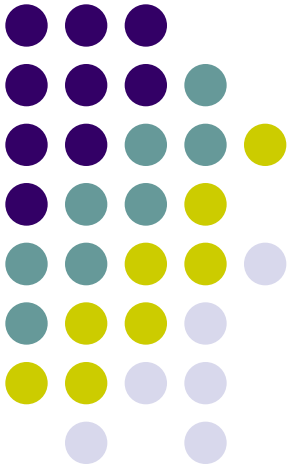
Distance and similarity

Hongxin Zhang
zhx@cad.zju.edu.cn

State Key Lab of CAD&CG, ZJU
2012-03-01



In the last lesson

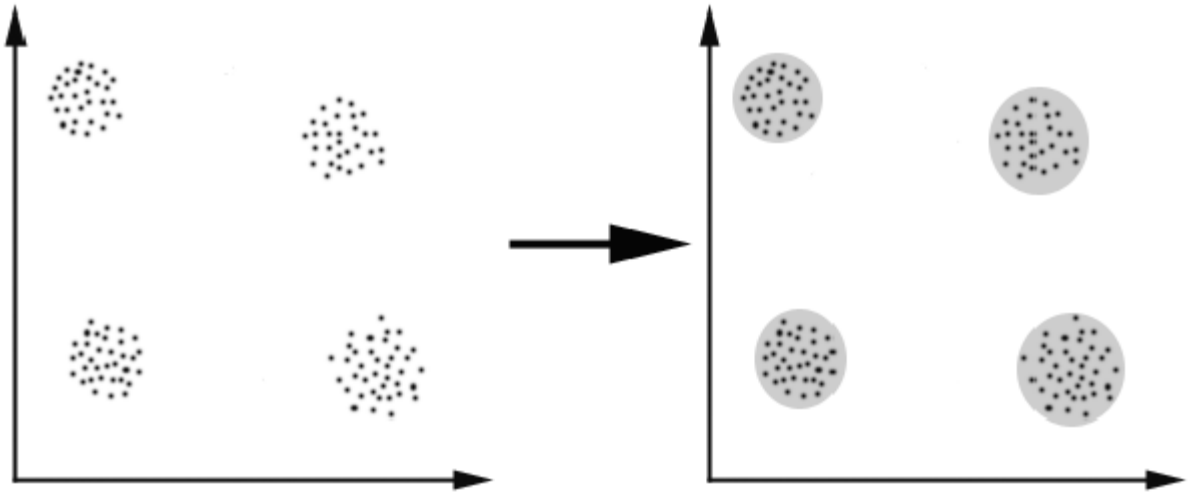
- Data driven decomposition:
  - Data driven curve fitting
  - Singular value decomposition (the power of orthogonal basis)
  - A sort of spectral analysis
- PCA and its related techniques are very useful

In the last lesson

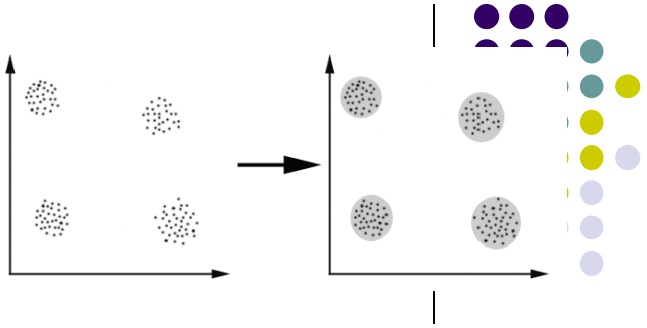
- Mathematical concepts and techniques
  - Least squares (LSQ)
  - Curve fitting
  - Norm (范数) and inner product (内积)
  - Singular value decomposition
  - Eigen vectors and eigen-values
  - Low rank matrix approximation and decomposition

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 x, 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

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

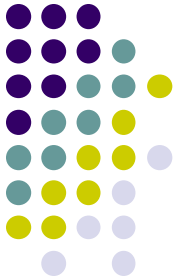
- L^p distance (Minkowsky)

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

$$dist(\mathbf{x},\mathbf{y};\infty) = \max_i \left| x_i - y_i \right|$$

切比雪夫

Distance, Norm and inner product



- Distance

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

- Norm

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

- Inner product

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

$$norm(\mathbf{x}-\mathbf{y};2) = dist(\mathbf{x},\mathbf{y};2)$$

$$dist(x,y;2) = \left(\mathbf{x}\cdot\mathbf{x} + \mathbf{y}\cdot\mathbf{y} - 2\,\mathbf{x}\cdot\mathbf{y} \right)^{1/2}$$

$$= \left(\|\mathbf{x}\|_2^2 + \|\mathbf{y}\|_2^2 - 2\,\mathbf{x}\cdot\mathbf{y} \right)^{1/2}$$

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:

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

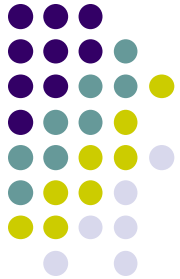
M-distance

- Consider the dependency of different dimensions

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

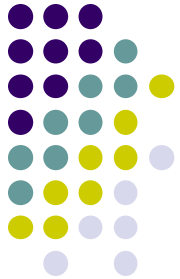
- M is the covariance matrix of data
- Transform invariance

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
    - Local distance approximation: LLE
    - …

Classical Multi-dimensional Scaling

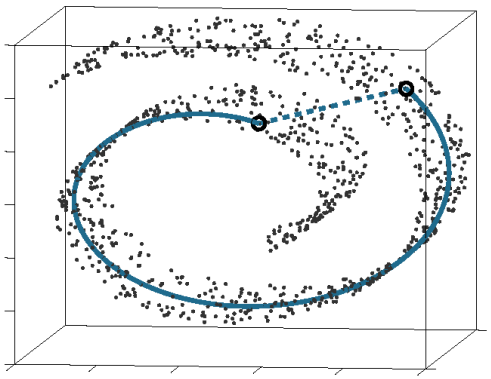


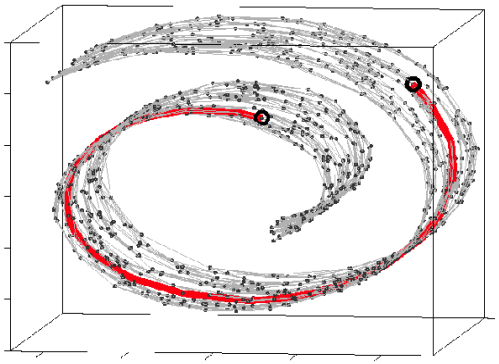
- MDS: 多维标度法
- Main idea:
  - Compute (match) distance between samples
  - Use SVD to find similarity

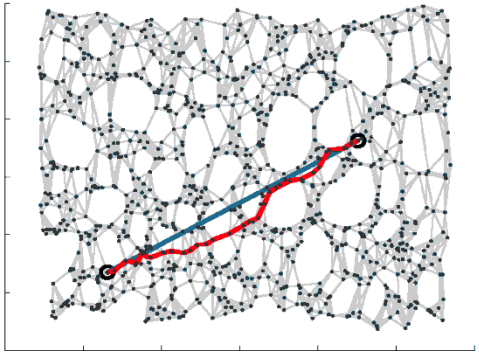
Isomap: (Science 2001) Isometric feature mapping

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

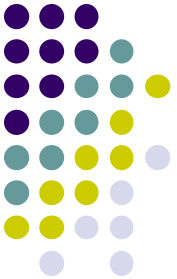
Three steps algorithm



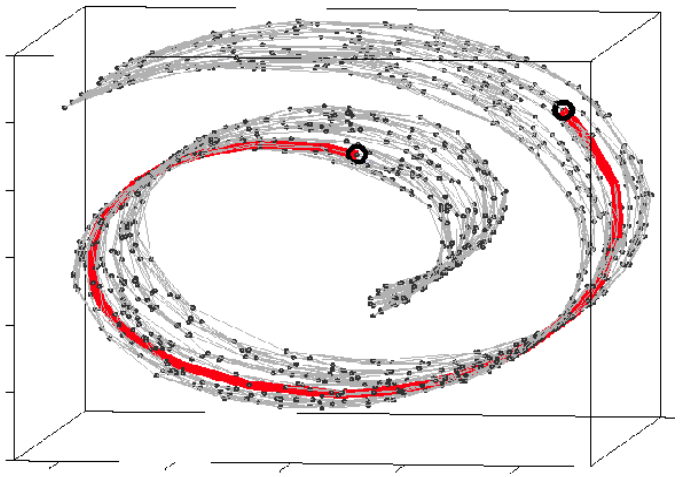




Isomap:
Construct Neighborhood Graph

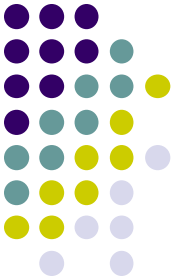


- Determine which points are neighbors, based on the distances d(i,j) .
  - K nearest neighbors
  - ε-radius



- Create a graph G, with edges between neighbors and distance weights.

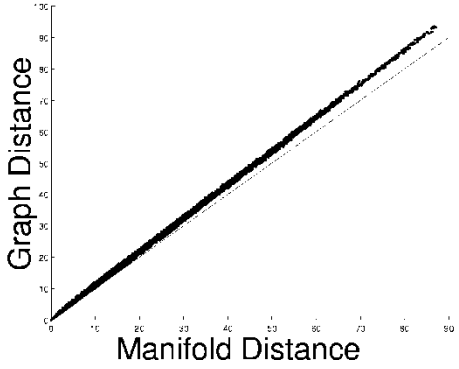
Isomap:
Compute Shortest Paths



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

$$d_G(i,j) = d(i,j) \quad \text{neighborin g i, j}$$

$$d_G(i,j) = \infty \qquad \text{othewise}$$



$$for\ \mathrm{k} = 1,2,\ldots,\mathrm{N}$$

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

Isomap:
Construct d-dimensional Embedding

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

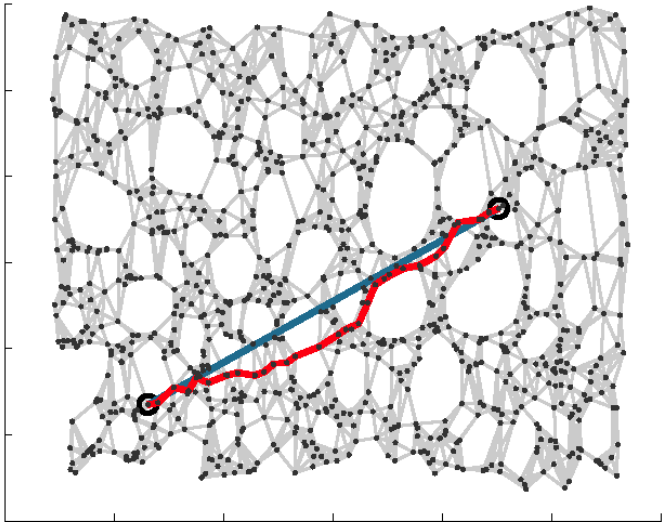
$$E = \left\| \tau(D_G) - \tau(D_Y) \right\|_{L^2}$$

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

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

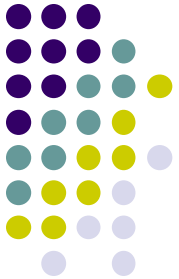
$$and$$

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



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

Isomap:
Classical Multi-dimensional Scaling



$$\mathbf{X}'\mathbf{X} = -\frac{1}{2}\mathbf{J}\mathbf{E}\mathbf{J}$$

E: Euclidian distance matrix

$$\mathbf{B} = -\frac{1}{2}\mathbf{J}\mathbf{M}\mathbf{J}$$

M: Manifold distance matrix

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

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

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

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

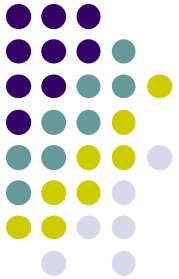
$$\mathbf{E} = \mathbf{c}\mathbf{1}' + \mathbf{1}\mathbf{c}' - 2\mathbf{X}\mathbf{X}'$$

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

$$\mathbf{B} = -\frac{1}{2}\mathbf{J}(\mathbf{c}\mathbf{1}' + \mathbf{1}\mathbf{c}' - 2\mathbf{X}\mathbf{X}')\mathbf{J} = -\frac{1}{2}\mathbf{J}\mathbf{c}\mathbf{0}' - \frac{1}{2}\mathbf{0}\mathbf{c}'\mathbf{J} + \mathbf{J}\mathbf{X}\mathbf{X}'\mathbf{J} = \mathbf{X}\mathbf{X}'.$$

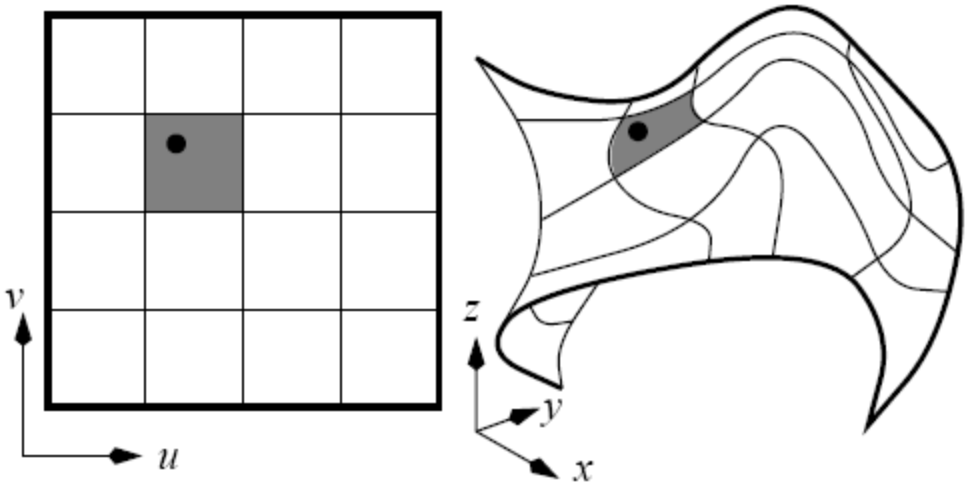
Eigen-structure analysis, SVD again

Isomap:
Classical Multi-dimensional Scaling (2D)



$$
\begin{aligned}
\mathbf{J} &= \text{eye}(n) - \text{ones}(n)./n; \\
\mathbf{B} &= -0.5 * \mathbf{J} * \mathbf{M} * \mathbf{J}; \\
&\quad \% \text{ Find largest eigenvalues+their eigenvectors:} \\
[\mathbf{Q}, \mathbf{L}] &= \text{eigs}(\mathbf{B}, 2, \text{'LM'}); \\
&\quad \% \text{ Extract the coordinates:} \\
\text{newy} &= \text{sqrt}(\mathbf{L}(1,1)). * \mathbf{Q}(:,1); \\
\text{newx} &= \text{sqrt}(\mathbf{L}(2,2)). * \mathbf{Q}(:,2);
\end{aligned}
$$

Isomap: application texture mapping



(a) (b)

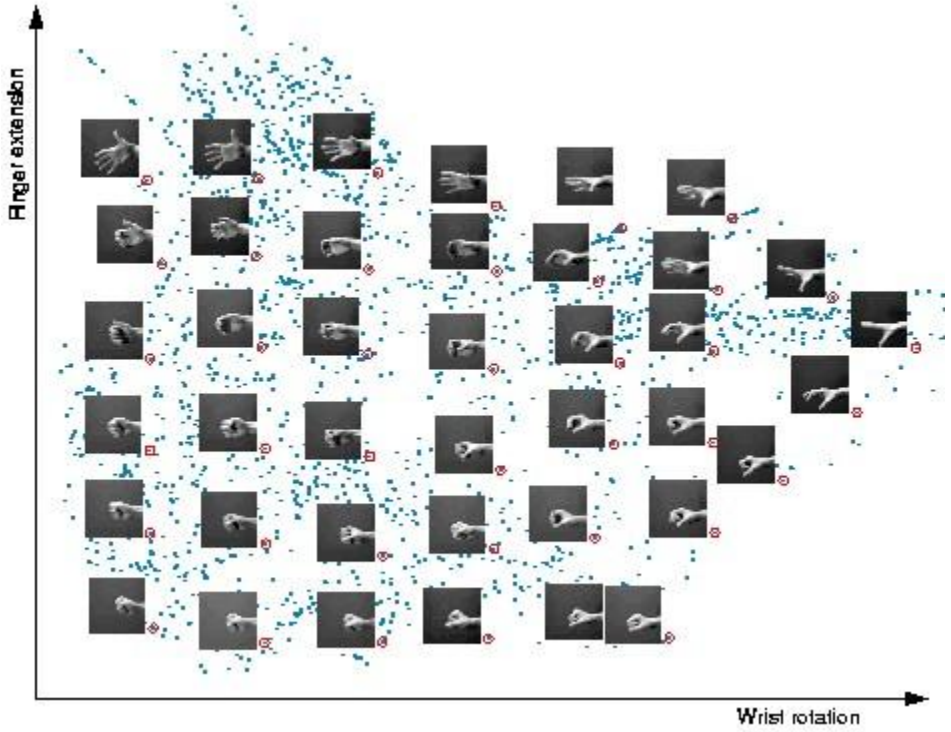
Fig. 3. An example of a face flattening. (a) A 3D reconstruction of a face. (b) The flattened texture image of the face.

(a)

(b)

Isomap: Examples

- *N*=2000 images 64x64 pixels K=6

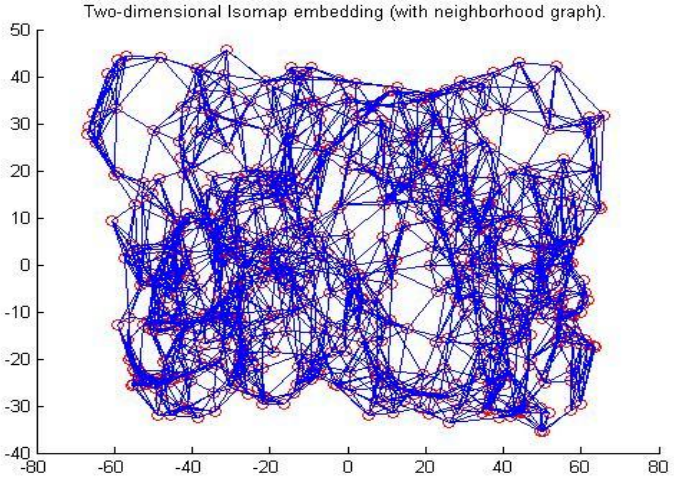


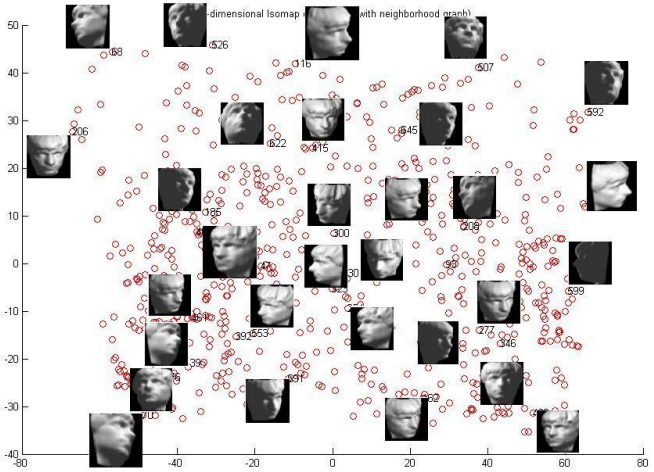
Isomap:
More Results

Input: 698
images of 64x64

K=7, d=2

Outputs:

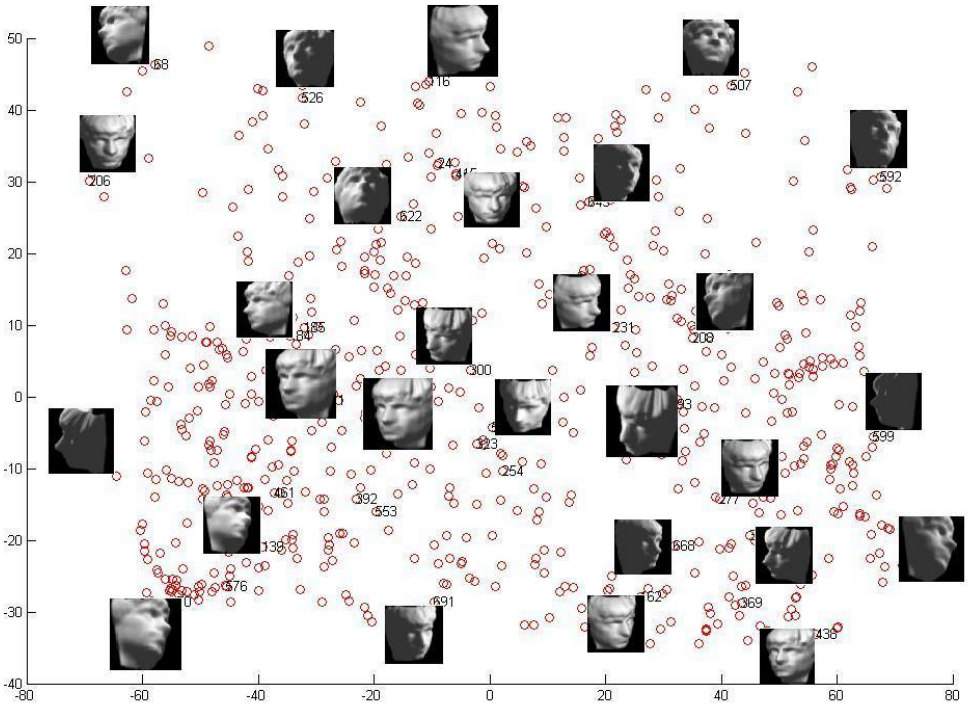




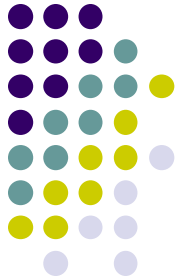
Isomap:
More Results

- Same inputs, but this time with d=3

698 images of 64x64 K=7



Locally Linear Embedding (LLE)

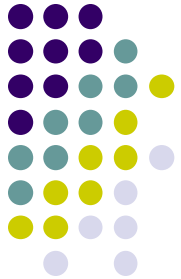


- Recovers global nonlinear structure from locally linear fits.
- Each data point and it's neighbors is expected to lie on or close to a locally linear patch.
- Each data point is constructed by it's neighbors:

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

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

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 constrains:

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

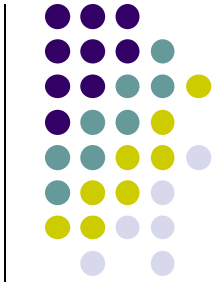
$$\sum_j W_{ij} = 1$$

- Solution (using Lagrange multipliers):

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

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

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

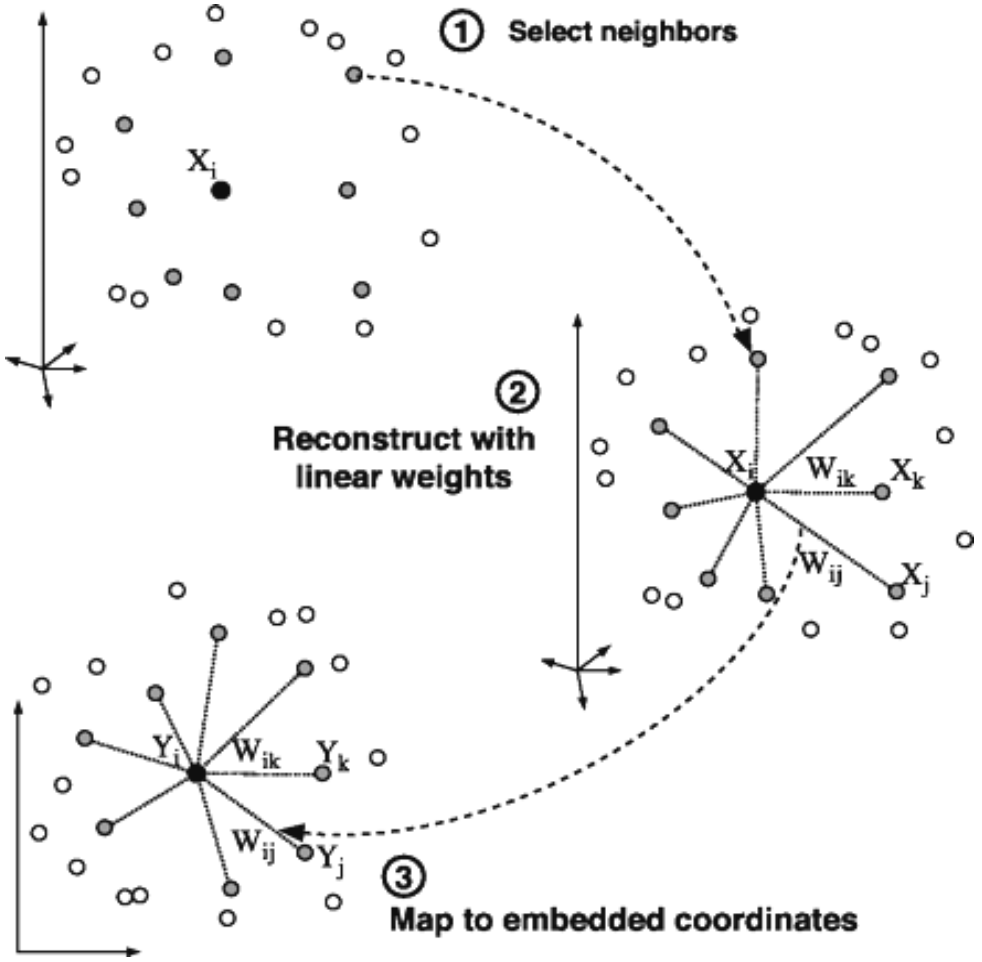
  Quadratic form: $\phi(Y) = \sum_{ij} M_{ij} (\vec{Y}_i \vec{Y}_j)$

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

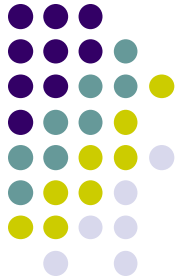
- Solution: compute bottom d+1 eigenvectors of M. (discard the last one)

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 Xi to every other point Xj

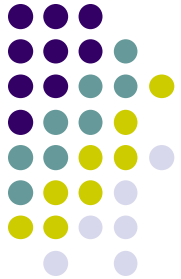
    find the K smallest distances

    assign the corresponding points to be neighbors of Xi

end

http://www.cs.toronto.edu/~roweis/lle/algorithm.html

LLE:
Algorithm Pseudocode (II)



Solve for reconstruction weights W.

for i=1:N

create matrix Z consisting of all neighbors of Xi
subtract Xi from every column of Z

compute the local covariance C=Z'*Z

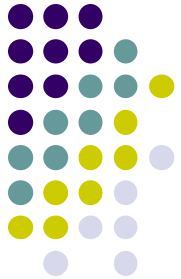
solve linear system C*w = 1 for w

set Wij=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.

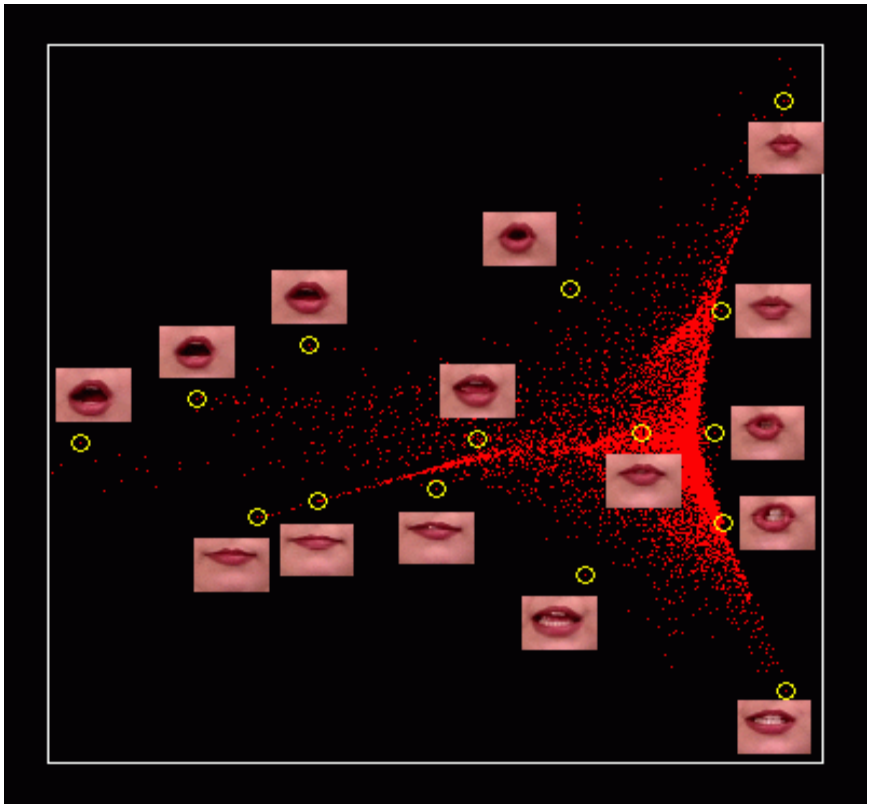
create sparse matrix M = (I-W)'*(I-W)

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...] with eigenvalue zero)

LLE: Example

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



Spectral clustering

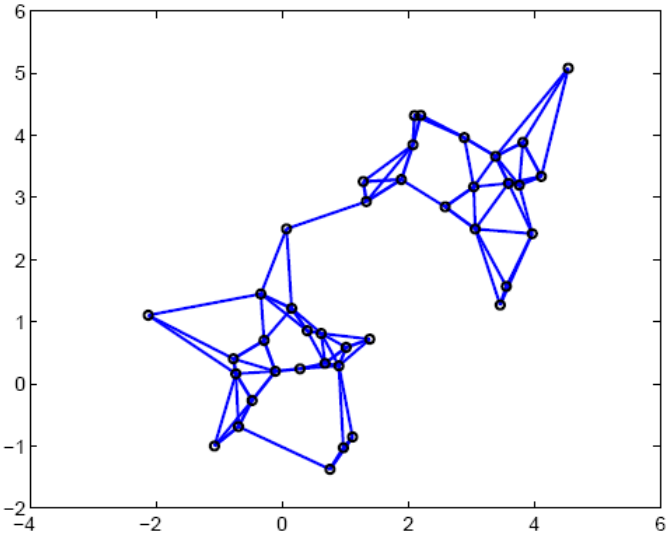
- The spectral clustering method we define relies on a random walk representation over the points. We construct this in three steps
  1. a nearest neighbor graph
  2. similarity weights on the edges:

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

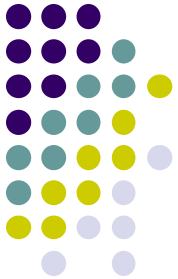
     where $W_{ii} = 1$ and the weight is zero for non-edges.

  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

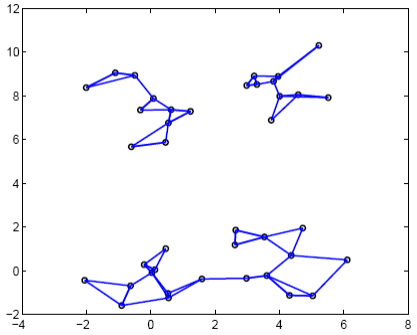
$$i_1 \sim P_{i_0\, i_1}, \qquad P_{ij} = \frac{W_{ij}}{W_{i.}}, \quad \text{where } W_{i.} = \sum_j W_{ij}$$

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

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

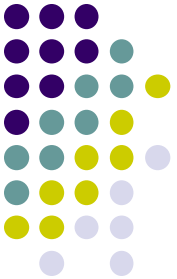
$$\ldots$$

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



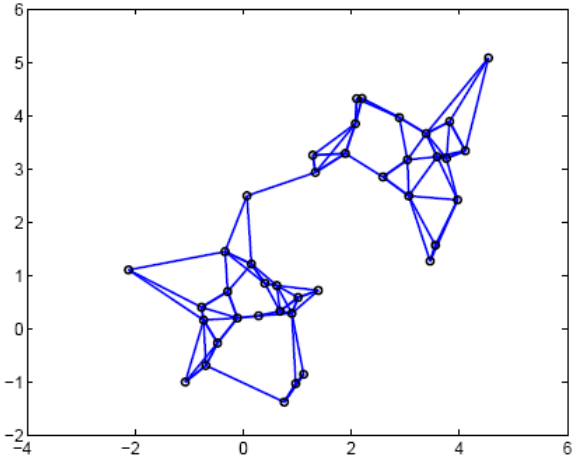
where $P^t = PP \ldots P$ (t matrix products) and $[\cdot]_{ij}$ denotes the i, j component of the matrix.

Random walk and clustering

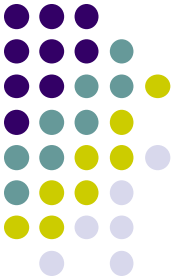


- The distributions of points we end up in after t steps converge as t increases. If the graph is connected, the resulting distribution is independent of the starting point

  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

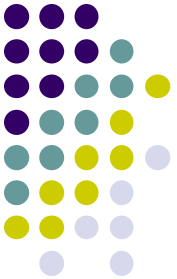
$$P = D^{-1}W$$

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

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

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

Eigenvalues/vectors cont'd



- The symmetric matrix is related to P^t since

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

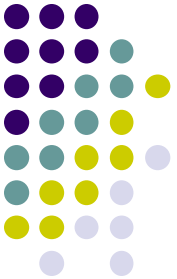
This allows us to write the t step transition probability matrix in terms of the eigenvalues/vectors of the symmetric matrix

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

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

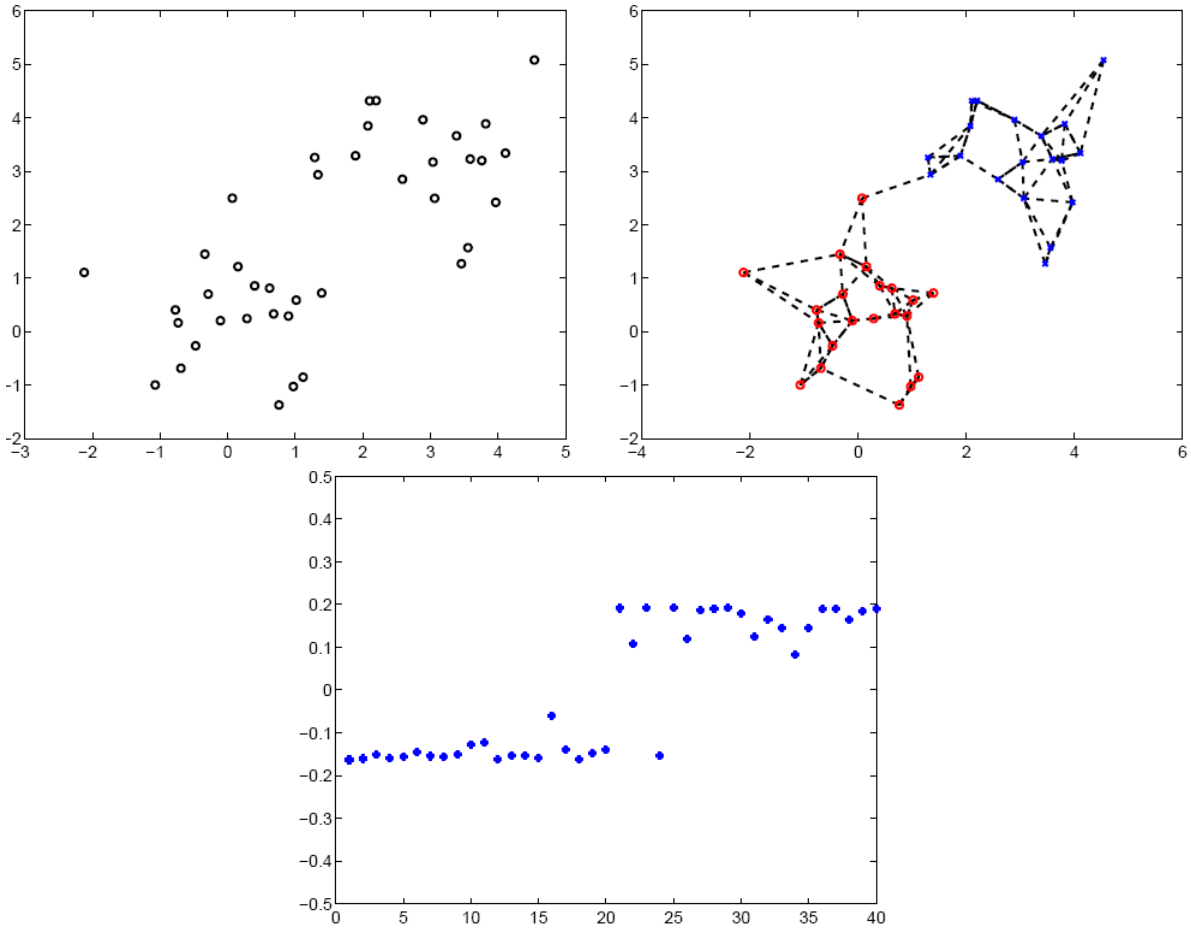
$$P^t \approx P^\infty + D^{-\frac{1}{2}} \left(\lambda_2^t \, \mathbf{z}_2 \mathbf{z}_2^T \right) D^{\frac{1}{2}}$$

  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



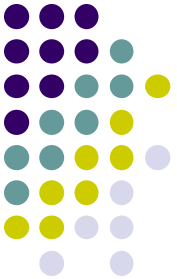
Components of the eigenvector corresponding to the second largest eigenvalue

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

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

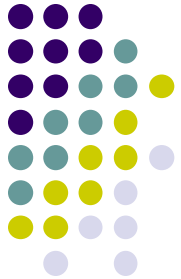
b) Average linkage:

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

c) Centroid linkage:

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

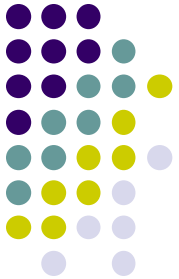
Hierarchical (bottom-up) clustering



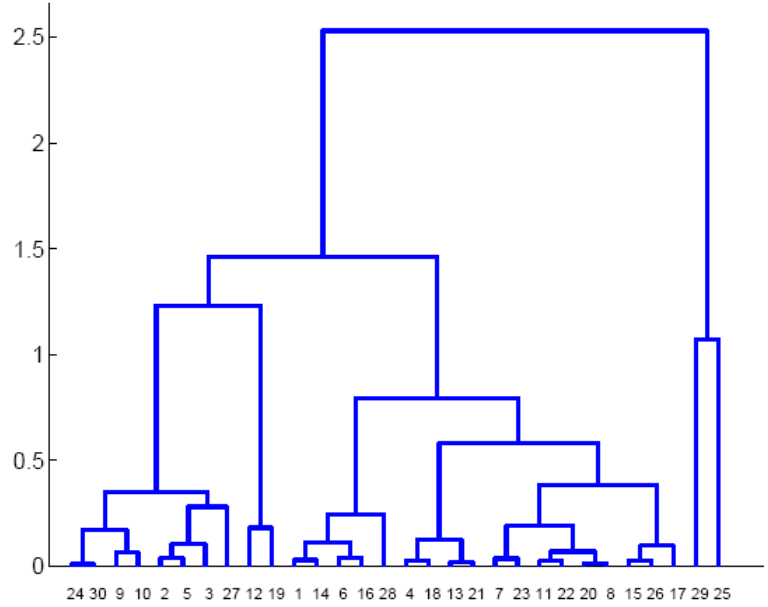
- Hierarchical agglomerative clustering: we sequentially merge the pair of “closest” points/clusters

- 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(xi, xj) between two points
  2. distance measure between clusters (cluster linkage)

Hierarchical (bottom-up) clustering

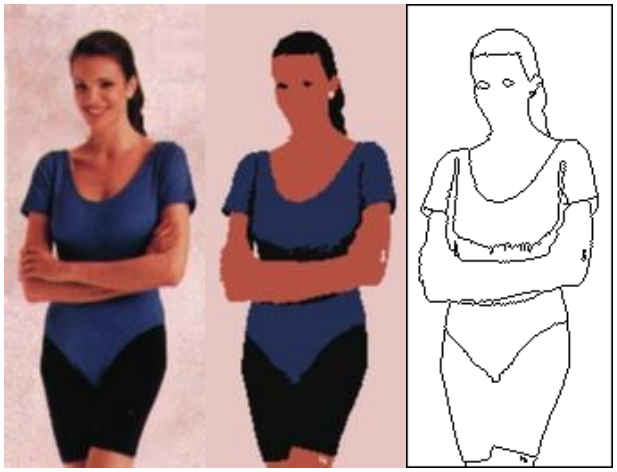


- A dendrogram representation of hierarchical clustering



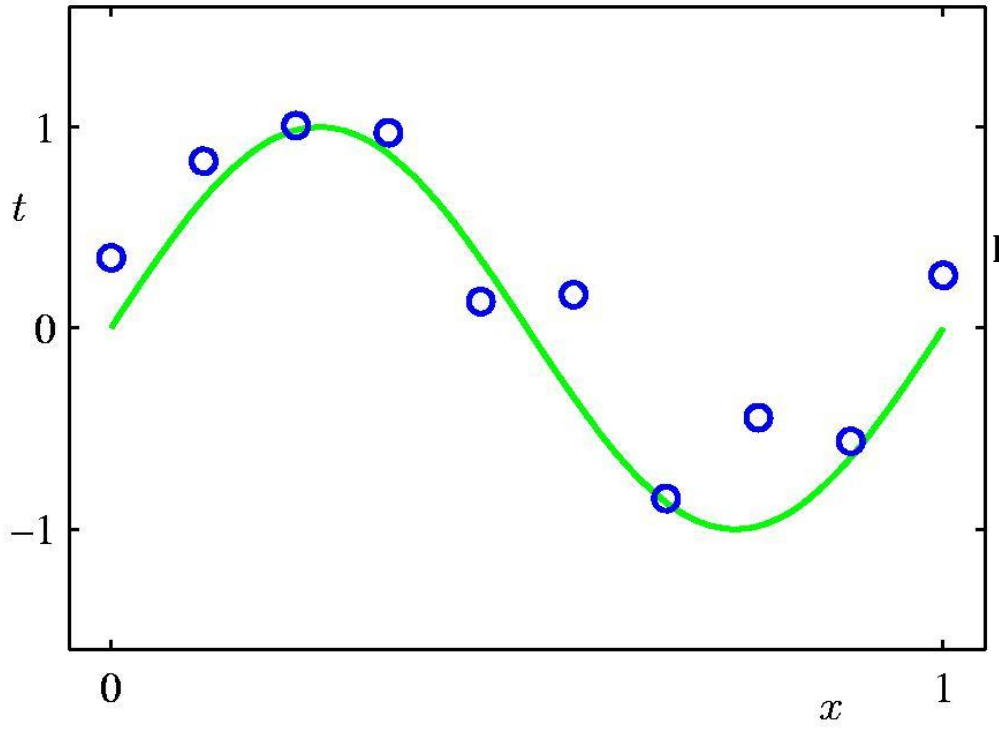
The height of each pair represents the distance between the merged clusters; the specific linear ordering of points is chosen for clarity

Clustering and image segmentation



Mean-shift segmentation

Regression revisit: Polynomial Curve Fitting



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

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

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

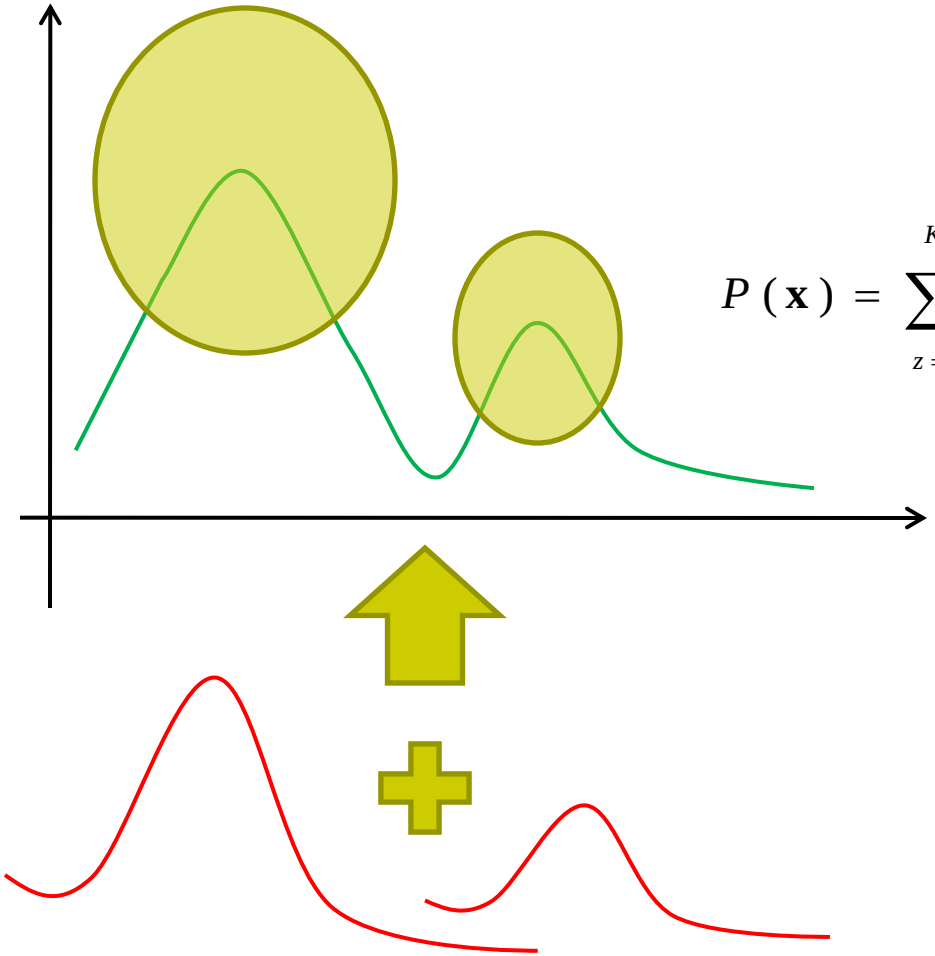
Normal equation

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

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

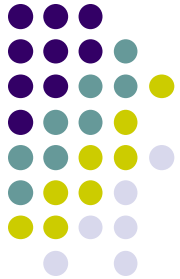
Basis function

Regression revisit: alternative approach



$$P(\mathbf{x}) = \sum_{z=1}^{K} P(Z = z \mid \pi) N(\mathbf{x} \mid \mu_z, \Sigma_z)$$

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 $N(\mu_z, \Sigma_z)$

$$P(\mathbf{x}) = \sum_{z=1}^{K} P(Z = z \mid \pi) N(\mathbf{x} \mid \mu_z, \Sigma_z)$$

- What object function shall we optimize?
  - Maximize data likelihood

Mixtures of Gaussians (cont.)

- Multivariate Gaussian model

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

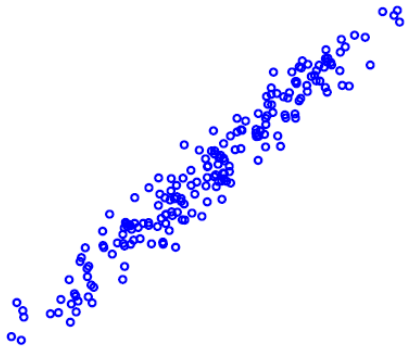
- How to generate it?

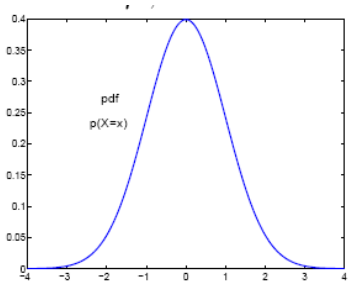
$$F_{\mu,\sigma^2}(x) = \int_{-\infty}^{x} p(z|\mu,\sigma^2)dz$$

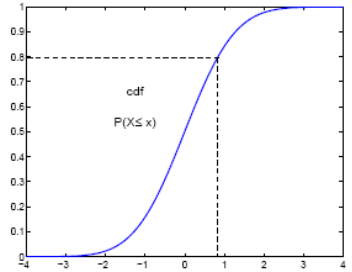
$$u \sim \text{Uniform}(0,1) \Rightarrow x = F^{-1}_{\mu,\sigma^2}(u) \sim p(x|\mu,\sigma^2)$$

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

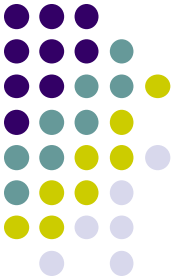
$$\mathbf{x} = \Sigma^{1/2}\mathbf{z} + \mu$$







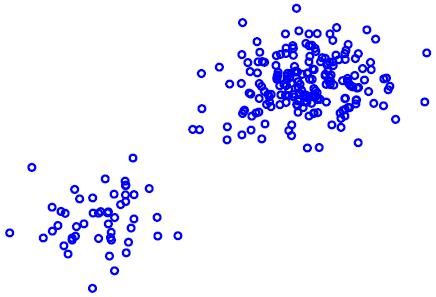
Multi-variate density estimation



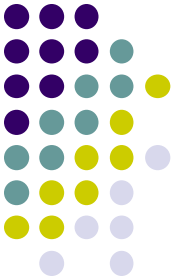
- A mixture of Gaussians model

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

where $\theta = \{p_1, \ldots, p_k, \mu_1, \ldots, \mu_k, \Sigma_1, \ldots, \Sigma_k\}$ 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

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

- Wishart prior

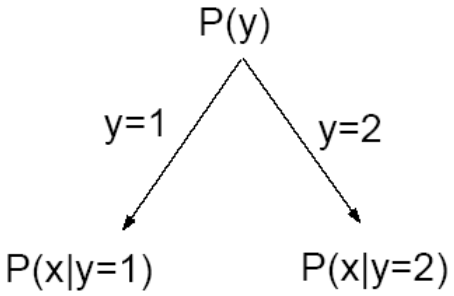
$$P(\Sigma|S, n') \propto \frac{1}{|\Sigma|^{n'/2}} \exp\left(-\frac{n'}{2}\text{Trace}(\Sigma^{-1} S)\right)$$

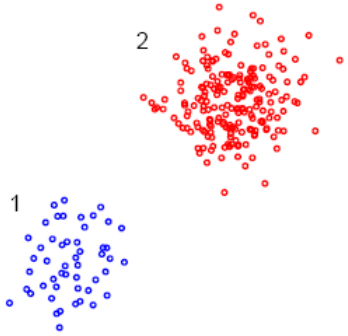
S = "prior" covariance matrix

n' = equivalent sample size

Mixture density

- Data generation process:



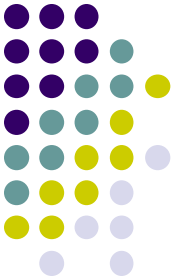


$$p(\mathbf{x}|\theta) = \sum_{j=1,2} P(y=j) \cdot p(\mathbf{x}|y=j) \quad \text{(generic mixture)}$$

$$= \sum_{j=1,2} p_j \cdot p(\mathbf{x}|\mu_j, \Sigma_j) \quad \text{(mixture of Gaussians)}$$

- 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

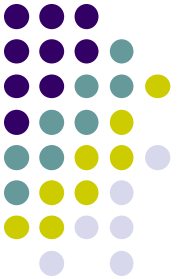
$$p(\mathbf{x}|\theta) = \sum_{j=1,2} p_j p(\mathbf{x}|\mu_j, \Sigma_j)$$

- 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



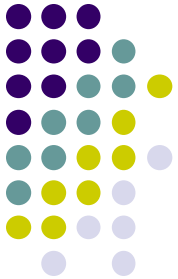
- Consider sampling $\mathbf{x}$ from the mixture density, then y from the posterior over the components given $\mathbf{x}$, and finally $\mathbf{x}'$ from the component density indicated by y:

$$\begin{aligned} \mathbf{x} &\sim p(\mathbf{x}|\theta) \\ y &\sim P(y|\mathbf{x},\theta) \\ \mathbf{x}' &\sim p(\mathbf{x}'|y,\theta) \end{aligned}$$

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

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

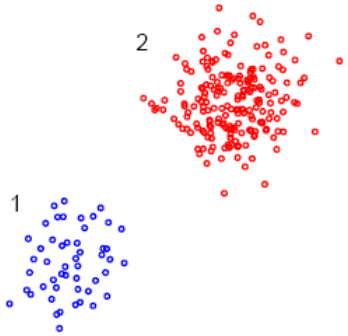
Mixture density estimation



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

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

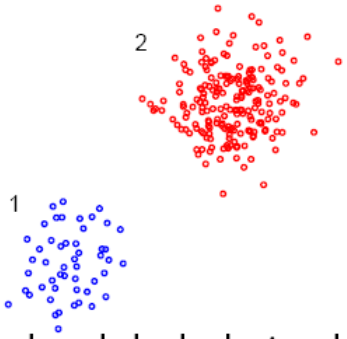
- If each example $\mathbf{x}_i$ in the training set were labeled $y_i = 1, 2$ according to which mixture component (1 or 2) had generated it, then the estimation would be easy.



- 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



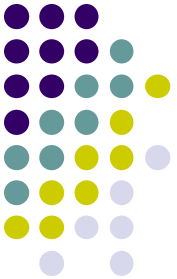
- If $\hat{n}_j$ is the number of examples labeled j, then for each $j = 1, 2$ we set

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

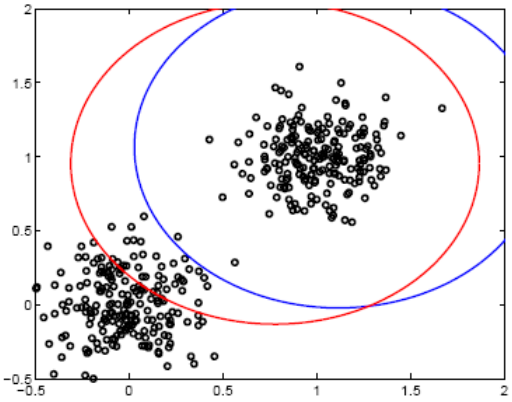
$$\hat{\mu}_j \leftarrow \frac{1}{\hat{n}_j} \sum_{i:y_i=j} \mathbf{x}_i$$

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

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:



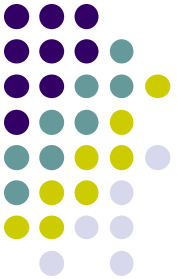
$$\hat{p}(j|i) \leftarrow P(y_i = j|\mathbf{x}_i, \theta)$$

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

- 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

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

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

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

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

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

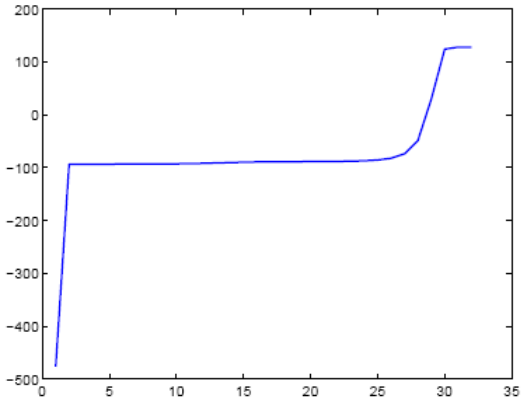
$$\hat{\Sigma}_j \leftarrow \frac{1}{\hat{n}_j} \sum_{i=1}^{n} \hat{p}(j|i)\, (\mathbf{x}_i - \hat{\mu}_j)(\mathbf{x}_i - \hat{\mu}_j)^T$$

The EM-algorithm

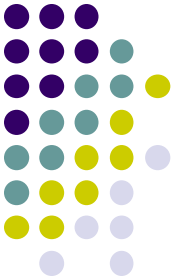
- Each iteration of the EM-algorithm *monotonically* increases the (log-)likelihood of the n training examples $\mathbf{x}_1, \ldots, \mathbf{x}_n$:

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

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 $l(\theta; D)$

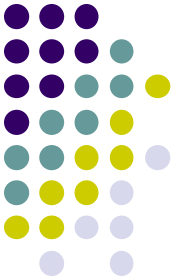
  E-step: evaluate the expected complete log-likelihood

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

  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

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

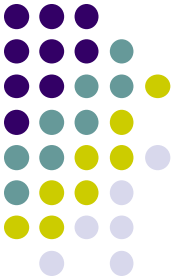
we only need to modify the M-step of the EM-algorithm.

Specifically, in the M-step, we find find θ 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

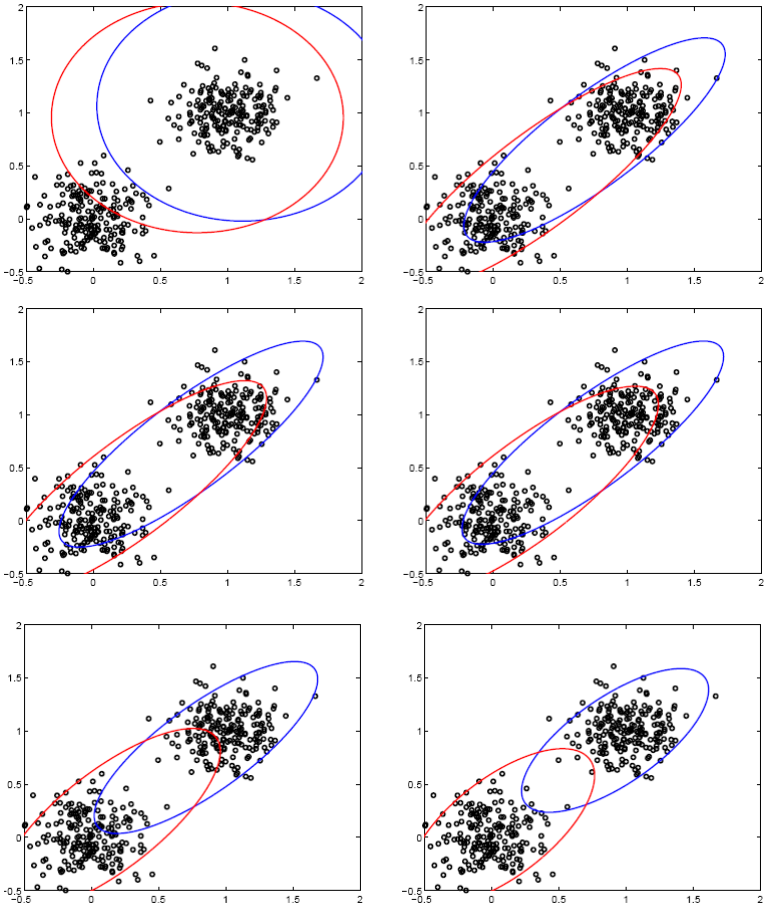


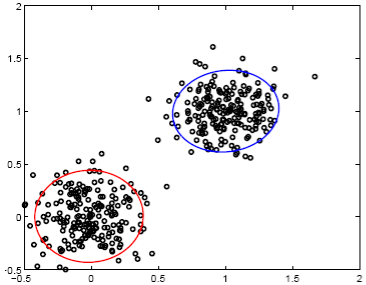
- As a simple strategy for selecting the appropriate number of mixture components, we can find k that minimize the following asymptotic approximation to the description length:

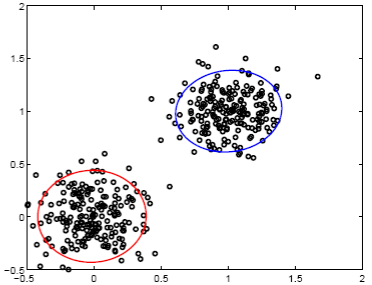
$$\text{DL} \approx -\log p(\text{data}|\hat{\theta}_k) + \frac{d_k}{2}\log(n)$$

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.

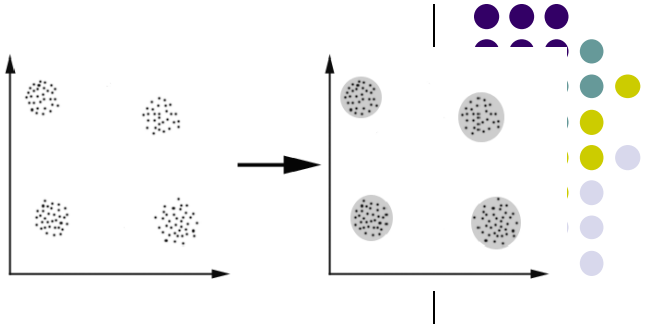
Mixture density estimation: example







K-means clustering



Given data <$x_1 \ldots x_n$>, and K, assign each x_i to one of K clusters, $C_1 \ldots C_K$, minimizing $J = \sum_{j=1}^{K} \sum_{x_i \in C_j} ||x_i - \mu_j||^2$

Where μ_j is mean over all points in cluster C_j

K-Means Algorithm:

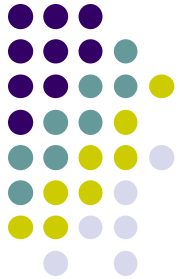
Initialize $\mu_1 \cdots \mu_K$ randomly

Repeat until convergence:

1. Assign each point x_i to the cluster with the closest mean μ_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

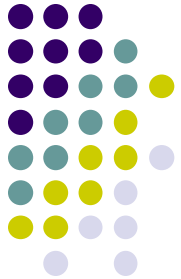


- Both are iterative algorithms to assign points to clusters

- Objective function
  - K Means: minimize $J = \sum_{j=1}^{K} \sum_{x_i \in C_j} ||x_i - \mu_j||^2$
  - MoG: maximize likelihood $P(X|\theta)$

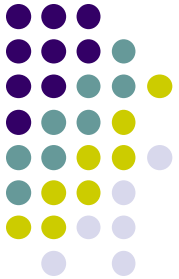
- MoG the more general formulation
  - Equivalent to K Means when $\Sigma_k = \sigma I$, and $\sigma \rightarrow 0$

Disadvantage of K-means and MOG



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

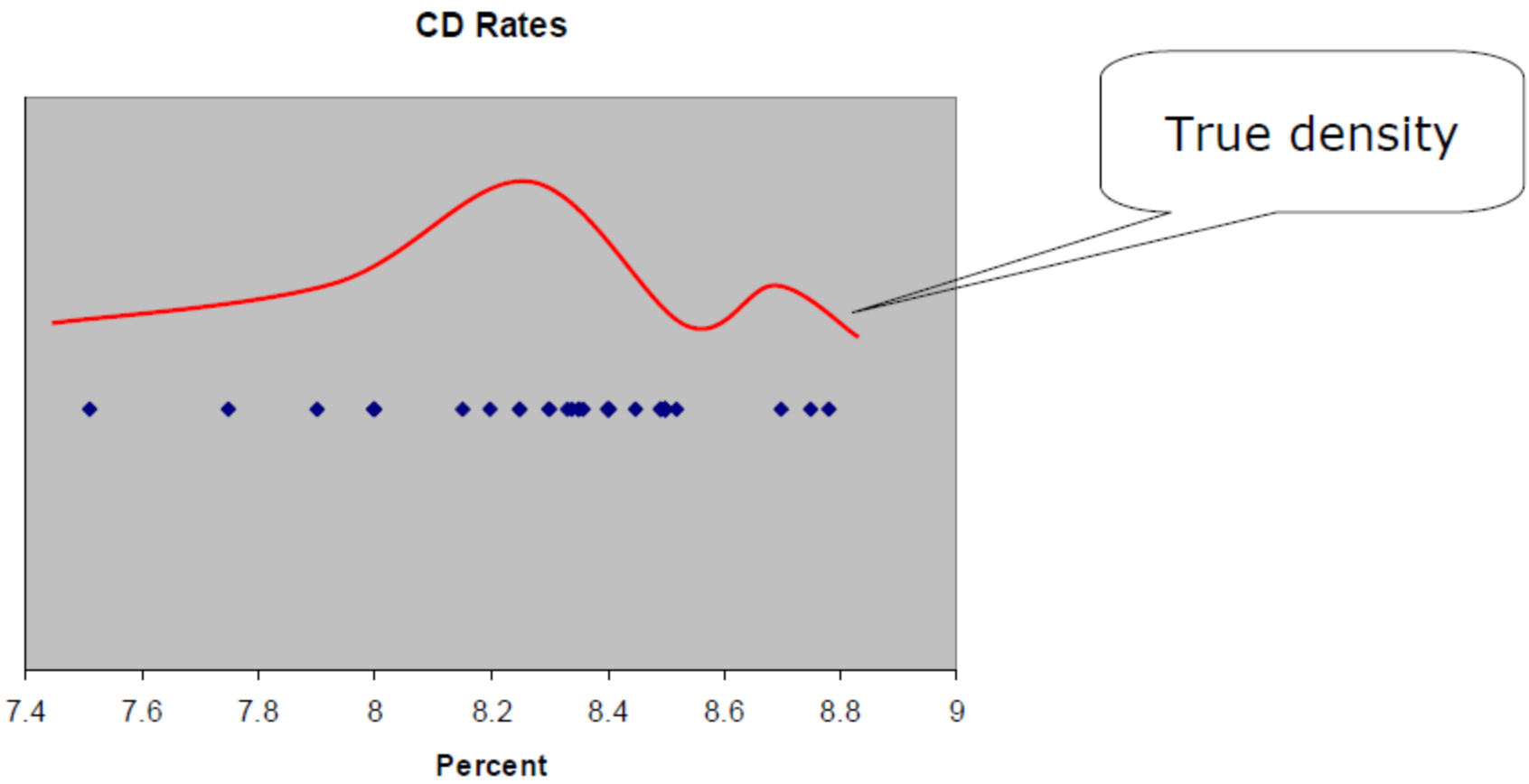
Mean shift



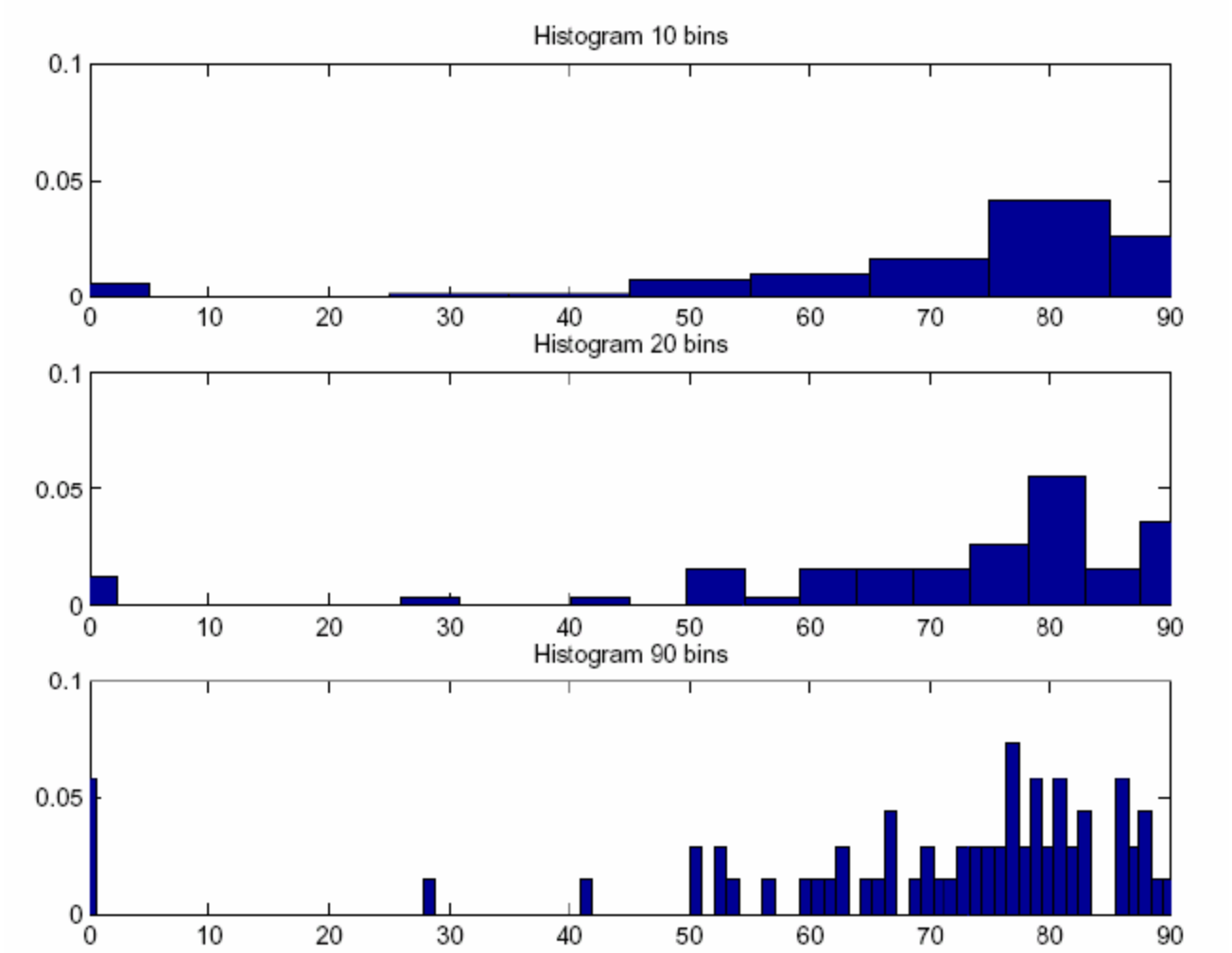
- First proposed by Fukunaga in 1970's
- Wildly used since 1998
  - In computer vision
  - And other areas

- The following several slides is mainly from:
  - http://www.cs.cornell.edu/courses/cs664/2005fa/Lectures/lecture3.pdf

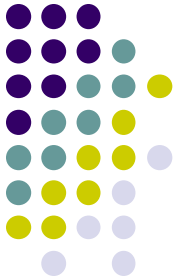
Density estimation



Histogram representation

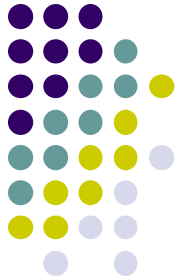


Histogram-based estimates



- 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

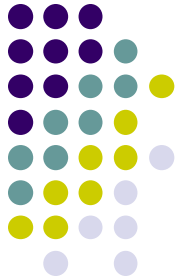


- To estimate the density, count the number of nearby data points
  - Like histogramming with sliding bins
  - Avoid bin-placement artifacts

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

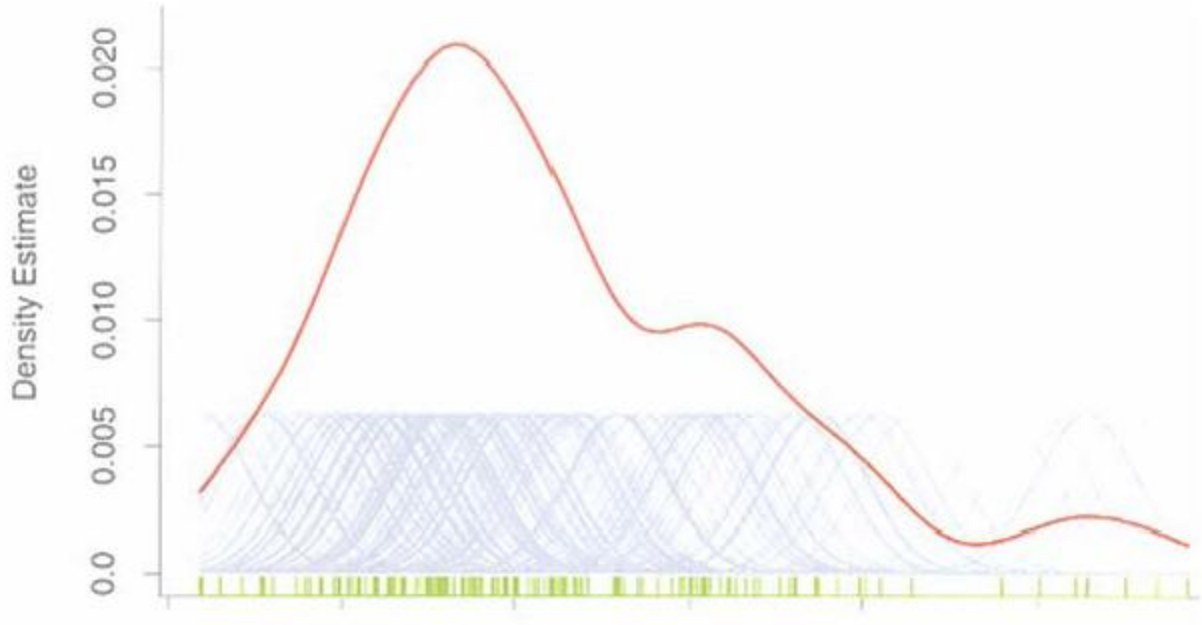
  - We can fix ε and compute this quantity, or we can fix the quantity and compute ε

Parzen estimation



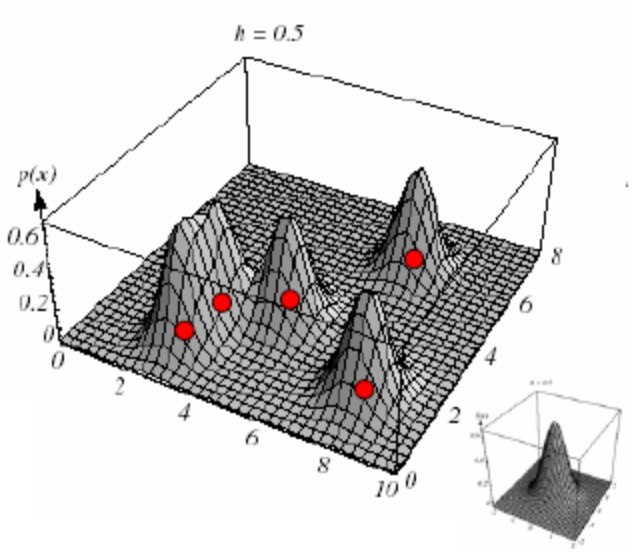
- Each observed data increases our estimate of the probability nearby
  - Simplest case: raise the probability uniformly within a fixed radius
    - Place a fixed-height “box” at each data point, add them up to get the density estimate
  - This is nearest neighbor with fixed ε
- More generally, you can 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”

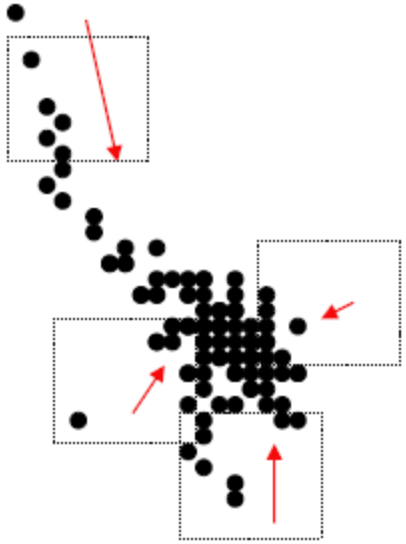
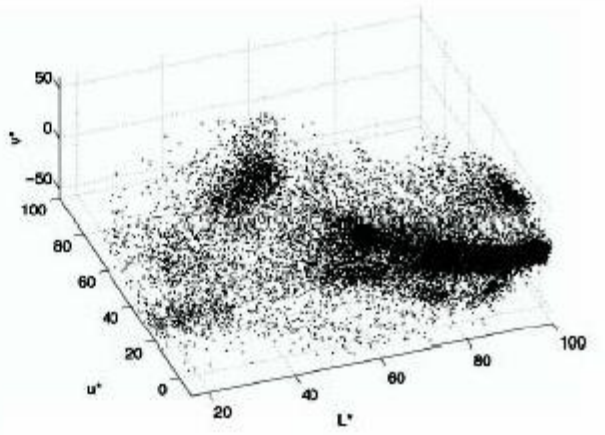
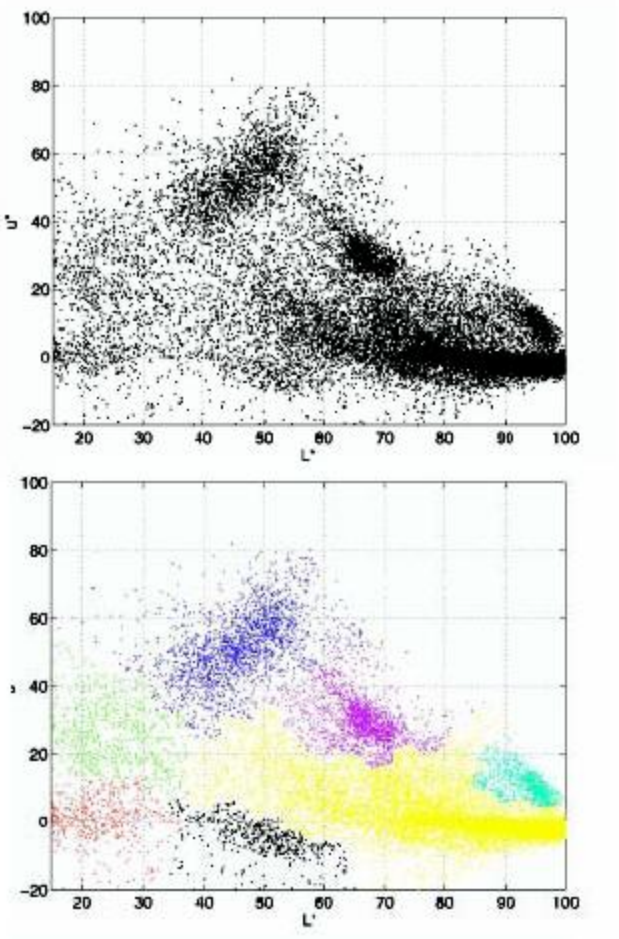
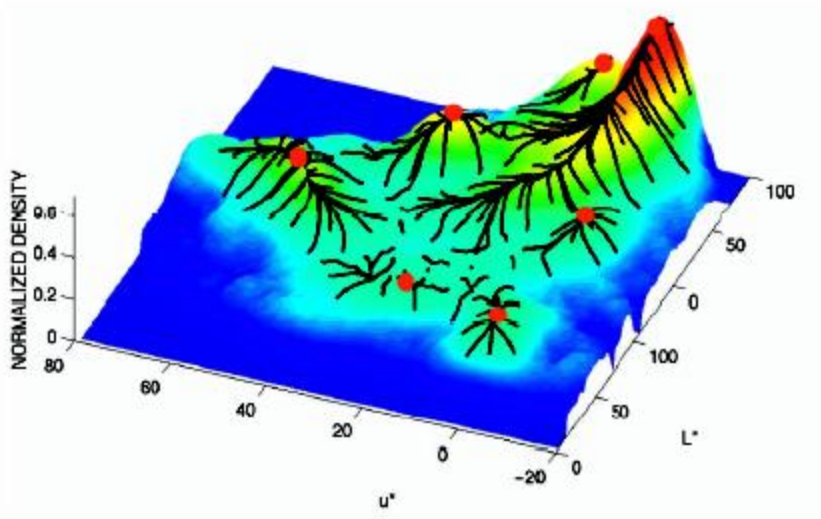


Image and histogram

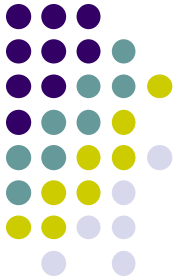


Local modes





Kernel Density Estimation



- Multivariate kernel density estimation

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

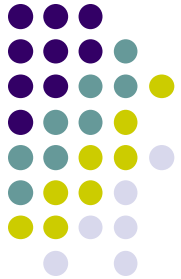
- Kernels
  - Gaussian

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

  - 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 x
  1. Compute the mean shift vector

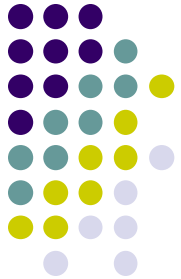
$$\hat{\nabla} f(\mathbf{x}) = \frac{2}{nh^{d+2}} \sum_{i=1}^{n} K_N\left(\frac{\mathbf{x}-\mathbf{x}_i}{h}\right) \left[\frac{\sum_{i=1}^{n} \mathbf{x}_i K_N\left(\frac{\mathbf{x}-\mathbf{x}_i}{h}\right)}{\sum_{i=1}^{n} K_N\left(\frac{\mathbf{x}-\mathbf{x}_i}{h}\right)} - \mathbf{x} \right]$$

  2. Translate density estimation window:

$$\mathbf{x}^{(t+1)} \leftarrow \mathbf{x}^{(t)} + \hat{\nabla} f(\mathbf{x}^{(t)})$$

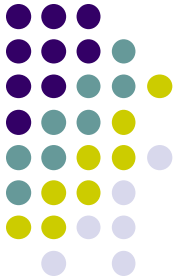
  3. Iterate steps 1. and 2. until convergence
     i.e., $\hat{\nabla} f(\mathbf{x}) \rightarrow 0$

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