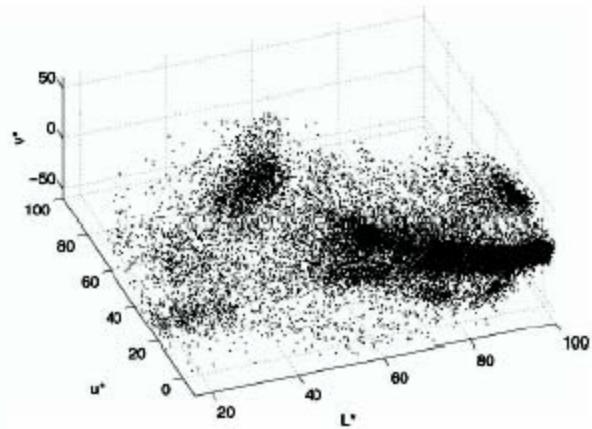
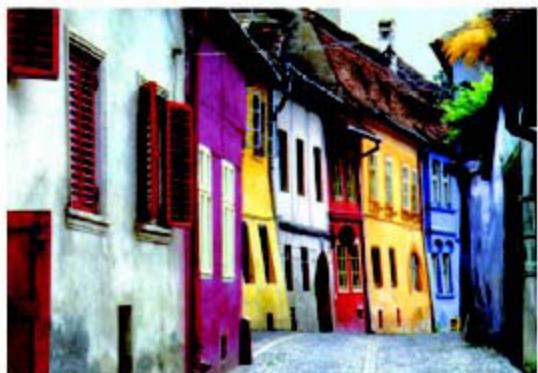
# Mean shift algorithm

- Non-parametric method to compute the nearest mode of a distribution
  - Density increases as we get near “center”



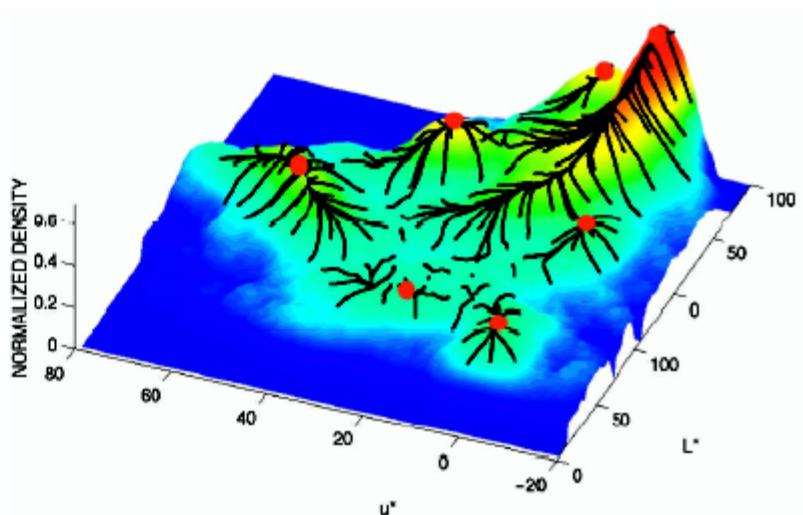
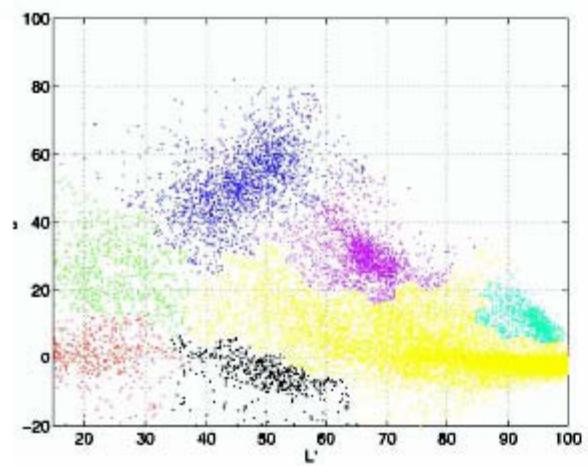
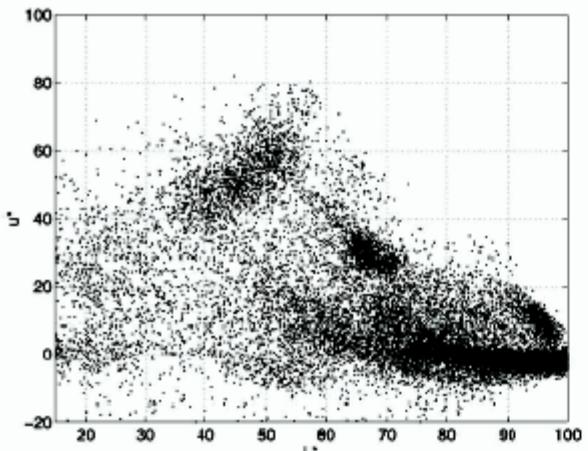


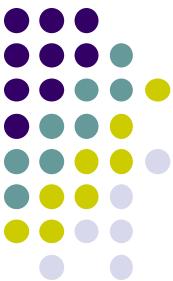
# Image and histogram





# Local modes





# Kernel Density Estimation

- Multivariate kernel density estimation

$$f(x) = \frac{1}{nh^d} \sum_{i=1}^n \frac{1}{h} K\left(\frac{x - x_i}{h}\right)$$

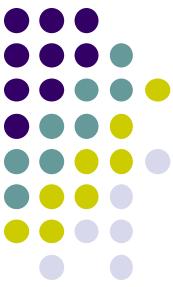
- Kernels

- Gaussian

$$K_N = (2\pi)^{-d/2} \exp\left(-\frac{1}{2} \|\mathbf{x}\|^2\right)$$

- Epanechnikov

$$K_E = \begin{cases} 1/2c_d^{-1}(d+2)(1-\|\mathbf{x}\|^2) & \text{if } \|\mathbf{x}\| < 1 \\ 0 & \text{otherwise} \end{cases}$$

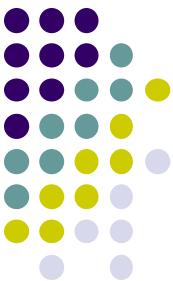


# Finding Mean-Shift Vector

- Gradient computation
  - For symmetric kernel

$$\hat{\nabla}f(\mathbf{x}) = \frac{2}{nh^{d+2}} \sum_{i=1}^n K_N\left(\frac{\mathbf{x} - \mathbf{x}_i}{h}\right) \left[ \frac{\sum_{i=1}^n \mathbf{x}_i K_N\left(\frac{\mathbf{x} - \mathbf{x}_i}{h}\right)}{\sum_{i=1}^n K_N\left(\frac{\mathbf{x} - \mathbf{x}_i}{h}\right)} - \mathbf{x} \right]$$

- Always converges to the local maximum!

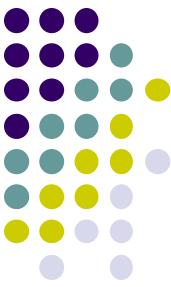


# The mean shift procedure

- Give a point  $\mathbf{x}$ 
  1. Compute the mean shift vector

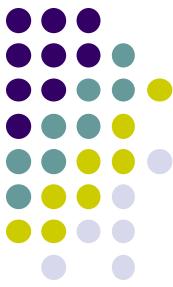
$$\hat{\nabla}f(\mathbf{x}) = \frac{2}{nh^{d+2}} \sum_{i=1}^n K_N\left(\frac{\mathbf{x} - \mathbf{x}_i}{h}\right) \left[ \frac{\sum_{i=1}^n \mathbf{x}_i K_N\left(\frac{\mathbf{x} - \mathbf{x}_i}{h}\right)}{\sum_{i=1}^n K_N\left(\frac{\mathbf{x} - \mathbf{x}_i}{h}\right)} - \mathbf{x} \right]$$

2. Translate density estimation window:
$$\mathbf{x}^{(t+1)} \leftarrow \mathbf{x}^{(t)} + \hat{\nabla}f(\mathbf{x}^{(t)})$$
3. Iterate steps 1. and 2. until convergence  
i.e.,  $\hat{\nabla}f(\mathbf{x}) \rightarrow 0$



# Applications

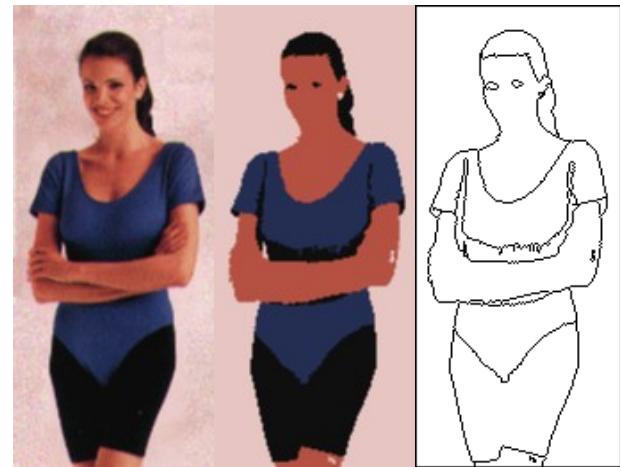
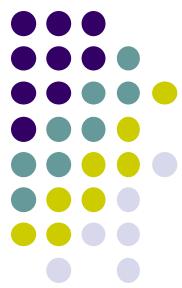
- Pattern recognition
  - Clustering
- Image processing
  - Filtering
  - Segmentation
- Density estimation
  - Density approximation
  - Particle filter
- Mid-level application
  - Tracking
  - Background subtraction



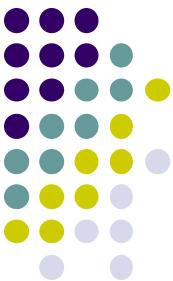
# Summary

- The distance computing plays an important role in data analysis to find out
  - the suitable similarity measurement
  - the intrinsic structure of data
- Further reading on metric learning
- In the next lesson, we will explore more complex data with structure

# Image segmentation based on mean shift

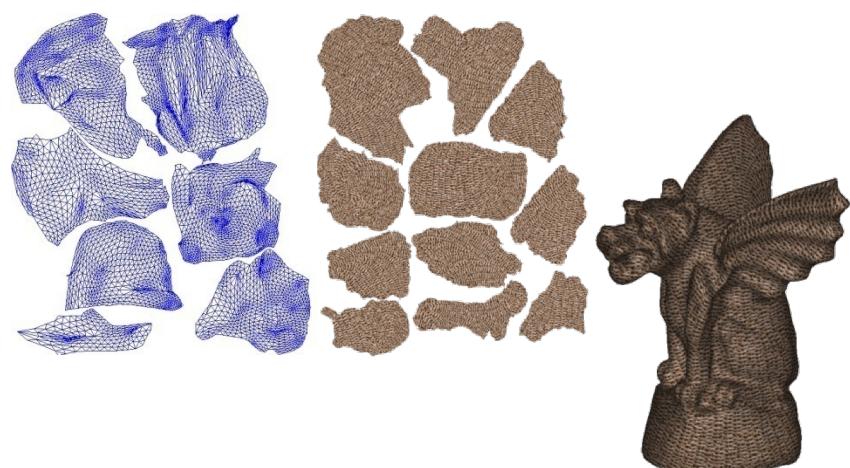
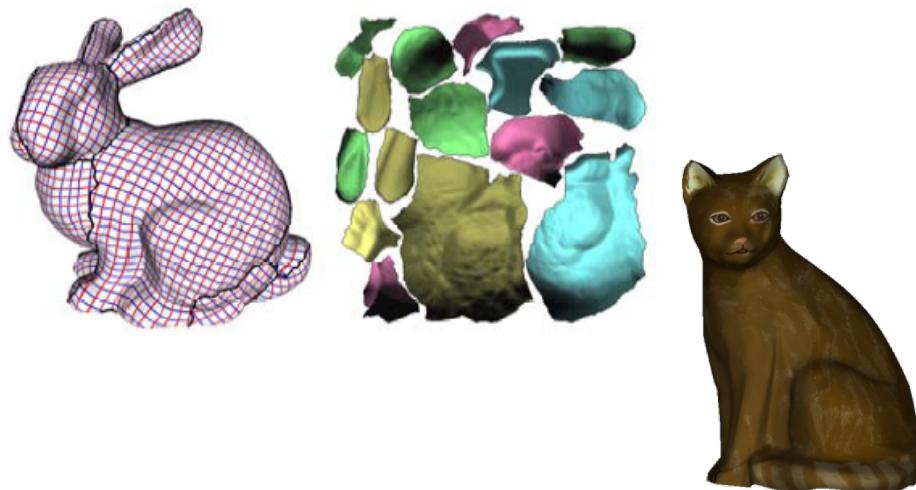


Mean-shift segmentation



# An example: ISO/BLE-charts

- ISO-Charts:
  - ISOMAP + Spectral Clustering + Stretch Minimization
- BLE-Charts (my early JCST paper): [[PDF](#)]
  - Statistical Embedding + Spectral Clustering + Stretch Minimization



# The End

新浪微博: @浙大张宏鑫

微信公众号:

