Nonlinear dimensionality reduction

Victor Kitov

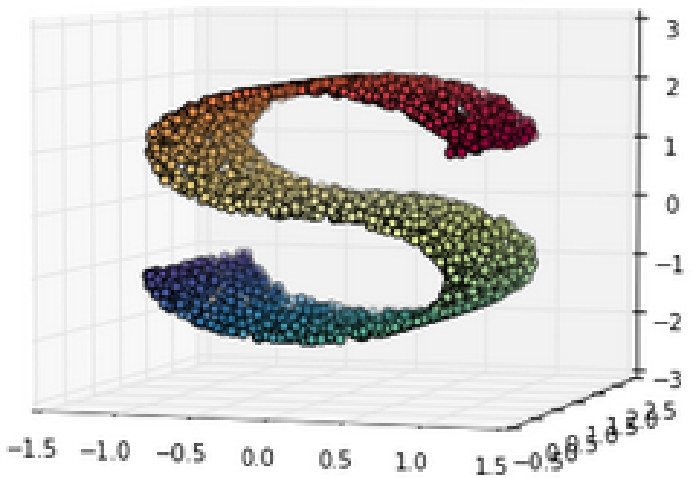
v.v.kitov@yandex.ru

Advantages of dimensionality reduction

- Reduce operational time and storage costs.
- Remove multi-collinearity in features.
- Visualize in 2D or 3D.

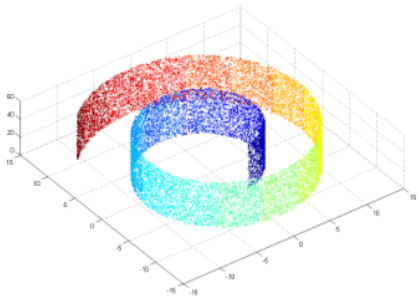
Non-linear dimensionality reduction

- Based on assumption that original data $x \in \mathbb{R}^D$ is distributed compactly on non-linear surface with dimensionality $d < D$.
- Let $y \in \mathbb{R}^d$ denote the coordinates of x on the surface.
- d is usually unknown.
- Sample dataset:

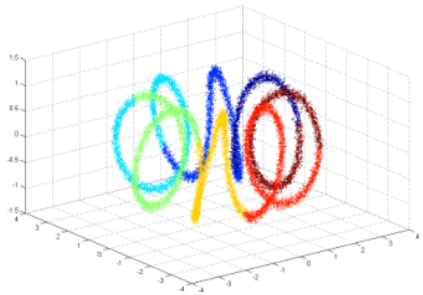


- Linear dimensionality reduction techniques will fail here.

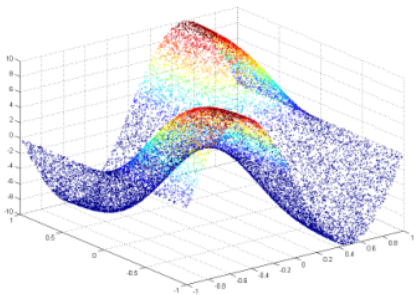
Typical datasets for dimensionality reduction evaluation



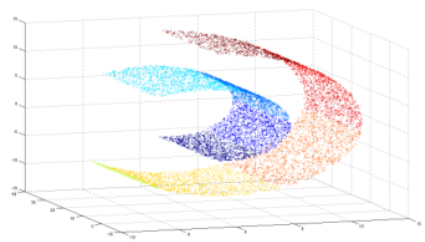
(a) Swiss roll dataset.



(b) Helix dataset.



(c) Twinpeaks dataset.



(d) Broken Swiss roll dataset.

Comment: true datasets have much more dimensions, more complex structure, errors, outliers, etc.

Categorization

Non-linear approaches of dimensionality reduction:

- preserving global properties
  - kernel PCA, autoencoders, MDS, ISOMAP, diffusion maps, MVU
- preserving local properties
  - LLE, LTSA
- global alignment of local linear models (not considered here)

Table of Contents

Multi-dimensional scaling

Multi-dimensional scaling

Map $x \to y$ preserving distances as much as possible.

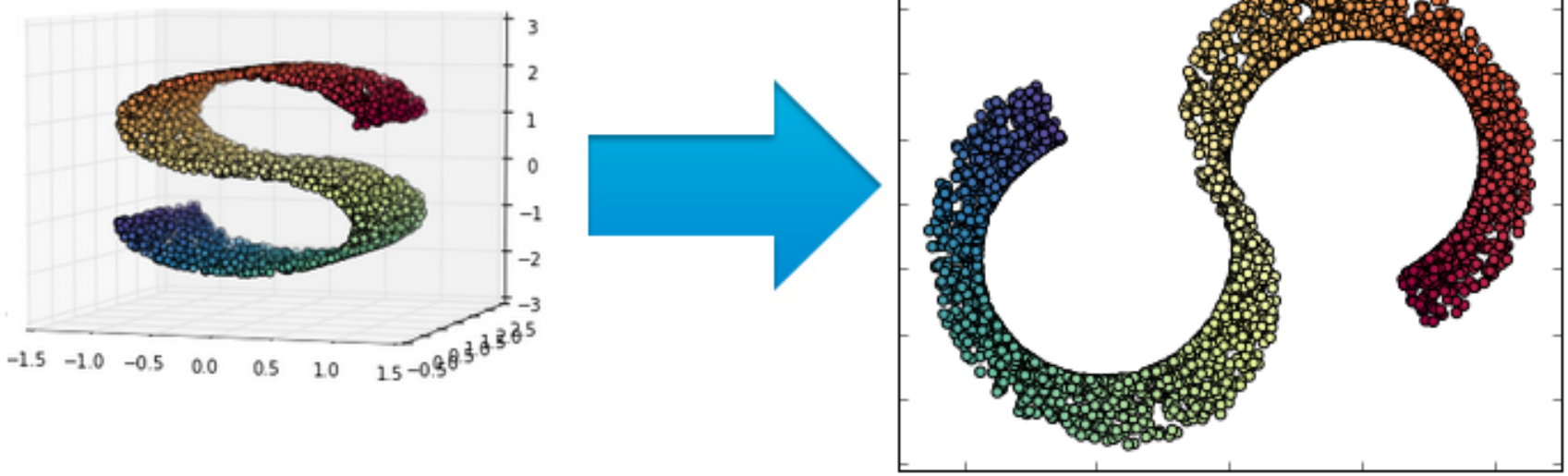
- Approaches:
  - absolute difference

$$\sum_{i,j} \left(\|x_i - x_j\| - \|y_i - y_j\| \right)^2 \to \min_Y$$

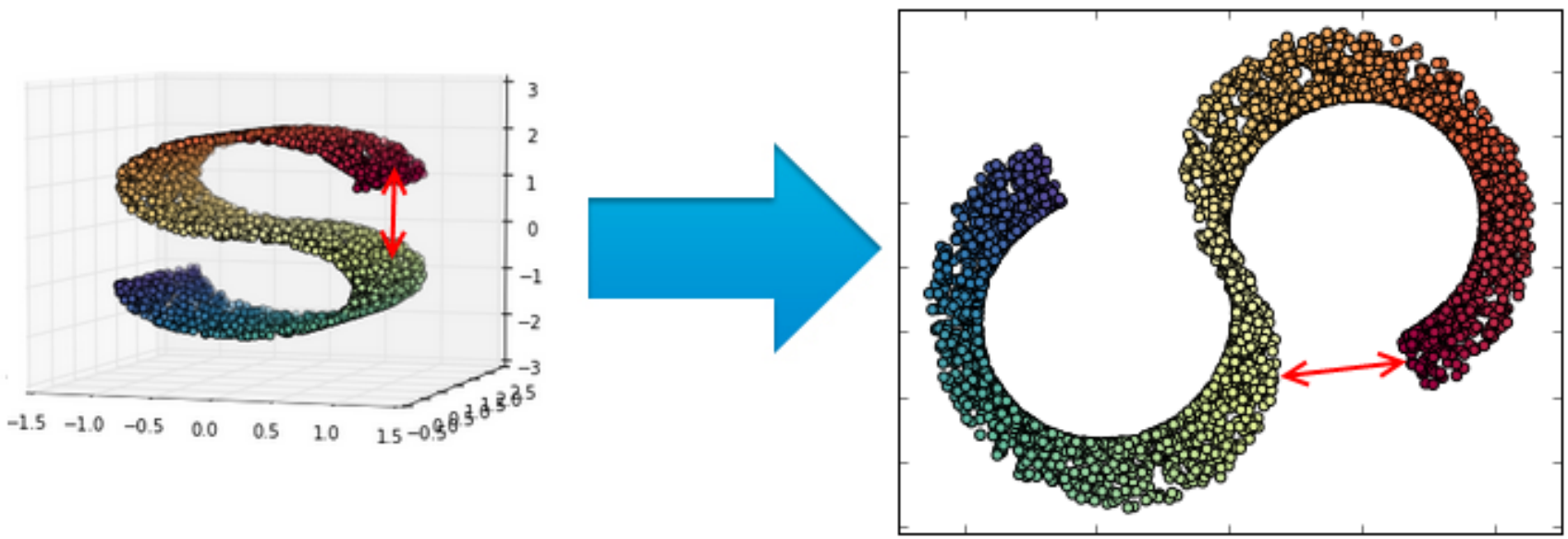
  - relative difference (more attention to small distances)

$$\sum_{i,j} \frac{\left(\|x_i - x_j\| - \|y_i - y_j\| \right)^2}{\|x_i - x_j\|^2} \to \min_Y$$

Example



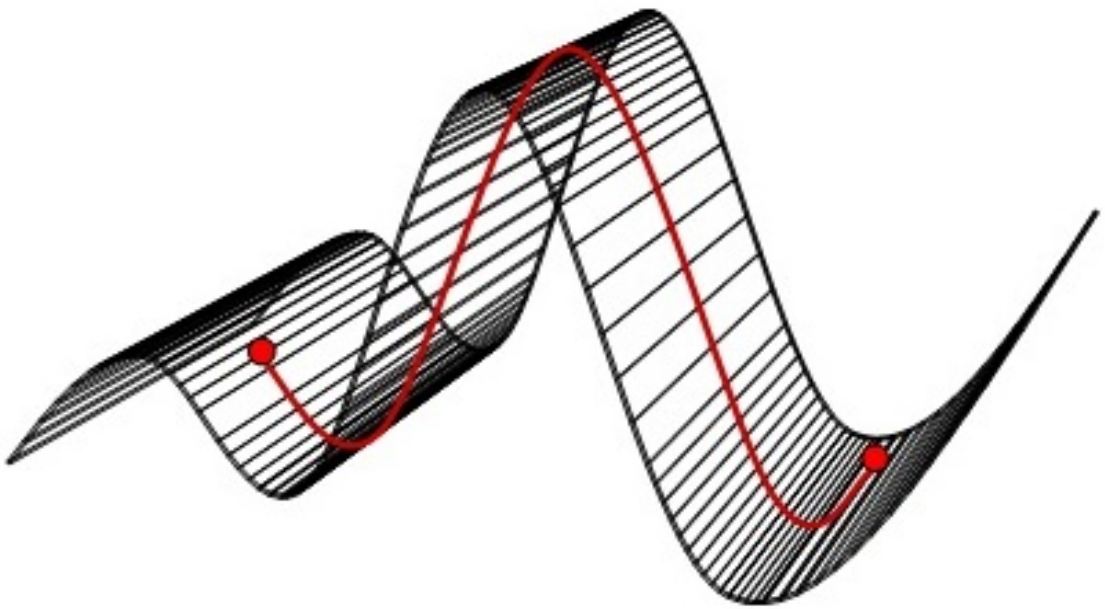
Analysis



Issue: small $\|x_i - x_j\|$ should not always imply small $\|y_i - y_j\|$.

Solution

Isomap: Map $x \to y$ preserving correspondence between distance in target space and geodesic distance along the surface in original space.



Isomap

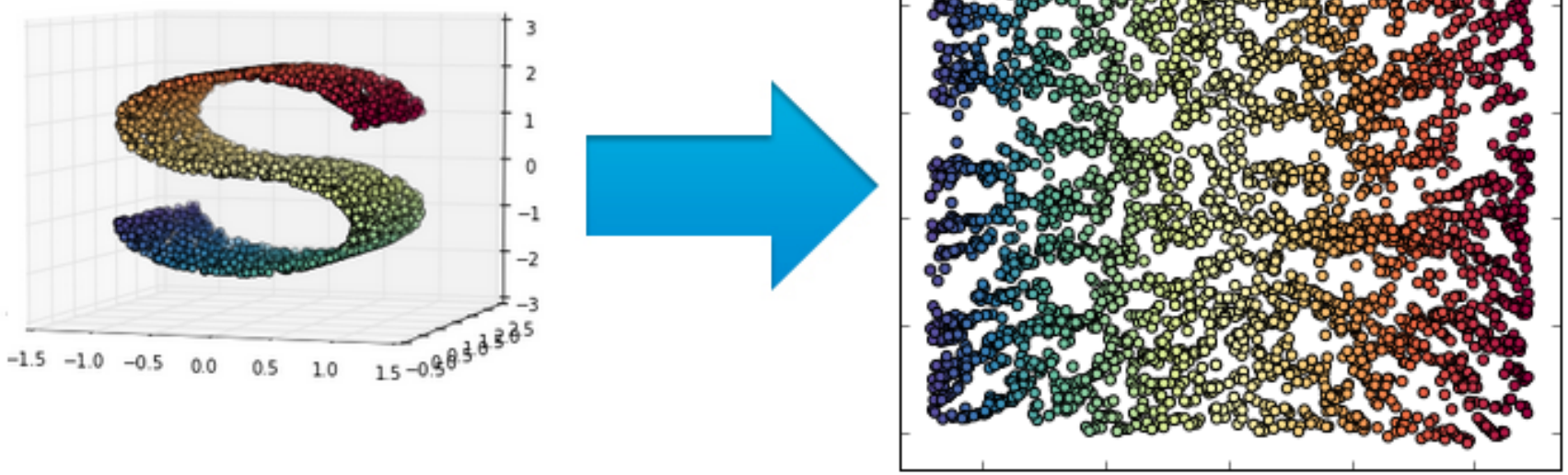
Isomap algorithm

1. Geodesic distance calculation:
   1. for each x_n find its K nearest neighbours
   2. build the pairwise distance matrix, filling distance between samples and their nearest neighbours.
   3. calculate all pairwise distances using shortest-path algorithm of Dijkstra or Floyd.
2. Apply MDS to match $\|x_i - x_j\|_G$ and $\|y_i - y_j\|$, where $\|\cdot\|_G$ is geodesic distance.

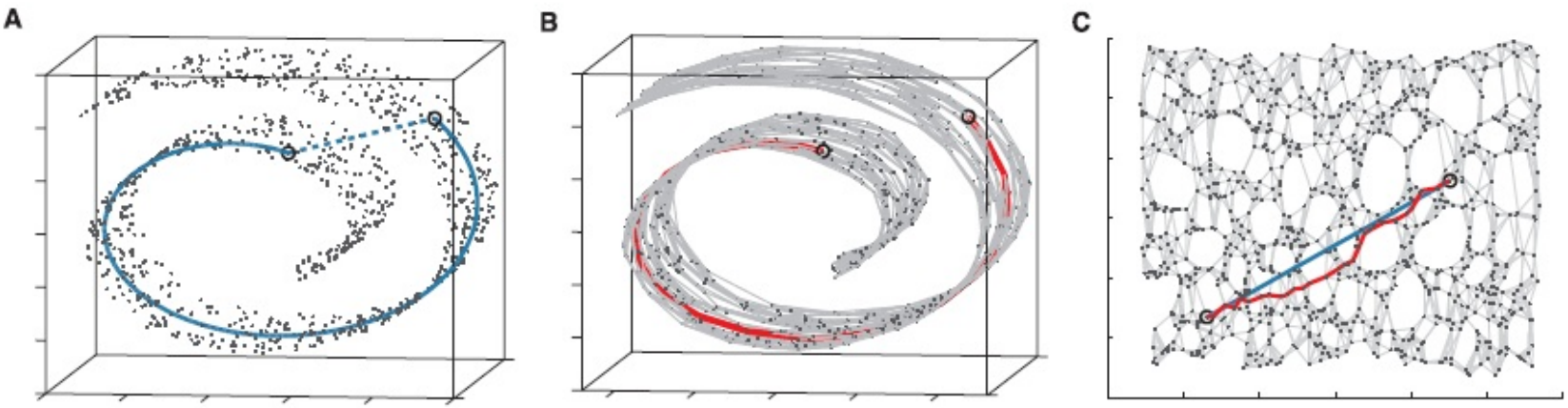
Issues of Isomap

- Noisy observations between distant parts of surfaces may make distant parts close
- Solutions:
  - remove observations with large total flows through them
  - remove nearest neighbours that violate local linearity
- Selection of K:
  - if too small, then poor approximation of geodesic distance
  - if too large, then increases chance of "short-circuiting" through noisy observations.

Example of ISOMAP



Example of ISOMAP[1]



[1] Picture source.

Maximum variance unfolding

Idea of MVU - maximally unfold the transformations, preserving local geometry of data.

initialize neighbourhood graph G with nodes being the samples $x_1, x_2, ...x_N$

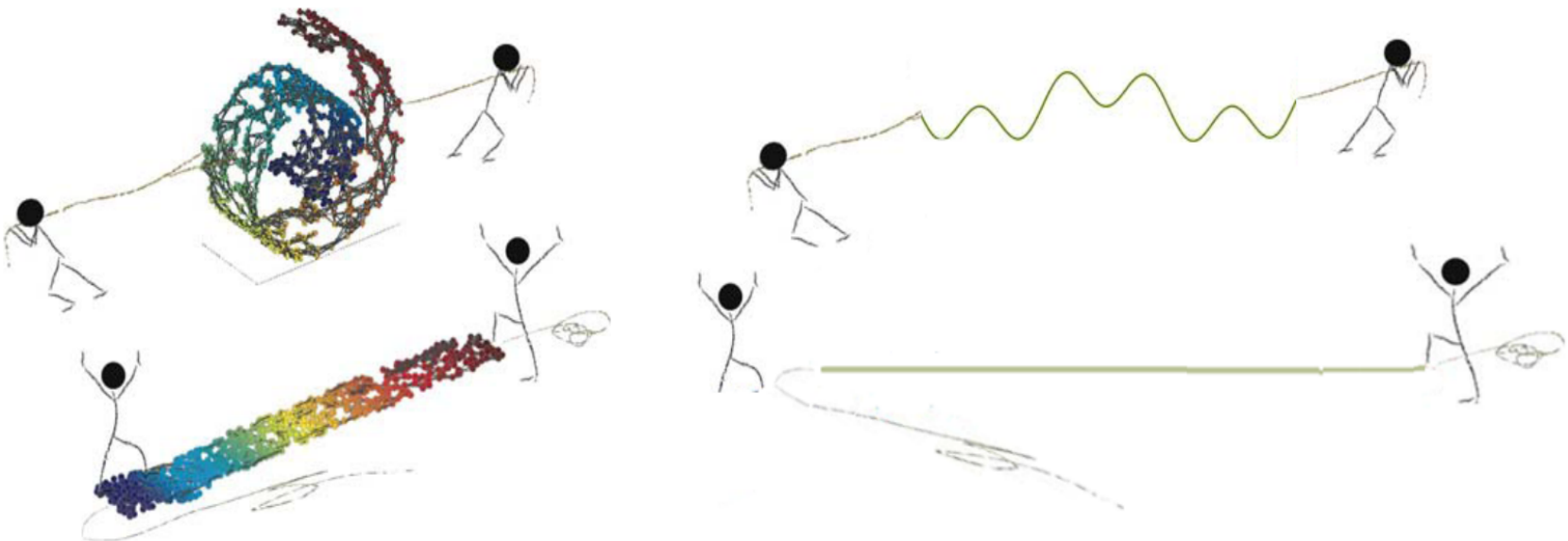
for each x_n:
  for $k = 1, 2, ...K$:
    find k-th nearest neighbour x_{n_k} to x_n
    add a link to G between x_n and x_{n_k}

solve the optimization problem:
$\sum_{i,j} ||y_i - y_j||^2 \to \max$ subject to: $||y_i - y_j||^2 = ||x_i - x_j||^2 \ \forall (i,j) \in G$

- noise sample may add redundant constraint, which may prevent manifold unfolding.
- $||y_i - y_j||^2 = ||x_i - x_j||^2 \ \forall (i,j) \in G$ - may have no solutions! So we can try to keep it small.

Visualization[2]

Unfolding, when nearest neighbours are tied firmly to each other:



[2] Picture source.

Kernel PCA

- Like PCA, but input space is expanded with kernels
- Easy computation of projections of new points
- Issue: kernel selection.
  - linear (reduces to ordinary PCA)
  - Gaussian
  - polynomial

Diffusion maps

1. Construct proximity graph
   - nodes: observations
   - edge weight between x_i and x_j:

$$w_{ij} = e^{-\frac{\left\|x_i - x_j\right\|^2}{2\sigma^2}}$$

2. for each x_i outgoing probabilities set to normalized weights:

$$p_{ij}^{(1)} = \frac{w_{ij}}{\sum_k w_{ik}} \tag{1}$$

3. random walk with probabilities $p_{ij}^{(1)}$ stored in matrix $P^{(1)}$ is assumed.
4. based on random walk assumption, the probability of walking from x_i to x_j after T steps is:

$$p_{ij}^{(T)} = \{\underbrace{P^{(1)} \times \ldots \times P^{(1)}}_{T \text{ times}}\}_{ij}$$

Diffusion maps

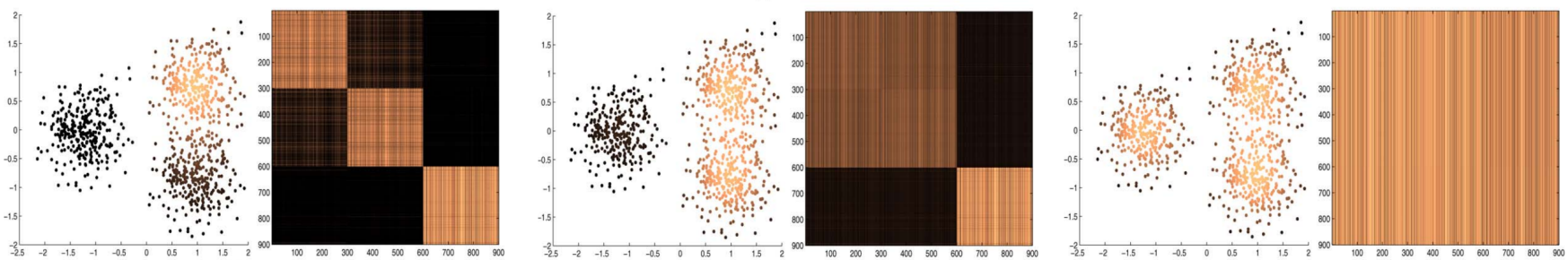
- Finally MDS is applied to match $\|y_i - y_j\|$ to *diffusion distance*:

$$D^T(x_i, x_j) = \sqrt{\sum_k \frac{(p_{ik}^{(T)} - p_{jk}^{(T)})^2}{p_k}}$$

  where $[p_1, p_2, ...p_N]$ is stationary distribution for Markov process with matrix $P^{(1)}$.
- p_i measures the probability to be at object i after big fixed number of trials.
- High p_k means that object k is central, connected to many objects.
- Normalization by p_k: connection to distant isolated objects is more important.

Discussion

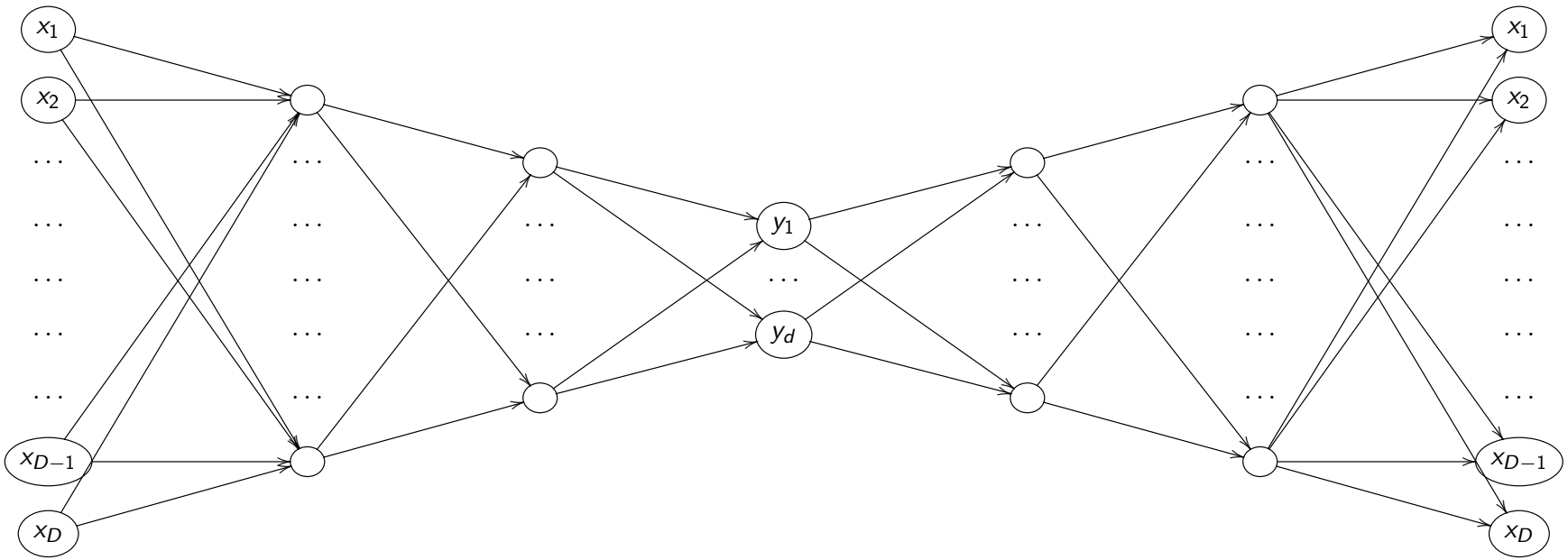
- **Benefit:** distance between points is based on multiple paths through the graph - more robust to noise.
- Selection of T is important:
  - too small: method from global becomes local, matching distances between neighbouring points
  - too big: all points become equally similar
    - Example: 3 clusters with transition probabilities set with (1), color indicates $p(i|j)$ after t steps.



(a) $t = 8$ (b) $t = 64$ (c) $t = 1024$

Link to picture source

Autoencoders



Autoencoders

- feed-forward neural network, tranined to reproduce input with MSE loss.
- D input and D output nodes
- d nodes in the central layer
- $x \in \mathbb{R}^D$ is transformed to $y \in \mathbb{R}^d$.
- User-defined number of layers and nodes

- **Advantages:**
  - can transform arbitrary x to lower-dimensional space
- **Disadvantages:**
  - slow convergence
    - may train layer by layer, then finetune all.
  - optimization gets stuck in local optima
  - many parameters (weights)
    - especially for big D and several layers.

Table of Contents

Local linear embedding

Local linear embedding

Method preserves reconstruction weights of objects through their nearest neighbors.

INPUT:
training sample $x_1, x_2, ...x_N$
number of neighbours K

ALGORITHM:
for each x_i:
find its K nearest neighbours: $x_{i(1)}, x_{i(2)}, ...x_{i(K)}$
find weights to reconstruct x_i using its neighbours:
$x_i \approx \sum_{k=1}^{K} w_{ik} x_{i(k)}$

solve optimization problem: $\sum_{n=1}^{N} (y_i - \sum_{k=1}^{K} w_{ik} y_{ik})^2 \to \min_Y$

OUTPUT: reduced space representation: $y_1, y_2, ...y_N$.

Weights

For $i = 1, 2 \ldots N$:

$$\begin{cases} \left\| w_{ik} x_{i(k)} - x_i \right\|^2 \to \min_{w_{i1}, \ldots w_{iK}} \\ \sum_{j=1}^{K} w_{ij} = 1 \end{cases}$$

Laplacian eigenmaps

Laplacian eigenmaps

Forces distances of points with nearest neighbours to be smaller.

INPUT:
training sample $x_1, x_2, \ldots x_N$
number of neighbours K

ALGORITHM:
for each x_i:
find its K nearest neighbours: $x_{i(1)}, x_{i(2)}, \ldots x_{i(K)}$
for each nearest neighbour j=$i(1), i(2), \ldots i(K)$:
calculate distance-based weights: $w_{ij} = e^{-\frac{\|x_i - x_j\|^2}{2\sigma^2}}$

solve optimization problem:
$\sum_{i=1}^{N} \sum_{j \in \{i(1), \ldots i(K)\}} w_{ij}(y_i - y_j)^2 \to \min_Y$

OUTPUT: reduced space representation: $y_1, y_2, \ldots y_N$.

Comments on local methods

- short-circuiting affects only local points in space
- local method, relying on K-NN => prone to curse of dimensionality
- prone to overfitting on outliers (when they become nearest neighbors)

Properties

Technique	Convex	Parameters	Computational	Memory
PCA	yes	none	$O(D^3)$	$O(D^2)$
MDS	yes	none	$O(N^3)$	$O(N^2)$
Isomap	yes	K	$O(N^3)$	$O(N^2)$
MVU	yes	K	$O((NK)^3)$	$O((NK)^3)$
Kernel PCA	yes	kernel	$O(N^3)$	$O(N^2)$
Diffusion maps	yes	σ, T	$O(N^3)$	$O(N^2)$
Autoencoders	no	network shape	$O(INW)$	$O(W)$
LLE	yes	K	$O(pN^2)$	$O(pN^2)$
Laplacian eigenmaps	yes	K, σ	$O(pN^2)$	$O(pN^2)$

D - input dimension, N - sample size, K - number of nearest neighbors, σ - smoothing parameter of Gaussian kernel, W number of weights in neural network, I - number of epochs (passes through whole training set), p - the fraction of non-zero entries in the weight matrix.

Comment: PCA is the most efficient, then come local methods (italic) and finally global methods.

Global vs. local methods

- Global methods try to preserve the whole geometry of data
  - less efficient
  - find "overall picture"
  - noise points can spoil whole picture
- Local methods try to preserve only local data geometry
  - more efficient
  - find "locally correct pictures", then join them
  - locally affected by noise points

Comments

- **Problem of transforming new previously unobserved samples.**
  - direct for PCA, Kernel PCA, autoencoders
  - only approximations possible for other methods.
    - suppose for new x its nearest neightbours form training set are: $x_{i(1)}, \ldots x_{i(K)}$
    - $x \approx \sum_{k=1}^{K} w_k x_{i(k)}$, so $y(x) \approx \sum_{k=1}^{K} w_k y(x_{i(k)})$
- **Selection of target dimensionality d:**
  - Cross-validation of the original task (e.g. classification)
  - How many components of local PCA explain most of the variance?
  - The growth rate of number of objects falling inside a growing hypersphere with center x:$\#\{x_i : \|x_i - x\| \leq R\}$
    - for d-dimensional manifold it should grow $\propto R^d$.
  - etc.

Experiment

- *L.J.P. van der Maaten, E.O. Postma, H.J. van den Herik. Dimensionality Reduction:A Comparative Review. Working paper. 2008.*
  - Extensive comparison of different dimensionality reduction methods
    - accuracy of 1 nearest neighbour in reduced space.
  - Non-linear techniques perform better than PCA on simulated data
  - PCA wins most of the time on real data
  - Problems:
    - global methods: short-circuiting
    - nearest neighbours based methods: curse of dimensionality, overfitting to outliers
    - unstable optimization for local methods: they reduce to eigenproblems, frequently $\lambda_{max}/\lambda_{min} \gg 1$.
    - suboptimal local optima for autoencoders.

Dangers of dimensionality reduction