Multidimensional scaling: part I

STAT 6312
Department of mathematics, UTA

ESL 14.5.3-4

Optional reading: ESL 14.5.5

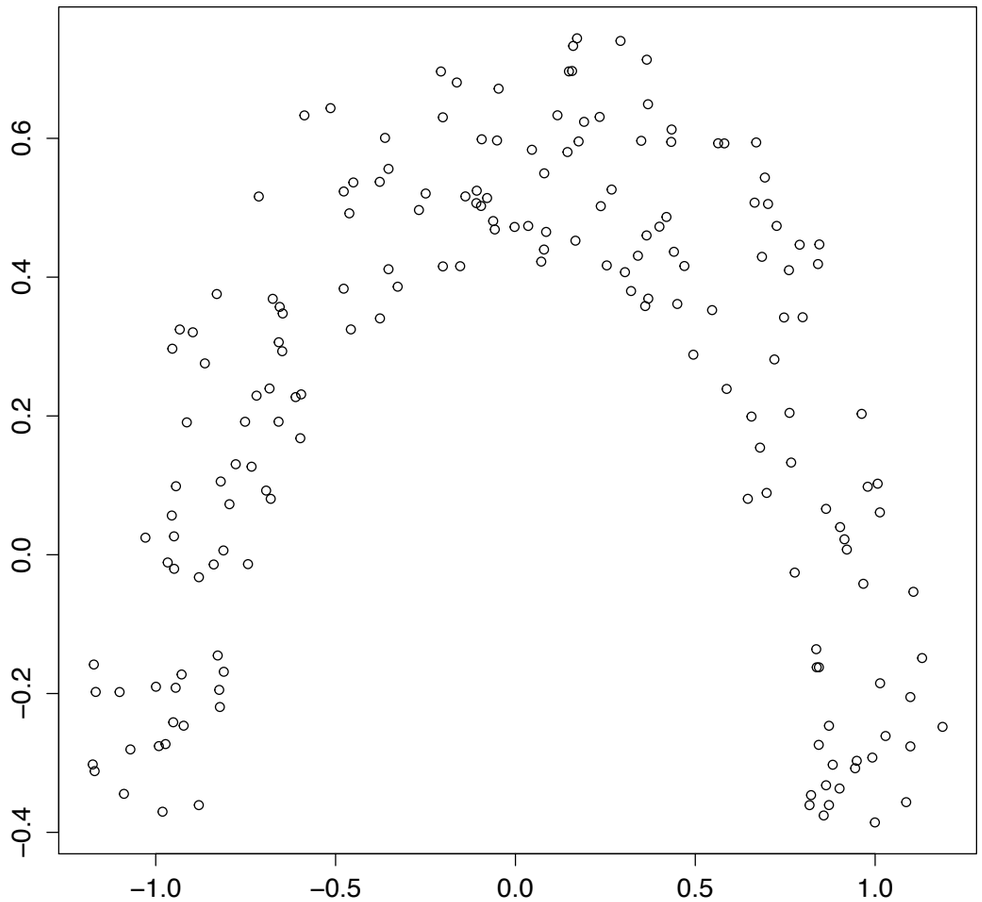
Kernel PCA

- So far, we've discussed linear PCA. It finds a line that has the largest variance in the feature space X.

- What if we want to find a curve instead of a line?

- Simple answer: take non-linear transformation on each row in X and apply linear PCA

- e.g. $x_i=(x_{i1},x_{i2})$, $h(x_i)=(x_{i1},x_{i2},x_{i1}x_{i2},x_{i1}^2,x_{i2}^2)$



- In linear PCA, PC loadings are obtained from the gram matrix ($X'X=V_X D_X^2 V_X'$), and column of V is linear combination of rows of X. Recall $S_X = X'X / N$, and it can be rewritten using row vectors as

$$S_X = \frac{1}{N}\sum_{i=1}^{N} x_i x_i^T$$

- We do feature mapping h() from the original feature space onto an enlarged feature space. $h(x_i \in R^p) \in R^M (p < M)$.

$$S = \frac{1}{N}\sum_{i=1}^{N} h(x_i) h(x_i)^T$$

- The covariance matrix for transformed variables (assume they are column centered) is an M by M matrix.

- Then, one can find m eigenvectors such that $Sv = \lambda v$.

- The challenge is how to simplify this?

Kernel tricks

- Non-linear transformation increases the dimensionality in the feature space.

- In SVM, we used the kernel tricks to find the SVM solution in the higher dimensional feature space without explicitly knowing complex feature maps.

- We can apply the kernel tricks for PCA to capture higher oder dependency structure in the feature space.

- Suppose v_i is an eigenvector.

$$Sv_i = \frac{1}{N}\sum_{i=1}^{N} h(x_i)h(x_i)^T v_i = \lambda_i v_i \text{ and } v_i = \sum_{k=1}^{N} \alpha_{ik} h(x_k) \quad \text{Goal: find} \quad \alpha_i = \begin{pmatrix} \alpha_{i1} \\ \alpha_{i2} \\ \vdots \\ \alpha_{iN} \end{pmatrix} \text{ for } \forall i$$

- $$\frac{1}{N}\sum_{i=1}^{N} h(x_i)\left[\sum_{k=1}^{N} \alpha_{ik} h(x_i)^T h(x_k)\right] = \lambda_i \sum_{k=1}^{N} \alpha_{ik} h(x_k)$$

$$\frac{1}{N}\sum_{i=1}^{N} h(x_j)^T h(x_i)\left[\sum_{k=1}^{N} \alpha_{ik} h(x_i)^T h(x_k)\right] = \lambda_i \sum_{k=1}^{N} \alpha_{ik} h(x_j)^T h(x_k)$$

$$\frac{1}{N}\sum_{i=1}^{N} K(x_j, x_i)\left[\sum_{k=1}^{N} \alpha_{ik} K(x_i, x_k)\right] = \lambda_i \sum_{k=1}^{N} \alpha_{ik} K(x_j, x_k)$$

kernel matrix (N by N)

$$\mathbf{K} = \begin{pmatrix} K(x_1, x_1) & \cdots & K(x_1, x_N) \\ \vdots & \ddots & \vdots \\ K(x_N, x_1) & \cdots & K(x_1, x_N) \end{pmatrix}$$

$$\mathbf{K}^2 \alpha_i = \lambda_i N \mathbf{K} \alpha_i \quad \Rightarrow \quad \mathbf{K} \alpha_i = \lambda_i N \alpha_i$$

Right differs only by eigenvector of K having 0 eigenvalues (i.e. zero variance through the direction) and do not effect other PC projections

How to get kernel PC score?

- KPC score on the i*th* KPC coordinate for n*th* observation is

$$h(x_n)^T v_i = \sum_{k=1}^{N} \alpha_{ik} h(x_n)^T h(x_k) = \sum_{k=1}^{N} \alpha_{ik} K(x_n, x_k)$$

- We assumed a matrix for transformed variables is column centered. Instead of controlling this matrix (sometimes it is infinite dimensional), we adjust the kernel matrix.

$$\bar{\mathrm{K}} = \mathrm{K} - 1_N \mathrm{K} - \mathrm{K} 1_N + 1_N \mathrm{K} 1_N \text{ whre } 1_N \text{ is } \mathrm{N} \times \mathrm{N} \text{ matrix of } \tfrac{1}{N}.$$

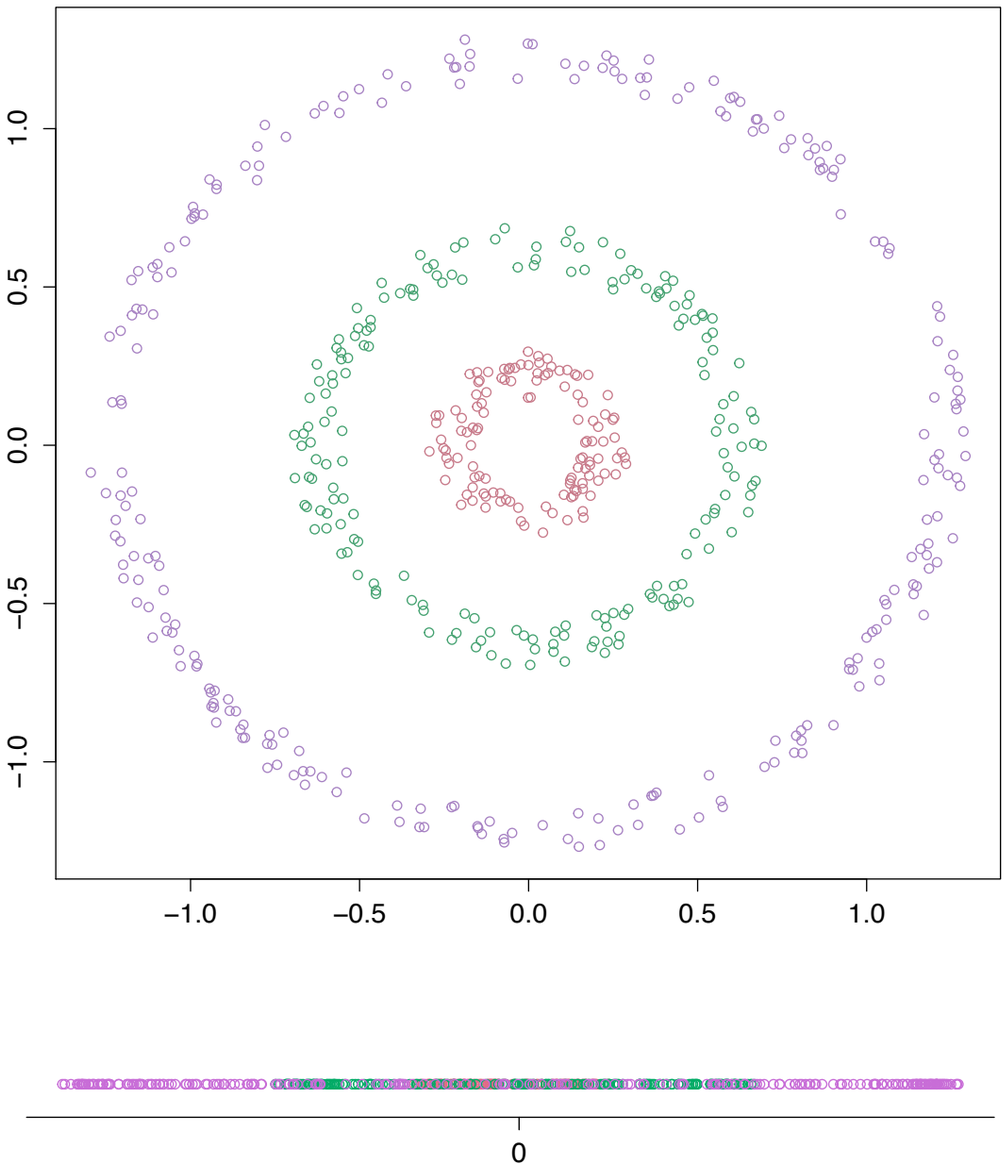
Some kernels

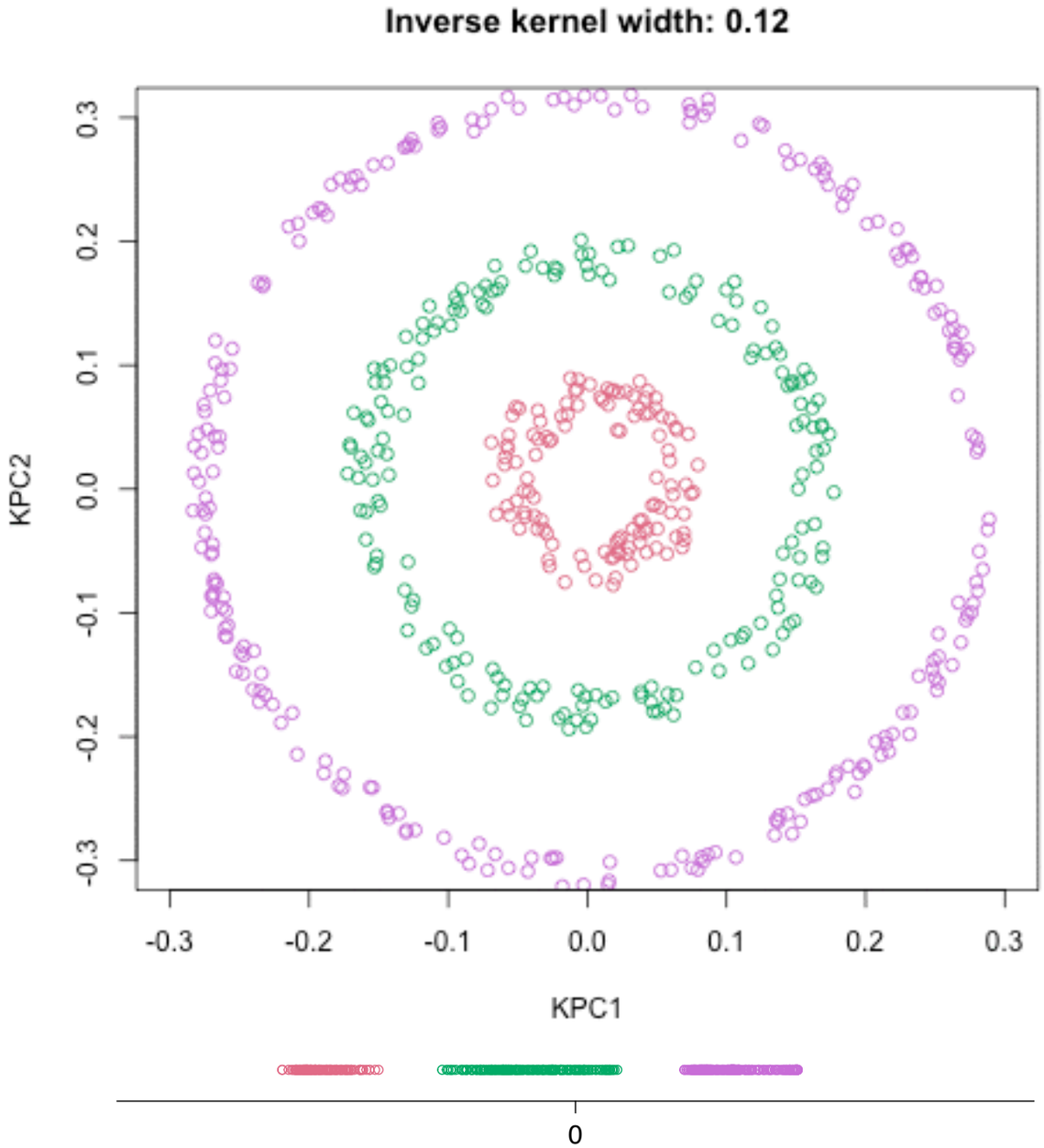
dth-Degree polynomial: $K(x, x') = (1 + \langle x, x' \rangle)^d$,

Radial basis: $K(x, x') = \exp(-\gamma \|x - x'\|^2)$, =Gaussian kernel

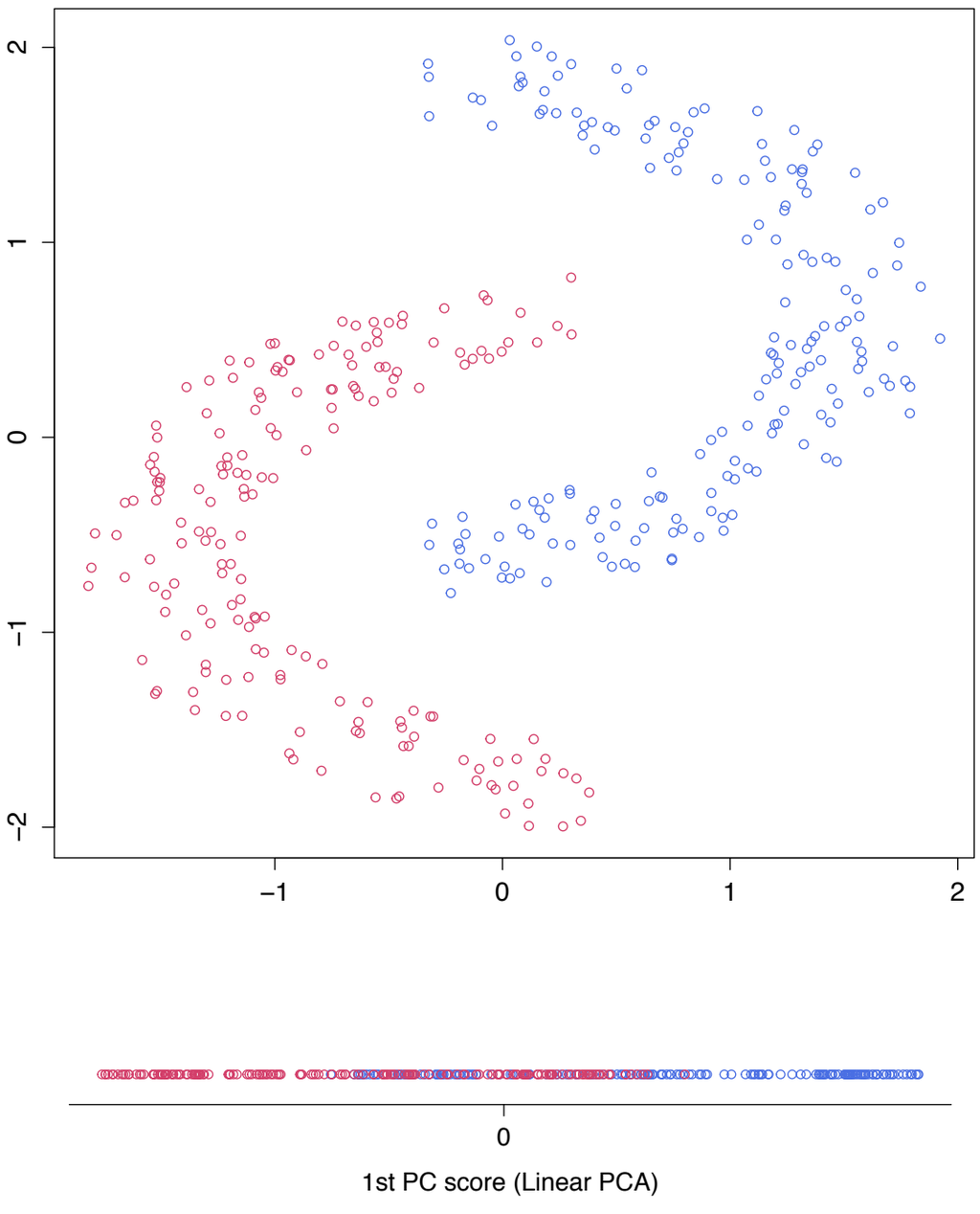
Neural network: $K(x, x') = \tanh(\kappa_1 \langle x, x' \rangle + \kappa_2)$.

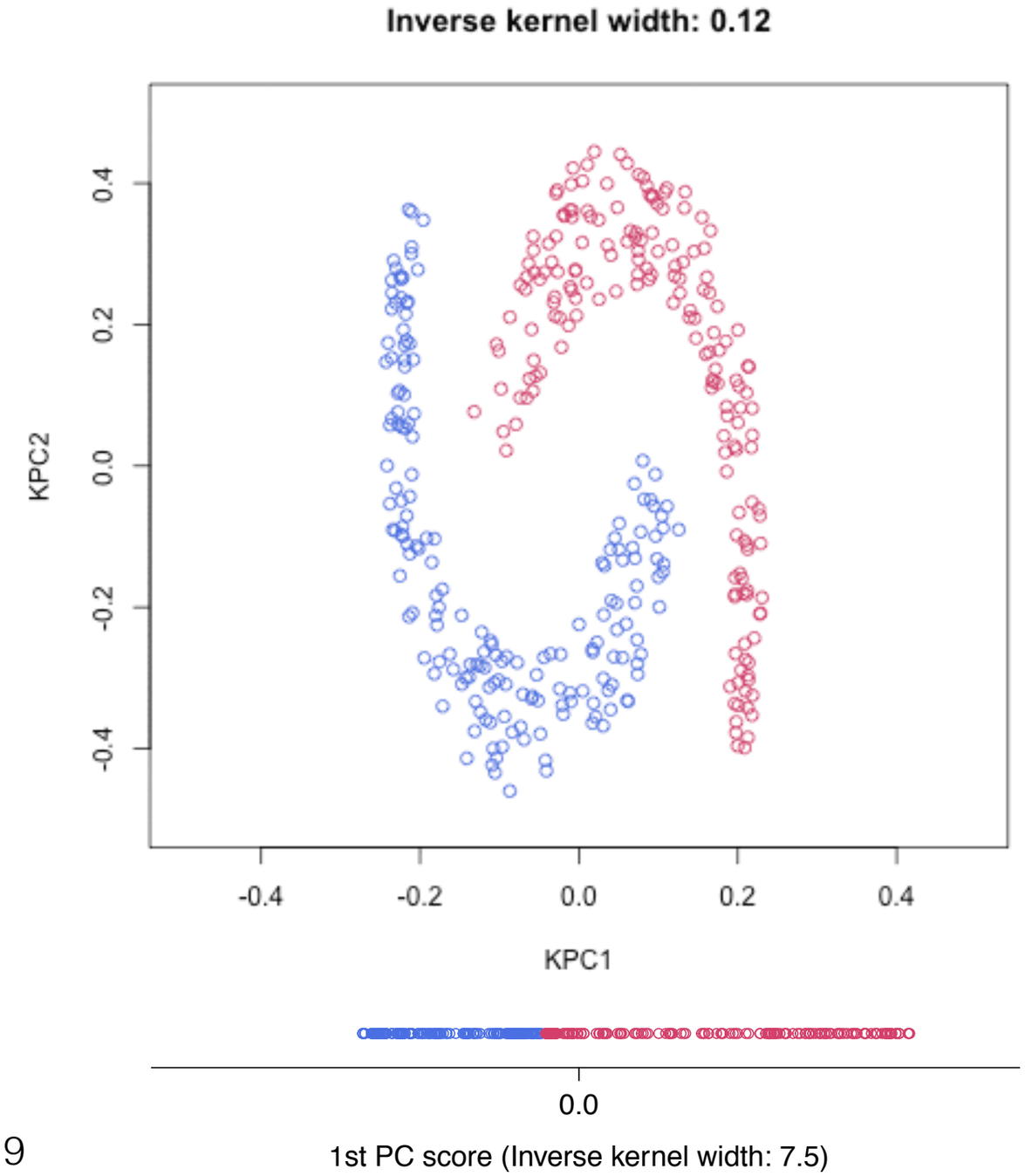
Gaussian kernel PCA: three concentric circles



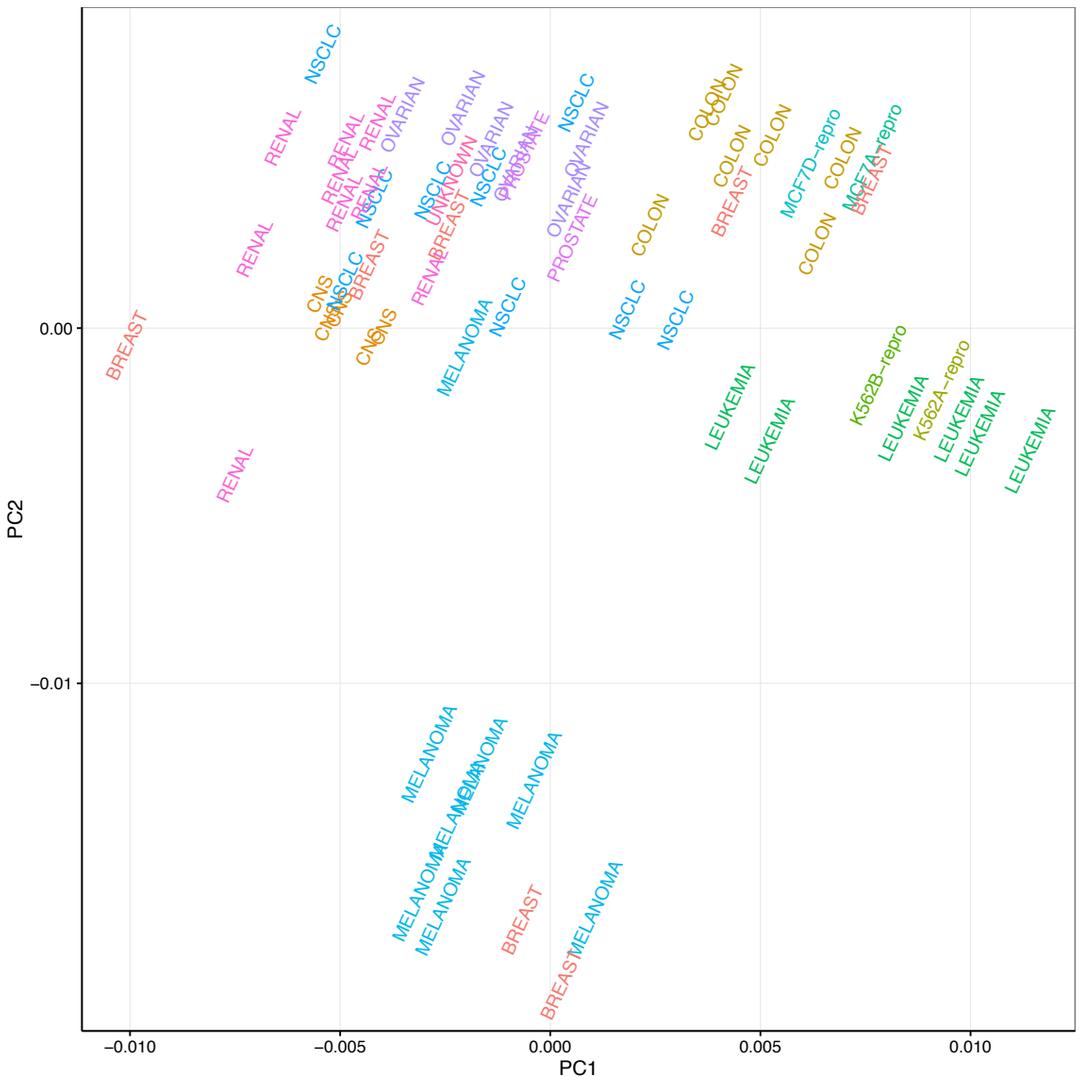


Gaussian kernel PCA: half moons

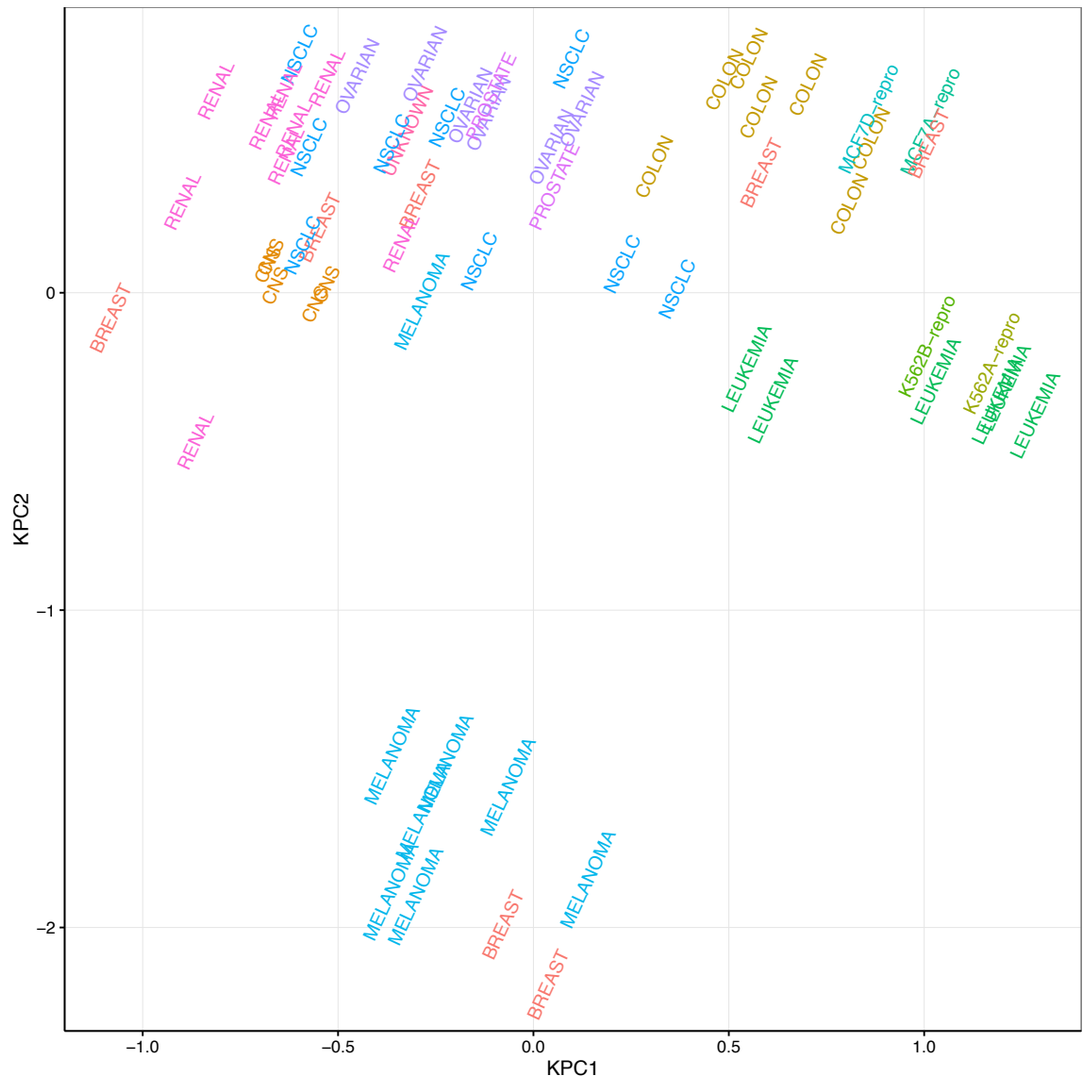




NCI microarray



linear PCA



Gaussian kernel PCA

Summary: KPCA

- Kernel PCA involves finding the eigenvectors of the N × N kernel matrix rather than the p × p sample covariance matrix of linear PCA.

- In linear PCA, # of PC loadings are exactly p. In kernel PCA, # of PC loadings are M (larger than p).

- As in linear PCA, we often retain some reduced number L < M of eigenvectors and then approximate the higher-order association in a data matrix.

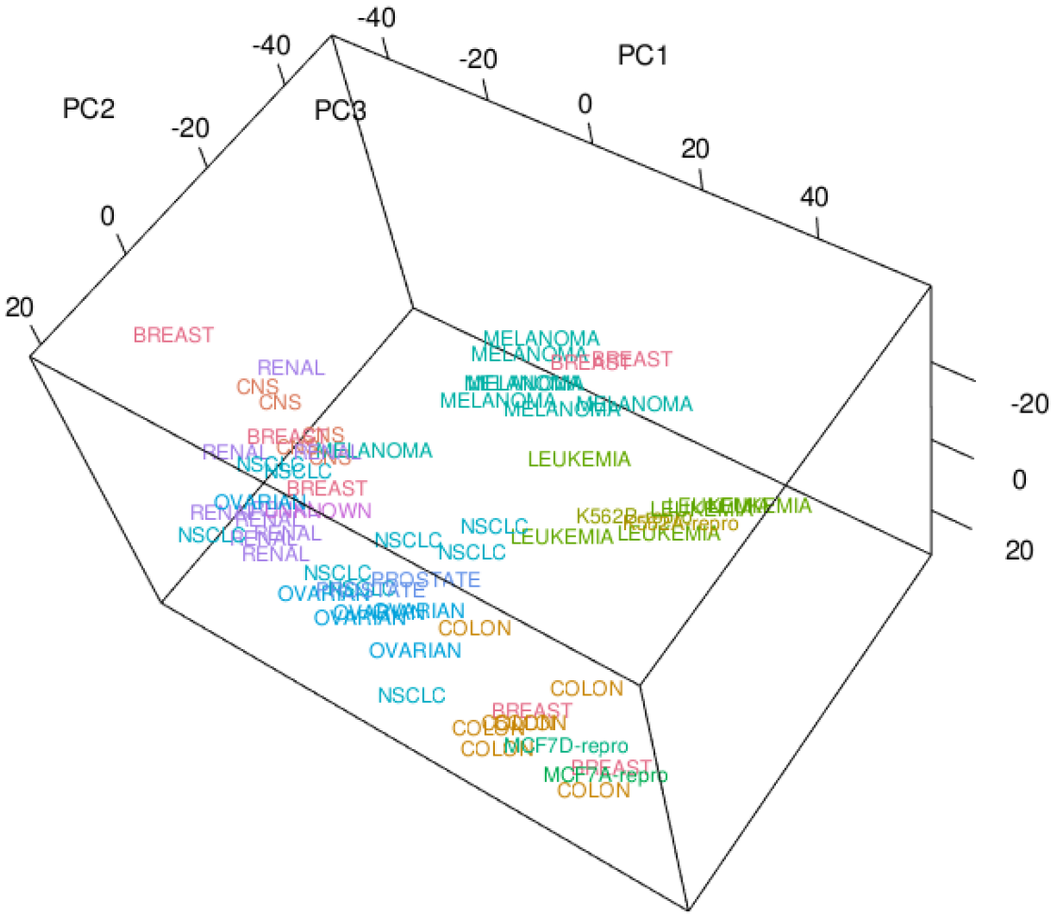
Multidimensional scaling: part II

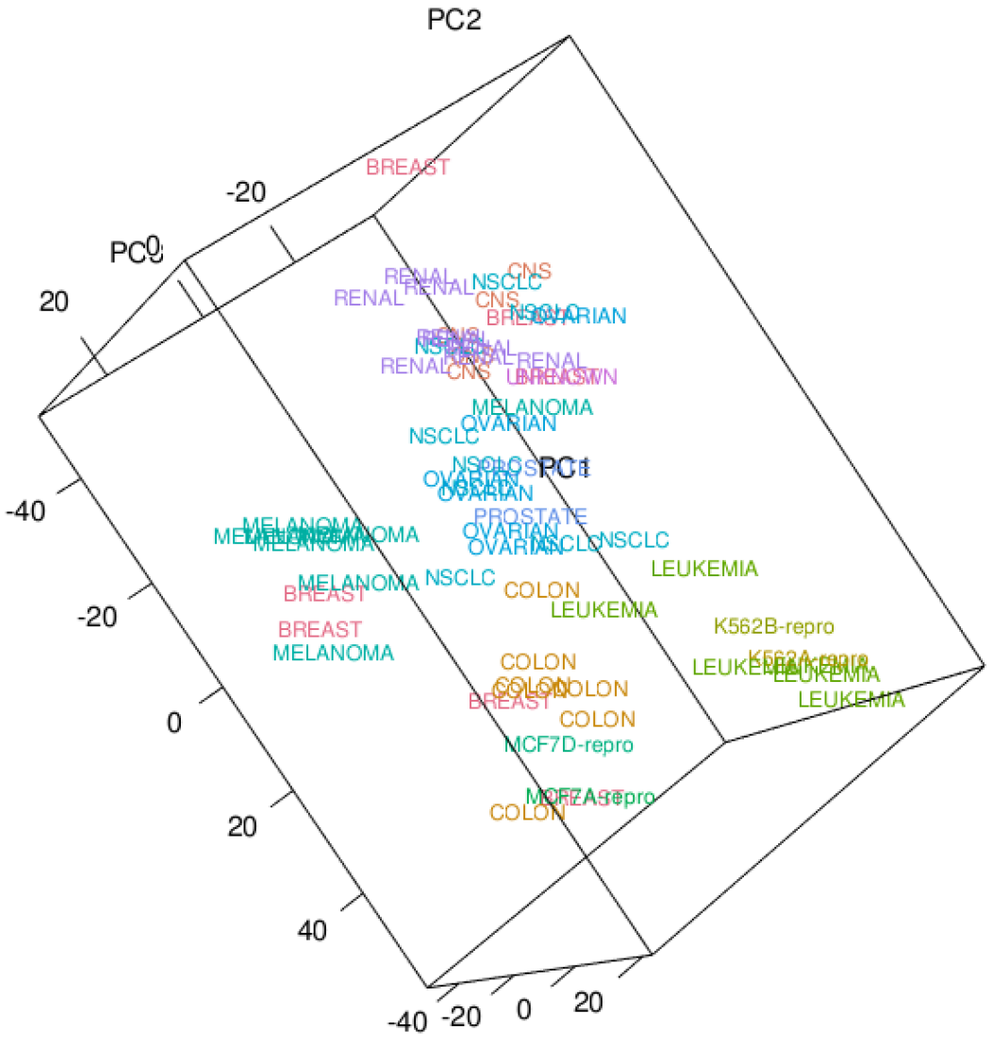
STAT 6312
Department of mathematics, UTA

ESL 14.8, 14.9

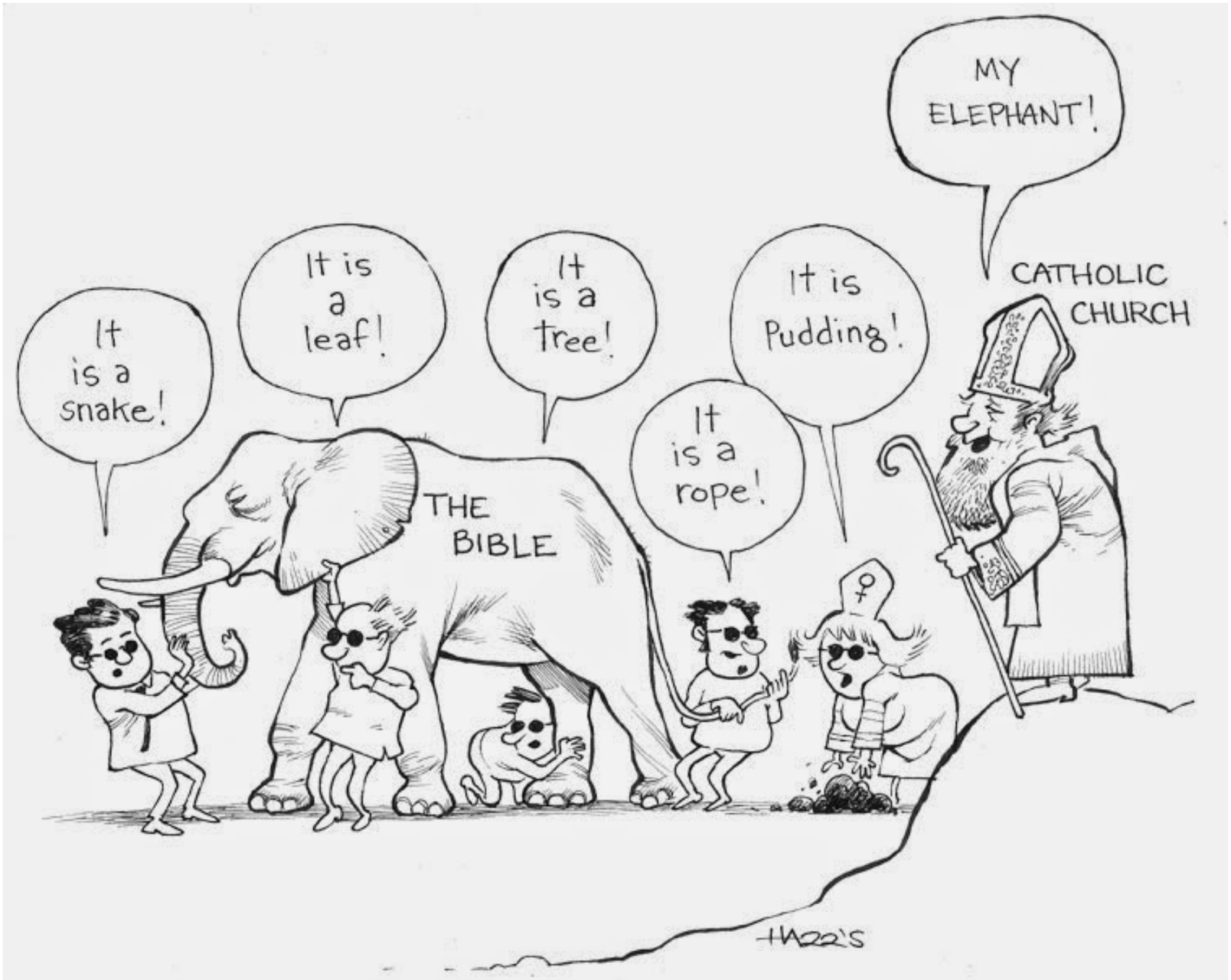
Optional reading: ESL 14.10

How to represent three dimensional data in two dimensions?





Can you see the full elephant?



Multidimensional scaling (MDS)

- Similar to PCA, MDS is a dimensional reduction technique to obtain a low-dimensional representation of data points, but its approach to the problem is somewhat different.

- MDS is based on some dissimilarity measure $d_{ij}=d(x_i,x_j)$
  - e.g. $d_{ij}=\|x_i-x_j\|$: Euclidean distance.

- MDS seeks to find $z_1,\ldots,z_N \in R^k$ to minimize the stress function

$$S_M(z_1, z_2, \ldots, z_N) = \sum_{i \neq i'} (d_{ii'} - ||z_i - z_{i'}||)^2.$$

- In words: to find a lower-dimensional representation of the data that preserves the pairwise distances as well as possible.

Classic MDS

- Here we are given pairwise distance matrix d $\in R^{N\times N}$, instead of data points

$$d = \begin{pmatrix} d(x_1,x_1) & \cdots & d(x_1,x_N) \\ \vdots & \ddots & \vdots \\ d(x_N,x_1) & \cdots & d(x_1,x_N) \end{pmatrix}$$

  - 1. Convert pairwise distance matrix (d) into the inner-product matrix $B = XX^T \in R^{N\times N}$

  - 2. Factorize B to get the first k principal component scores (same as PCA).

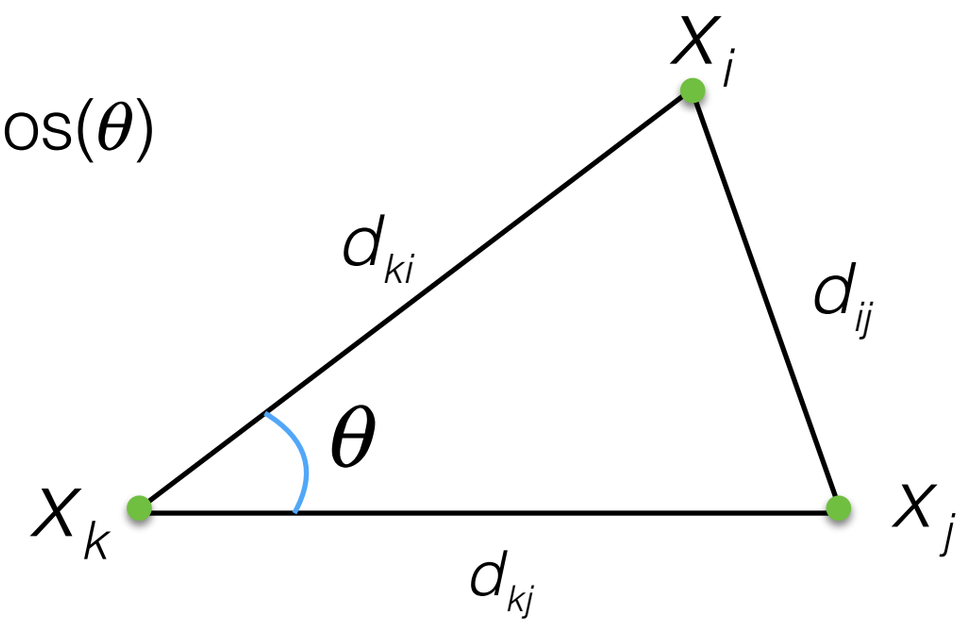
- How to do step 2?: Do spectral decomposition $B = UD^2U^T$, and the first k PC scores are the first k columns of UD.

How do we recover inner-product from distance?

- B can be computed using the distance matrix alone.

$$d_{ij}^2 = d_{ki}^2 + d_{kj}^2 - 2d_{ki}d_{kj}\cos(\theta)$$

$$b_{ij} = d_{ki}d_{kj}\cos(\theta)$$



$$b_{ij} = \frac{1}{2}(d_{ki}^2 + d_{kj}^2 - d_{ij}^2)$$

Recover inner-products

- The following procedure can be used to recover the inner-products $B = XX^T$ from the distance matrix d
  - 1. Compute $A_{ij} = -0.5d^2_{ij}$ to form the matrix $A \in R^{N \times N}$
  - 2. Double center A—i.e., center both the columns and rows of A—to recover the matrix $B \in R^{N \times N}$.
  - Note that this is the same as $B = (I - M)A(I - M)$ where M is N $\times$ N matrix of 1/N.
- Does it matter whether we first center the columns or the rows?

- If the d_{ij} is Euclidean distances between the rows of a centered matrix $X \in R^{N \times p}$, we get back the first k principal component scores exactly. Importantly, MDS can be applied to any d_{ij}, not just Euclidean distances

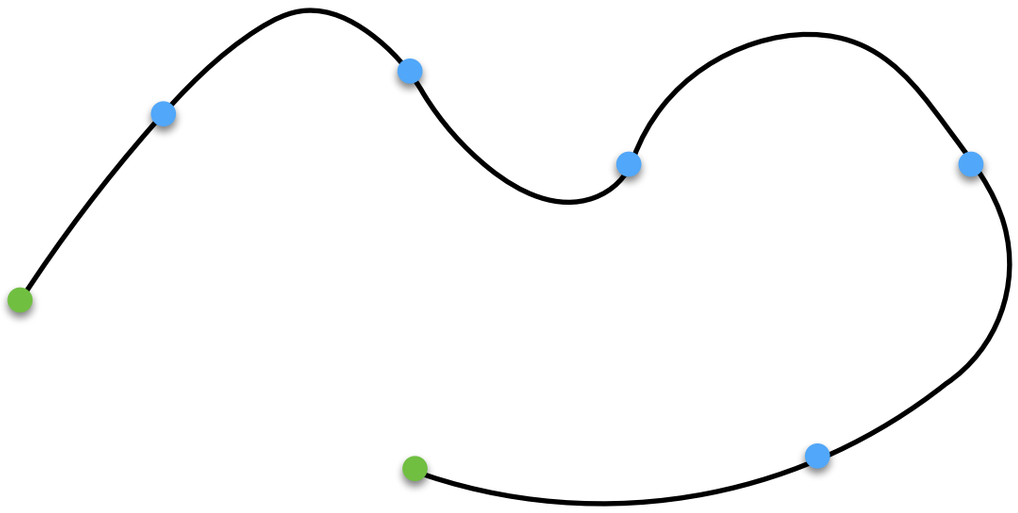
- There is a class of methods which construct a fancier metric d_{ij} between high-dimensional points, and then they feed these d_{ij} through MDS to get a low-dimensional representation $z_1,\ldots,z_n \in R^k$.

- In this case, we don't just get principal component scores, and our low-dimensional representation can end up being a nonlinear function of the data

- Why would we want to use non-Euclidean distances?

Nonlinear manifolds

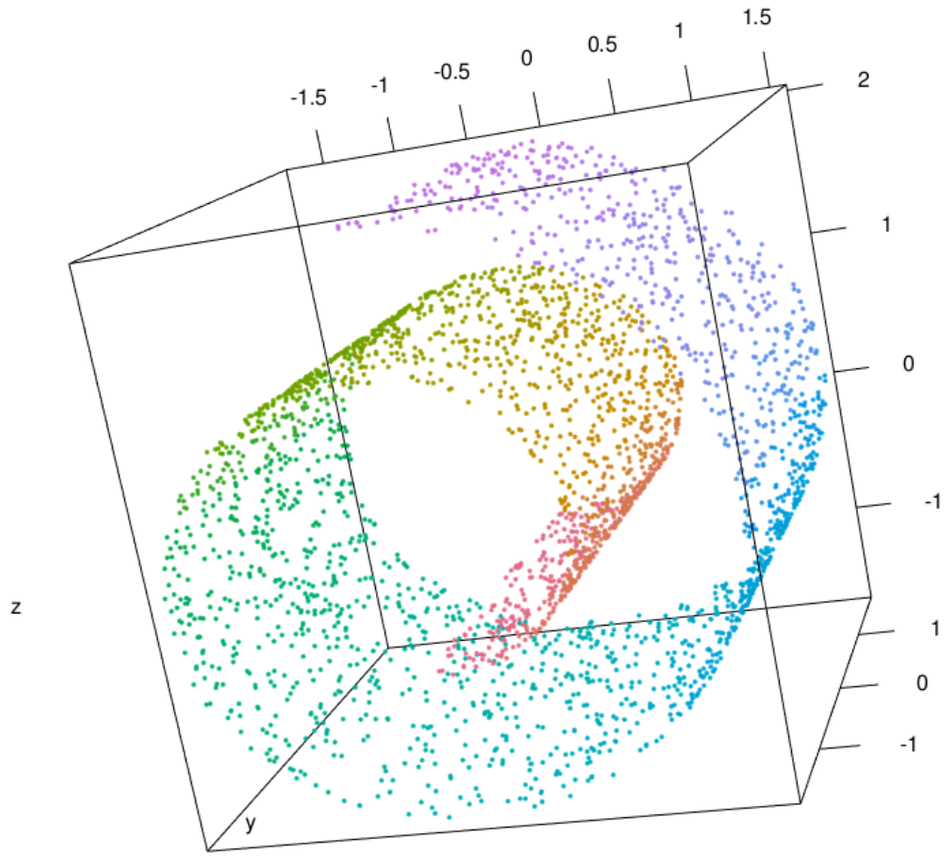
PCA and MDS see the Euclidean distance

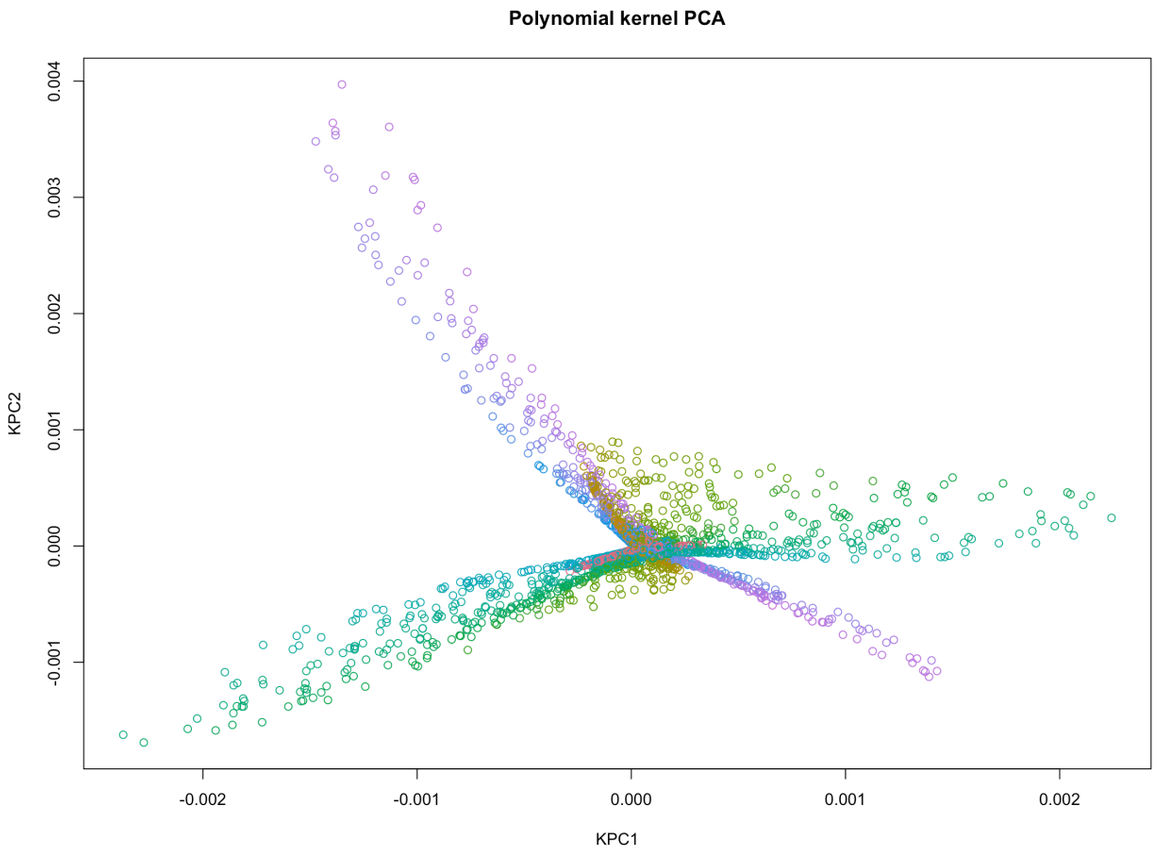


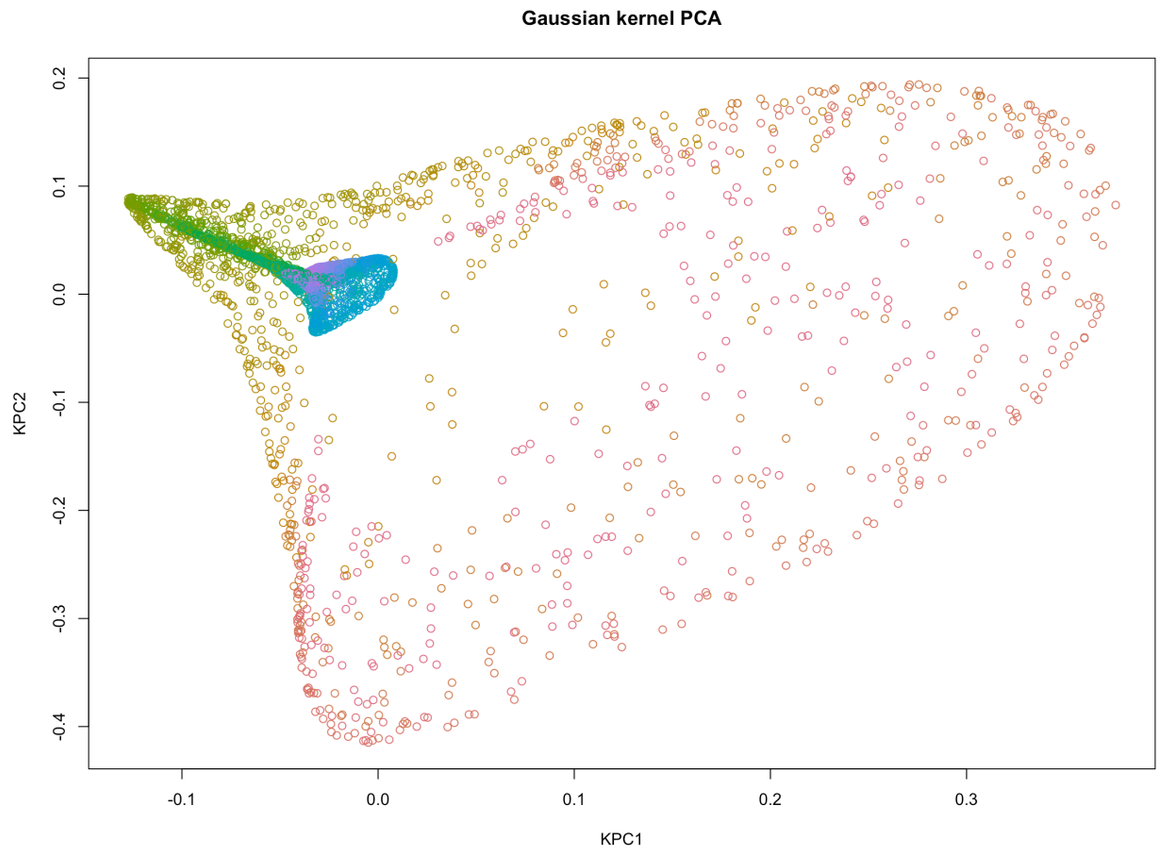
What is important is the geodesic distance (graph distance)

Unroll the manifold

Swiss roll


1.5
1
0.5
0
-0.5
-1
-1.5
2
1
0
-1
1
0
-1
z
y
x


Polynomial kernel PCA
KPC2
KPC1
0.004
0.003
0.002
0.001
0.000
-0.001
-0.002
-0.001
0.000
0.001
0.002

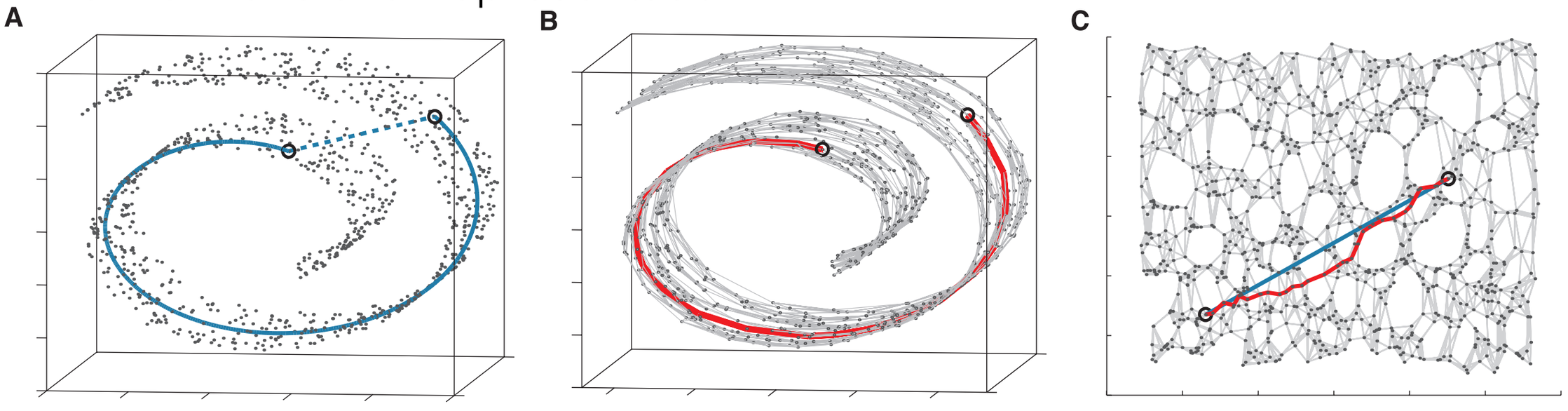

Gaussian kernel PCA
KPC2
KPC1
0.2
0.1
0.0
-0.1
-0.2
-0.3
-0.4
-0.1
0.0
0.1
0.2
0.3

Isometric feature mapping (Isomap)

- Isomap learns structure in a more general setting to define distances.

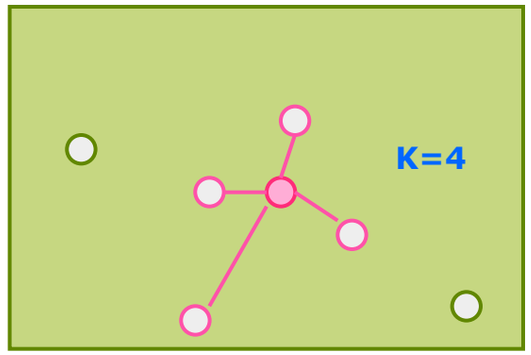
- The basic idea is to construct a graph G = (V, E), i.e., construct edges E between vertices V = {1,...N}, based on the structure between $x_1,\ldots,x_N \in R^p$.

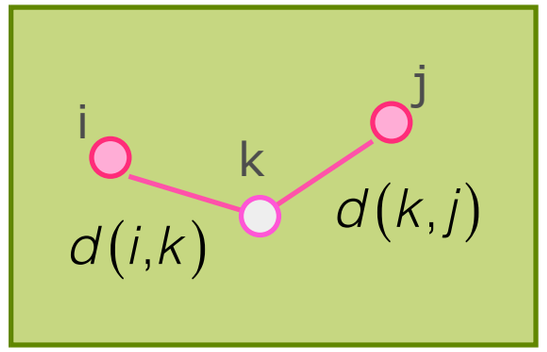
- We define a graph distance between i and j, and use MDS for our low-dimensional representation

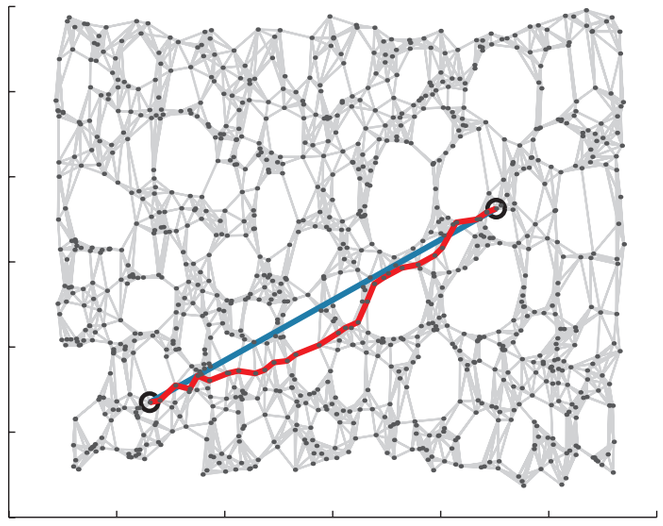


Isomap - Algorithm

- Determine K nearest neighbors
- Construct a neighborhood graph.
  - Each point is connected to the other if it is a K nearest neighbor.
  - Edge length equals the Euclidean distance (For neighboring points Euclidean distance is a good approximation to the geodesic distance).
- Compute the shortest paths in a graph between two nodes
  - For faraway points, estimate the distance by a series of short hops between neighboring points.
- Construct a lower dimensional embedding using classical MDS.





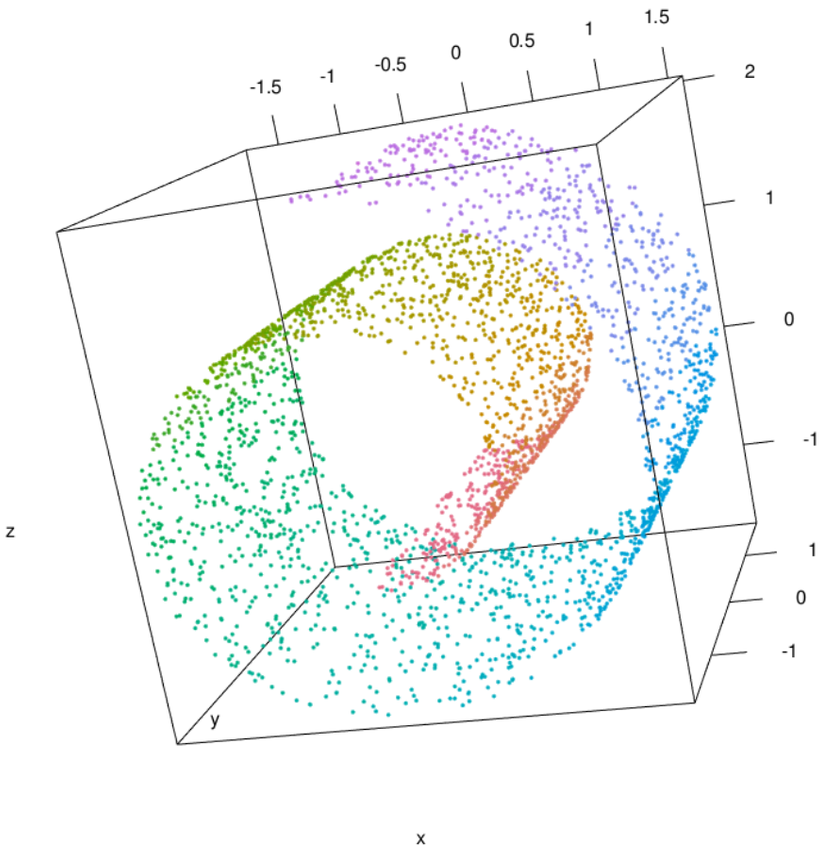


Some details

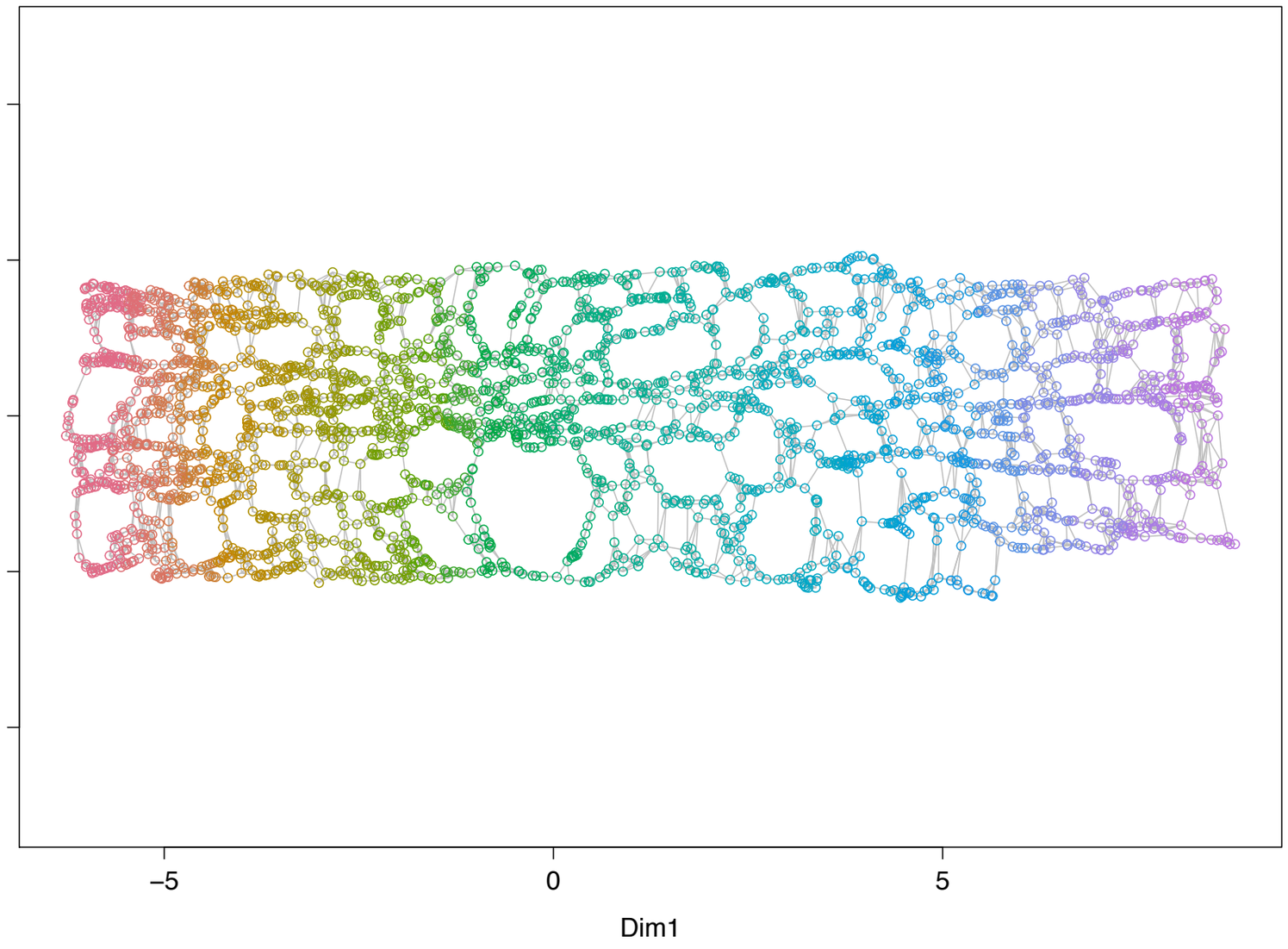
- Constructing the graph: for each pair i,j, we connect i,j with an edge if either:
  - x_i is one of x_j's K nearest neighbors, or
  - x_j is one of x_i's K nearest neighbors
- The weight of this edge e={i,j} is then $w_e = \|x_i - x_j\|_2$
- Defining graph distances (geodesic distance): now that we have built a graph, i.e., we have built an edge set E, we define the graph distance between x_i and x_j to be the shortest path in our graph from i to j

$$d_{ij}^{\text{Isomap}} = \min_{\text{path } P \subseteq E \text{ from } i \text{ to } j} \sum_{e \in P} w_e$$

- This can be computed using Floyd's or Dijkstra's algorithm

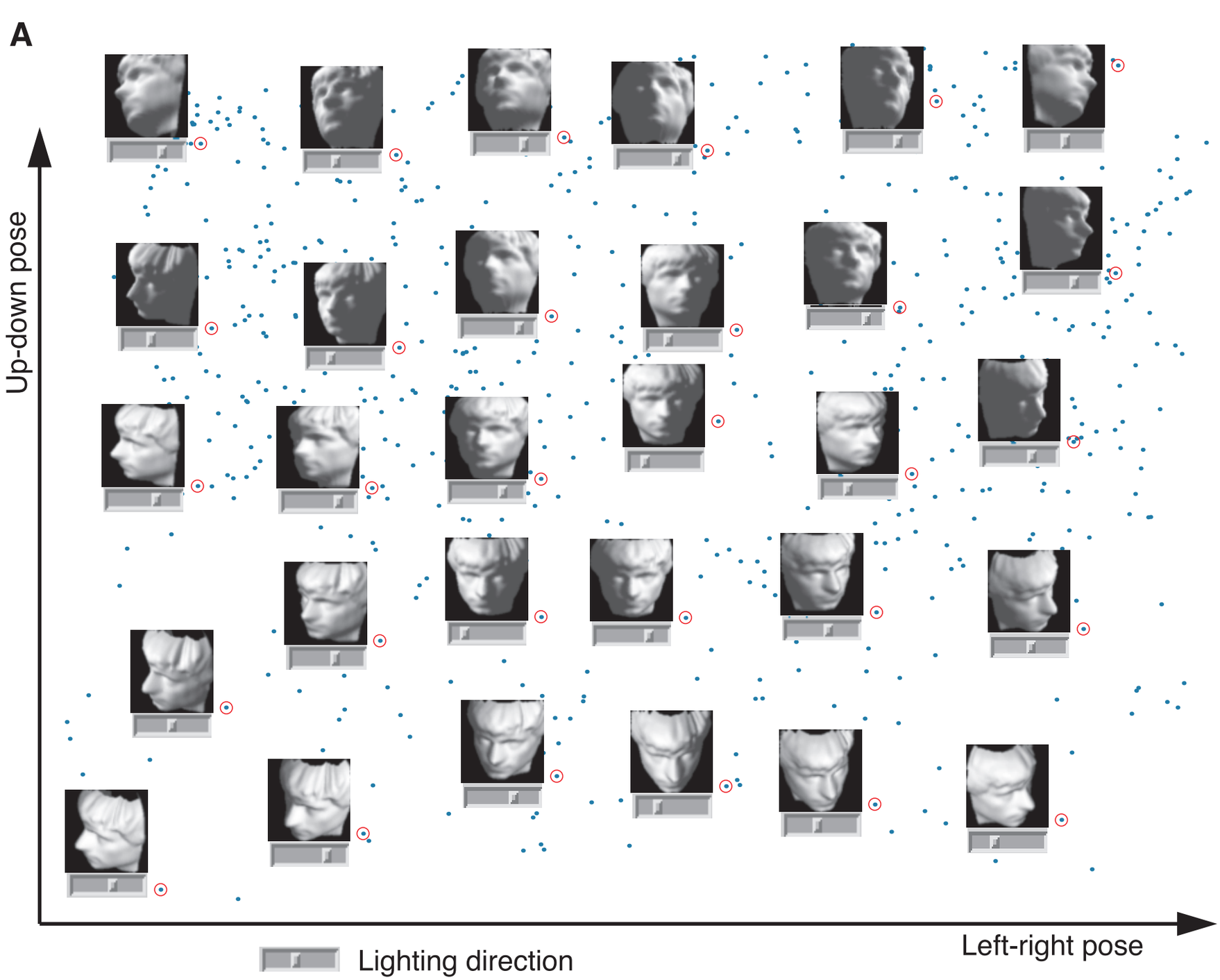


Isomap with k=5 nearest neighbors



Summary: Isomap

- Do MDS on the geodesic distance matrix
- Global approach: Use the geodesic distances between all pairs
- Preserve the neighborhoods and their geometric relation
- On a low dimensional embedding
  - Nearby points should be nearby.
  - Faraway points should be faraway.



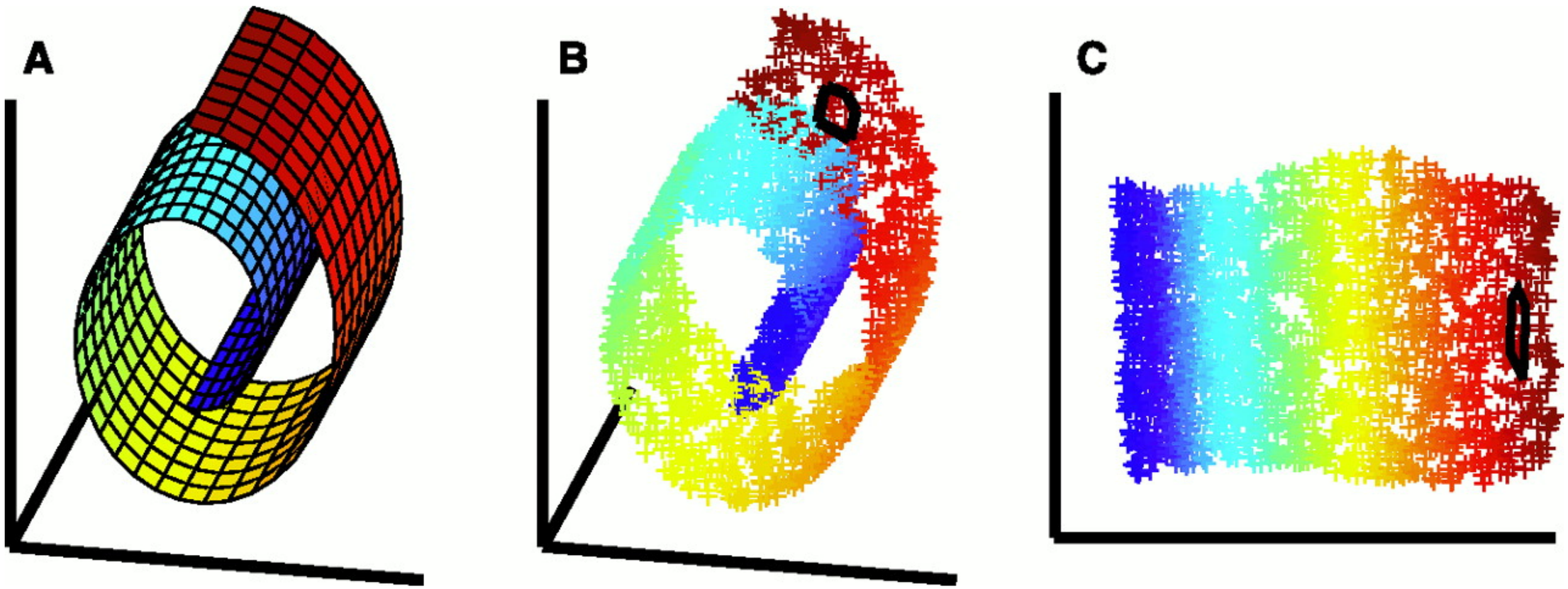
- The input consists of a sequence of 4096-dimensional vectors, representing the brightness values of 64 pixel by 64 pixel images of a face rendered with different poses and lighting directions.

Locally Linear Embedding (LLE)

- LLE is a similar method in spirit but its details are very different. It doesn't use MDS.
- The basic idea has two steps:
  - 1. Learn a bunch of local approximations to the structure between $x_1,\ldots,x_N \in R^p$
  - 2. Learn a low-dimensional representation $z_1,\ldots,z_N \in R^k$ that best matches these local approximations
- What is meant by such local approximations? We simply try to predict each x_i by a linear function of nearby points x_j (hence the name local linear embedding)

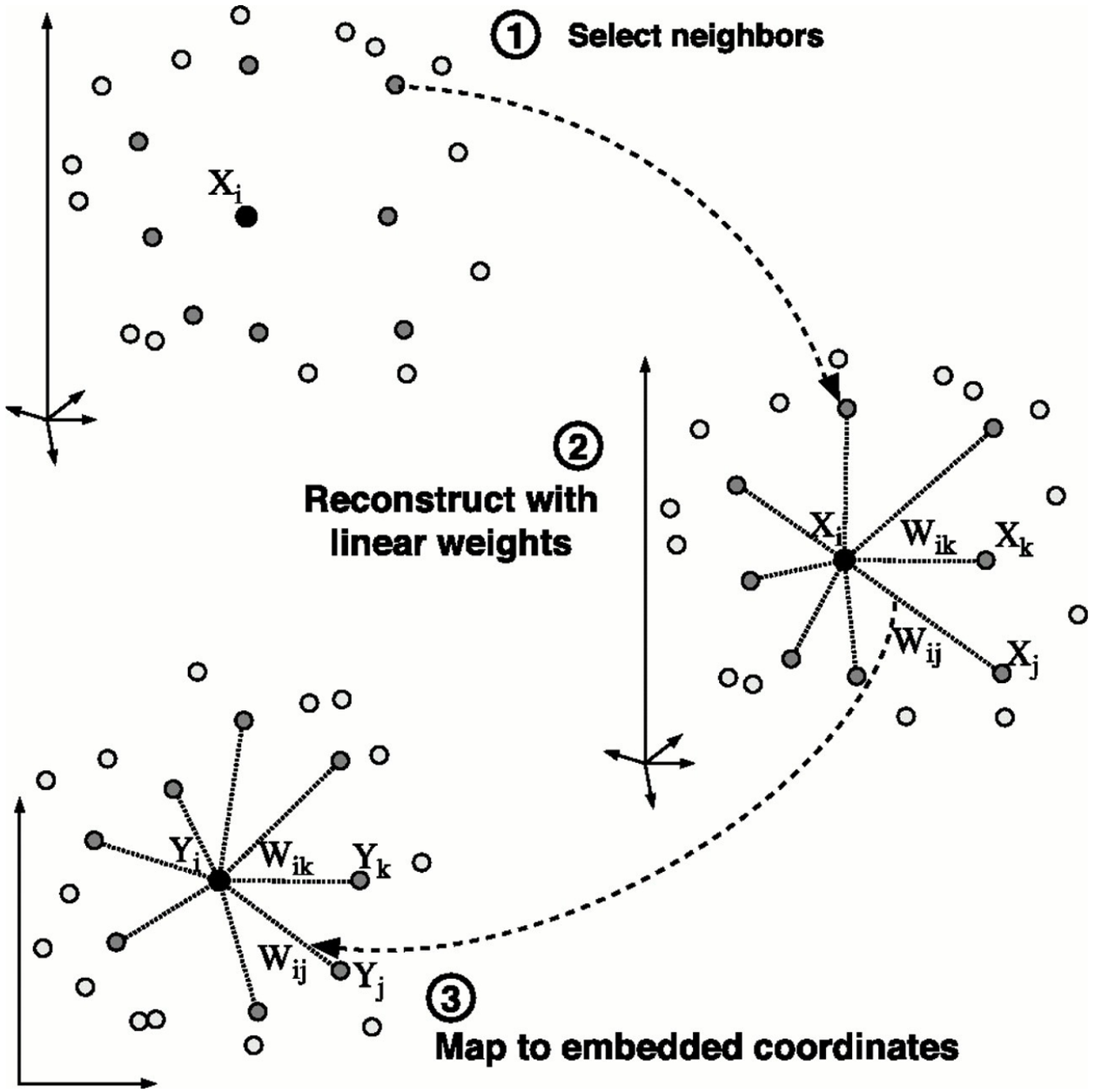
Swiss roll again…

- LLE finds a mapping to preserve local linear relationships between neighbors



Rowers and Saul, 2000. Science

Fit local linear relationships



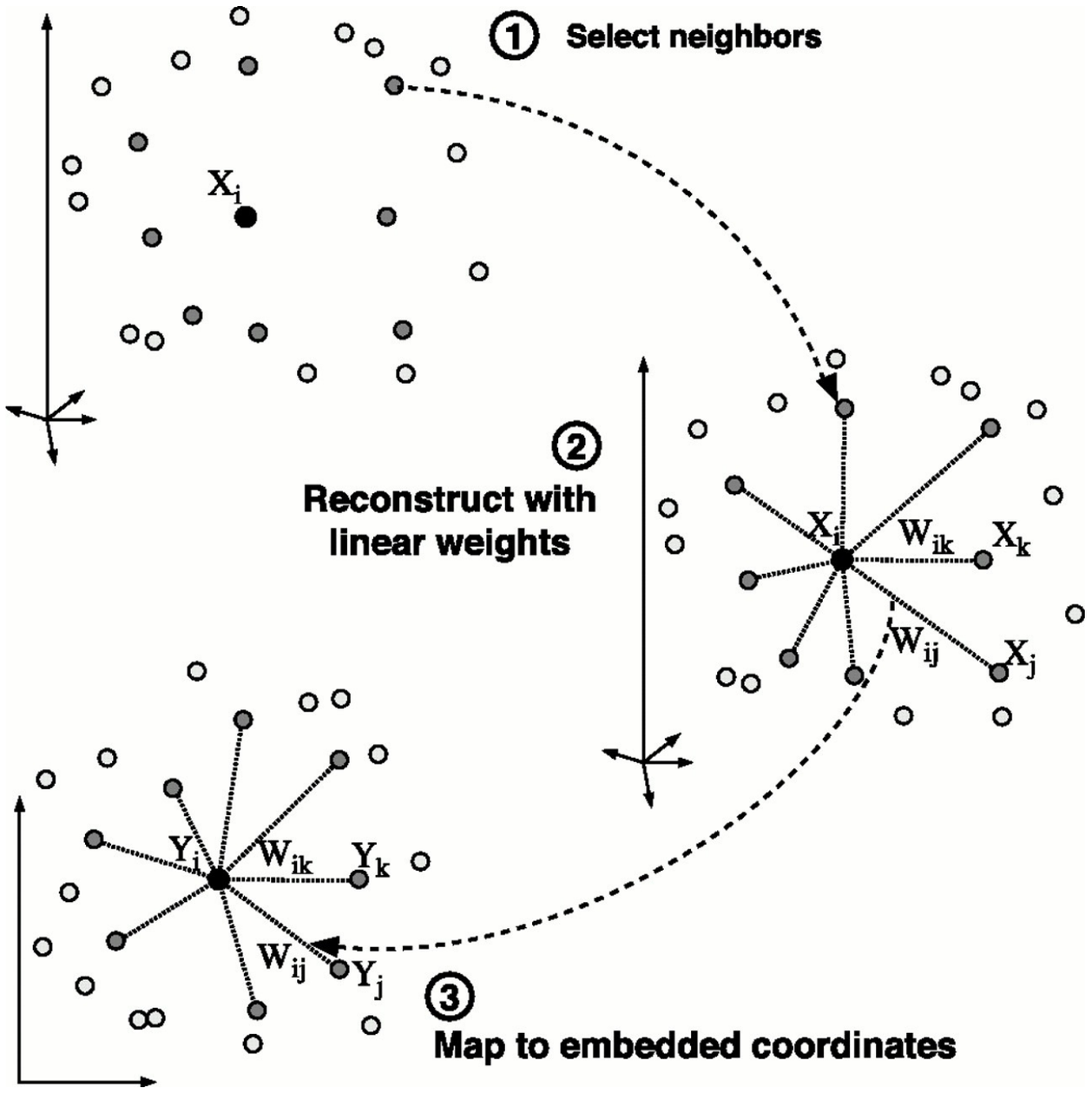
- We expect each data point and its neighbors to lie on or close to a locally linear patch of the manifold.

- Each point can be written as a linear combination of its neighbors.

- The weights chosen to minimize the reconstruction Error

$$\min_W \sum_i \| x_i - \sum_{j=1}^{k} w_{ij} x_j \|^2$$

Fit local linear relationships



- Compute y_i (low-dimensional representation of x_i in embedded coordinates) best reconstructed by the weights w_{ij}, minimizing

- $$\sum_i \| y_i - \sum_{j=1}^{K} w_{ij} y_j \|^2$$

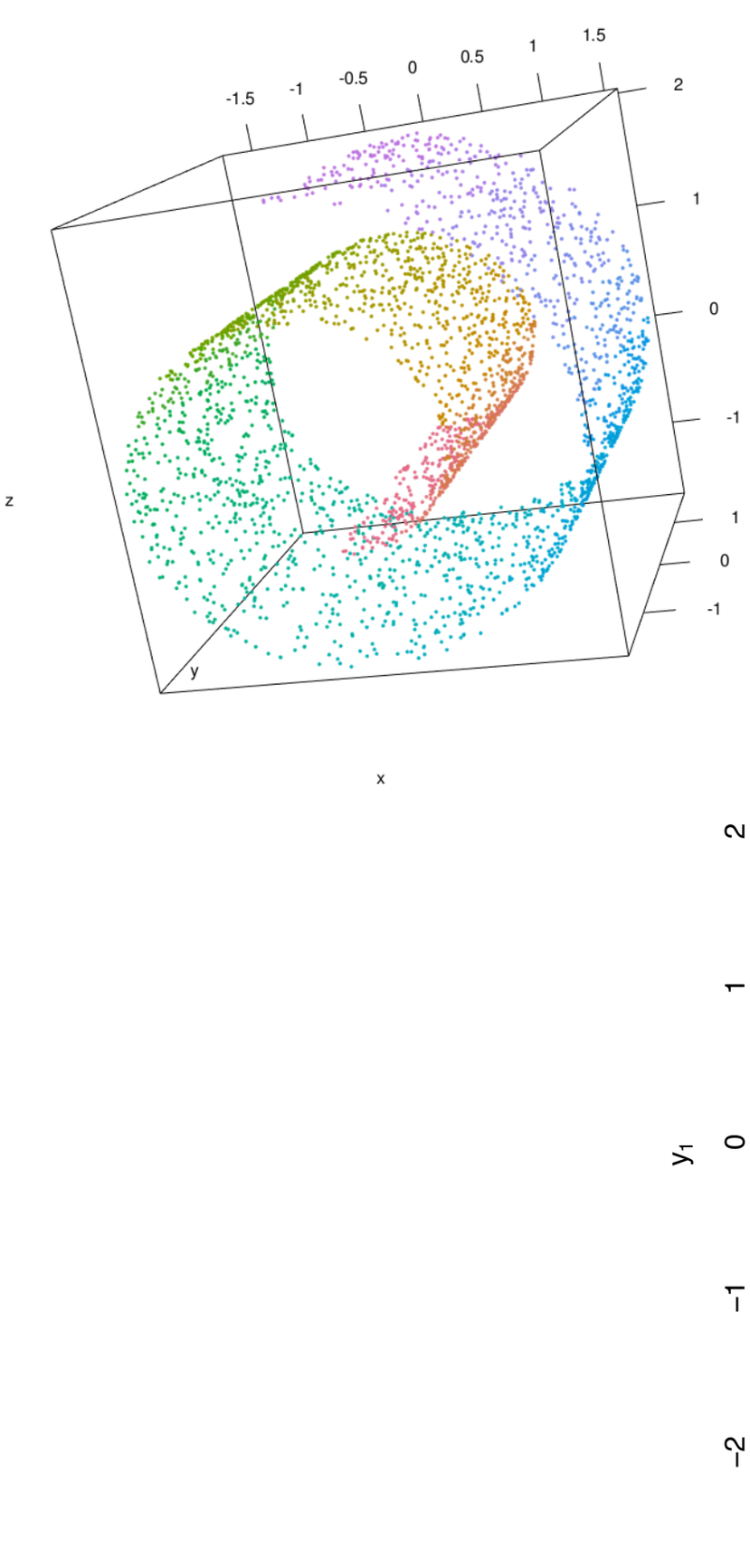
Some properties

- The weights that minimize the reconstruction errors are invariant to rotation, rescaling and translation of the data points.

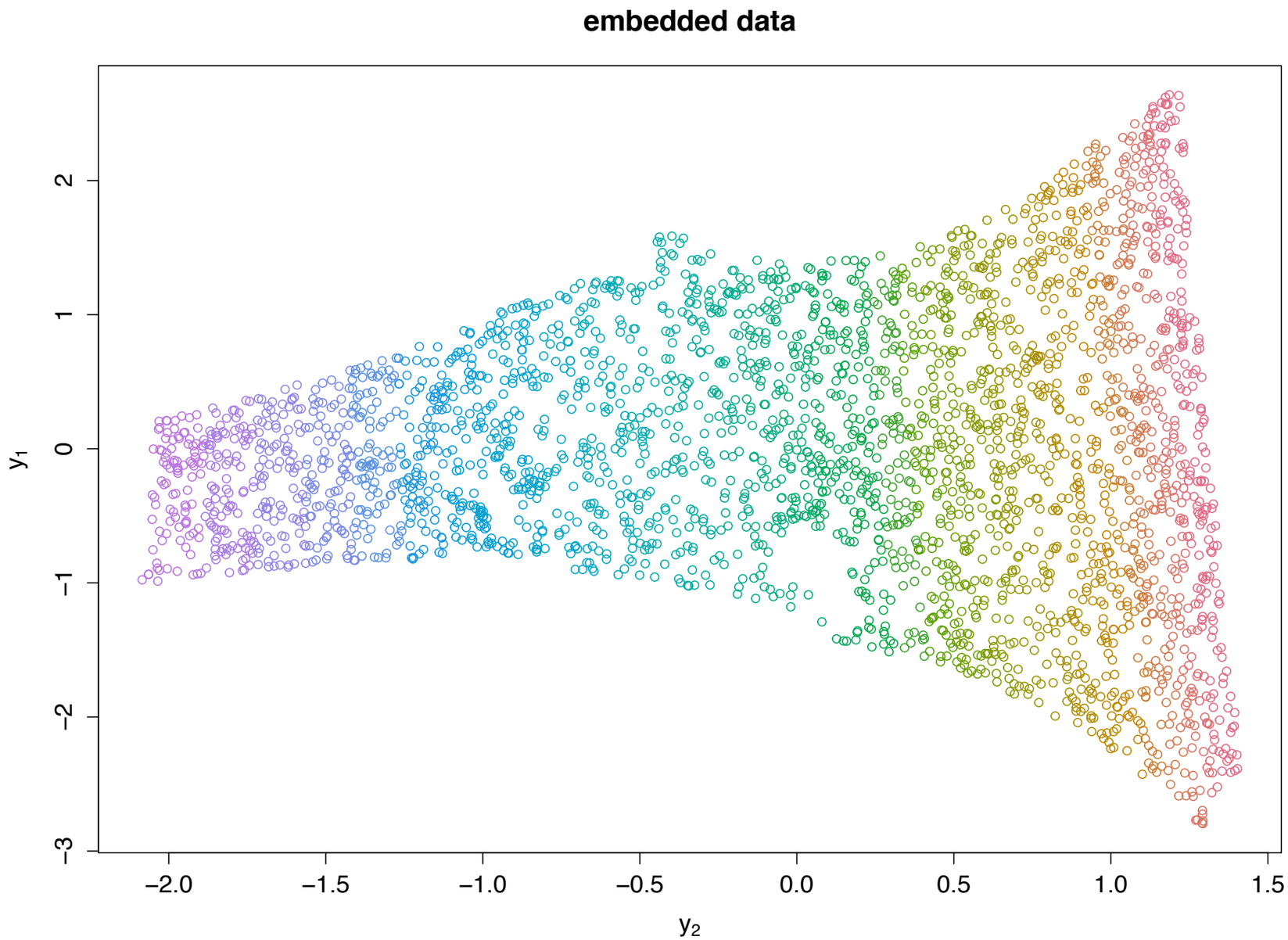
- Invariance to translation is enforced by adding the constraint that the weights sum to one.

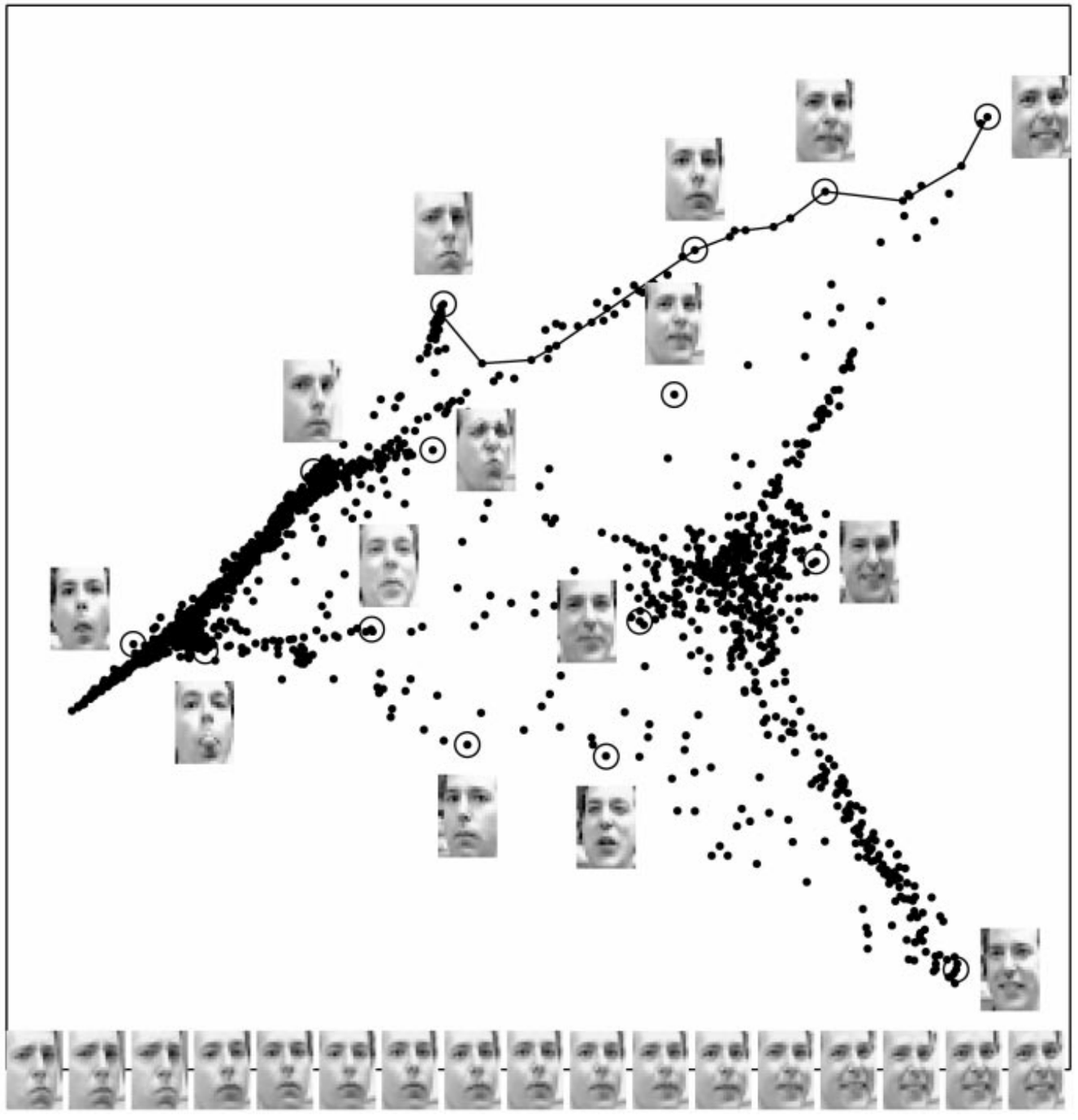
- The same weights that reconstruct the data points in p dimensions should reconstruct it in the manifold in k dimensions.

- The weights characterize the intrinsic geometric properties of each neighborhood.



LLE with k=9 nearest neighbor



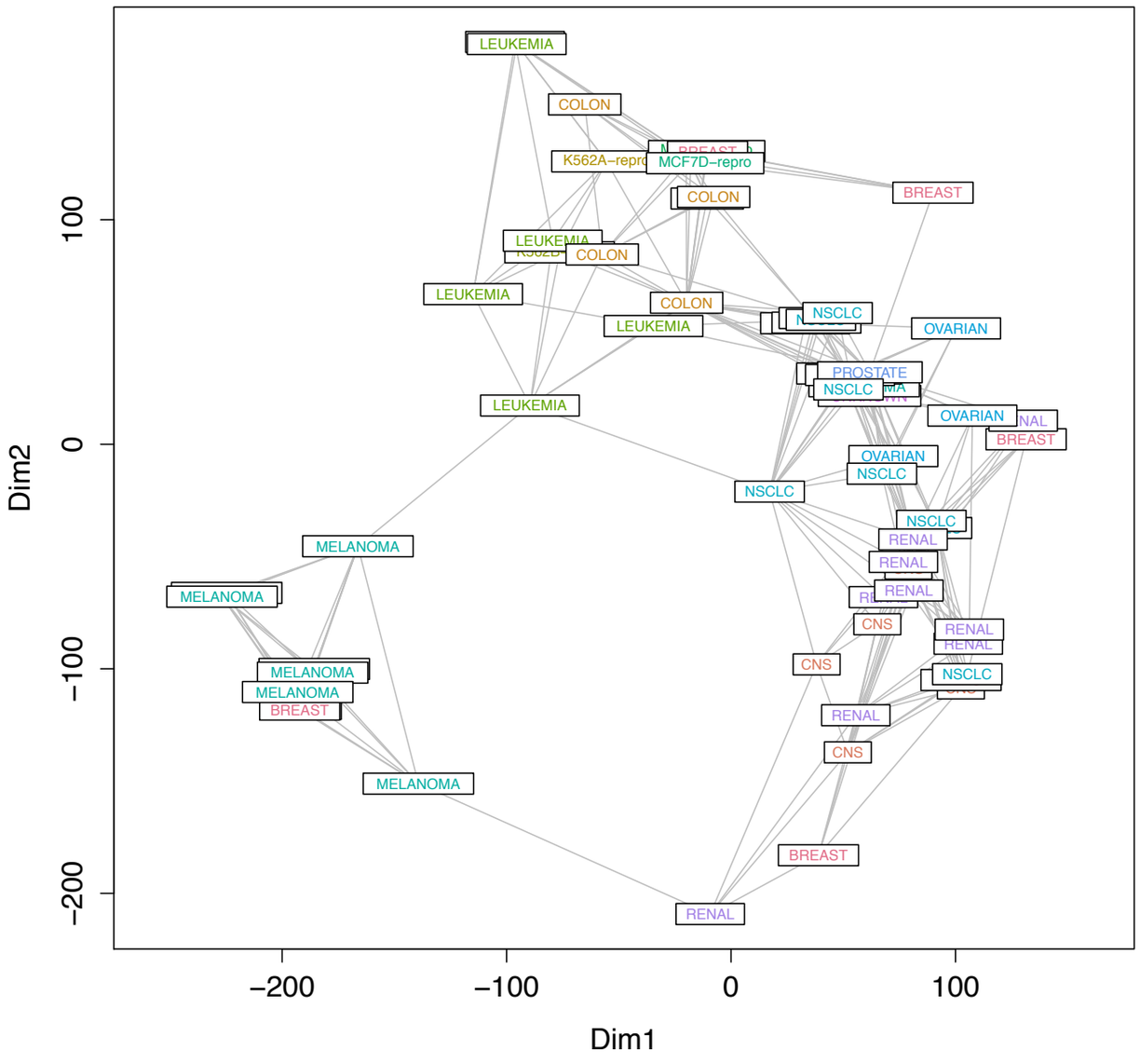


- Images of faces mapped into the embedding space described by the first two coordinates of LLE. Representative faces are shown next to circled points in different parts of the space.

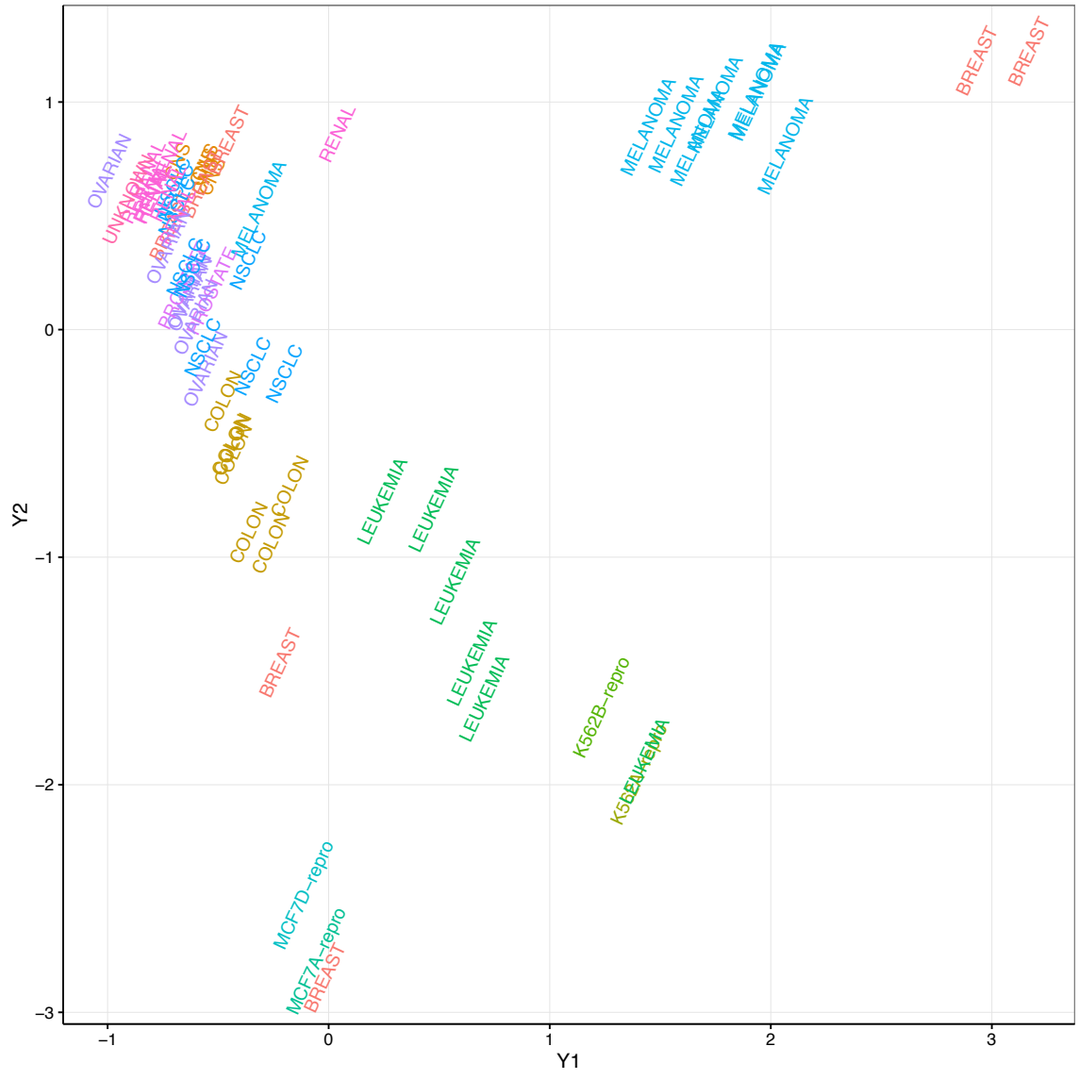
- The bottom images correspond to points along the top-right path (linked by solid line), illustrating one particular mode of variability in pose and expression

NCI microarray

Isomap K=5



LLE K=17



Summary: LLE

- Model local neighborhoods as linear a patches and then embed in a lower dimensional manifold.
- Preserve the neighborhoods and their geometric relation
- Computationally efficient (sparse matrices)
- No local minima, one free parameter (k)
- Local approach—Nearby points are embedded nearby. It can distort global structure