

(Non-linear) dimensionality reduction

Jiří Kléma

Department of Computer Science,
Czech Technical University in Prague



<http://cw.felk.cvut.cz/wiki/courses/b4m36san/start>

Outline

- motivation, task definition,
- linear approaches such as PCA
 - and its non-linearization: kernel PCA,
- distance preserving approaches
 - multidimensional scaling,
 - Isomap,
 - locally linear embedding,
 - tSNE,
- self-organizing maps
 - a vector quantization approach,
 - their relation to k-means clustering,

Task definition

- Input: $\mathbf{X} = \{\mathbf{x}_i\}_{i=1}^m \subset \mathcal{X}$ of dimension D (typically \mathbb{R}^D),
- assumed: \mathbf{X} at least approximately lies on a manifold with $d < D$,
- output:
 - a transformed space \mathcal{T} of dimension L ,
 - dimensionality reduction mapping $\mathbf{F} : \mathcal{X} \rightarrow \mathcal{T}$,
 - reconstruction mapping $\mathbf{f} : \mathcal{T} \rightarrow \mathcal{M} \subset \mathcal{X}$,
- such that:
 - $L < D$, L is as small as possible, at best $L = d$ (the intrinsic dimension),
 - the manifold approximately contains all the sample points

$$\{\mathbf{x}_i\}_{i=1}^m \underset{\sim}{\subseteq} \mathcal{M} \stackrel{def}{=} \mathbf{f}(\mathcal{T}),$$

- or alternatively, the reconstruction error of the sample is small

$$E_d(\mathbf{X}) \stackrel{def}{=} \sum_{i=1}^m d(\mathbf{x}_i, \mathbf{f}(\mathbf{F}(\mathbf{x}_i))).$$

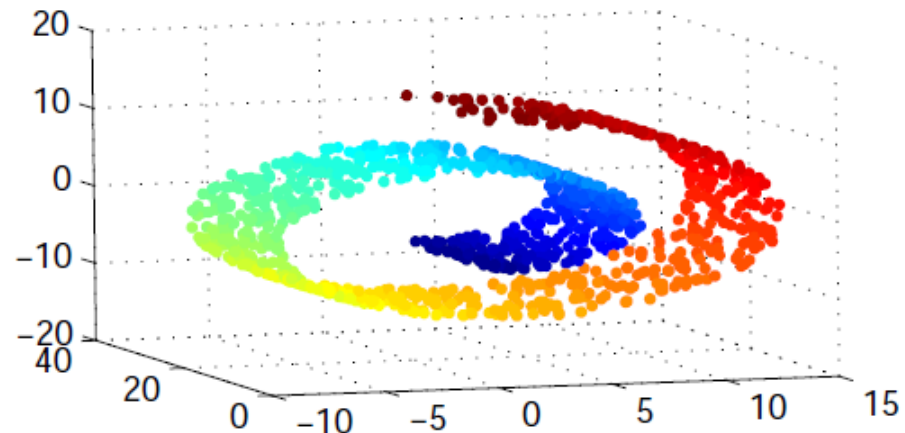
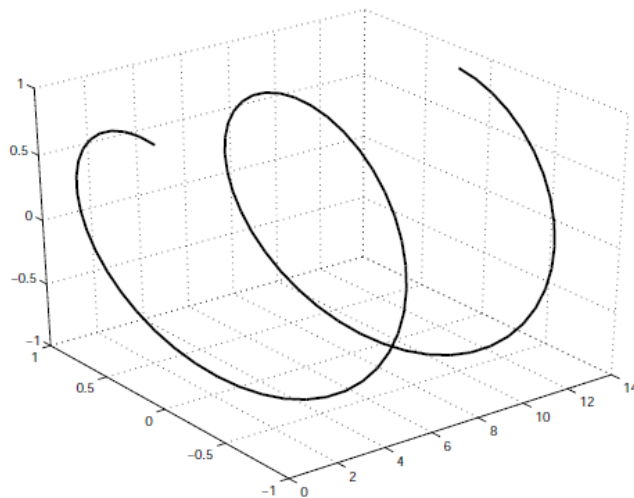
Manifold learning

:: Manifold

- a topological space that on a small enough scale resembles the Euclidean space,
- globally typically nonlinear,

:: Learning

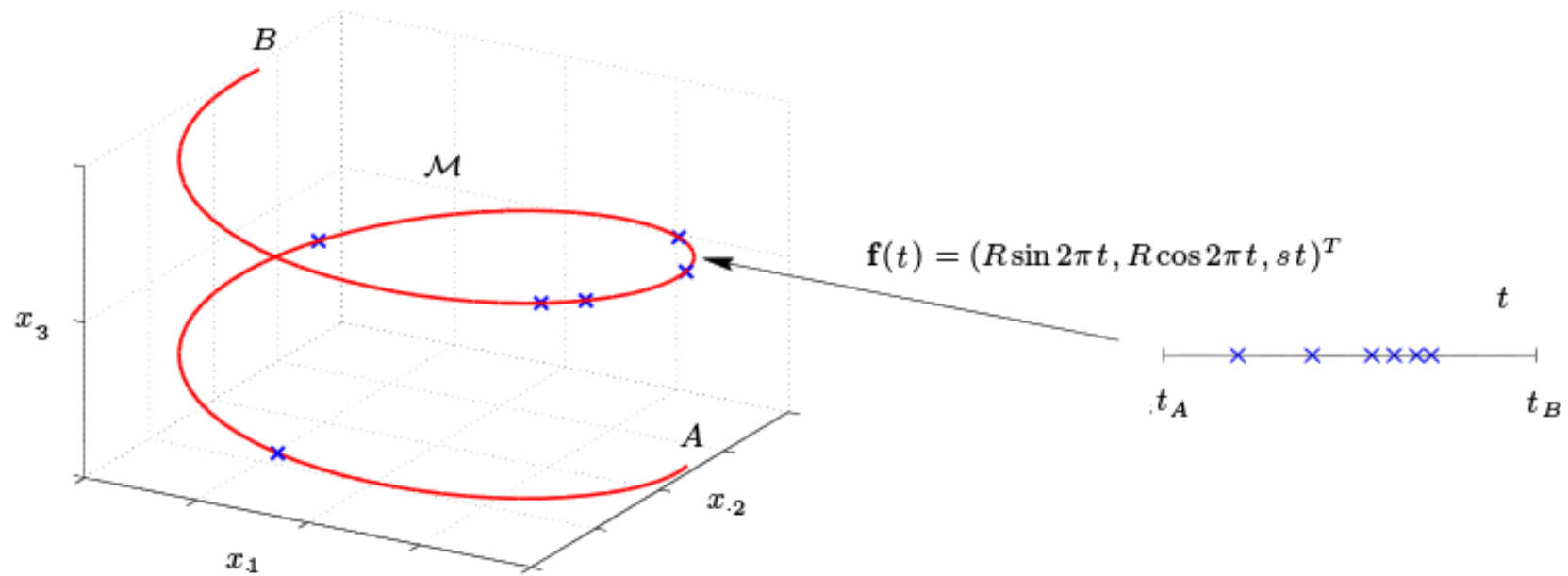
- identify a manifold dimension (it is embedded in a space of a higher dimension),
- project the problem (objects) into the low dimensional space – nonlinear dimension reduction,
- linear analogy: PCA or factor analysis.



Cayton: Algorithms for Manifold Learning.

Example of a manifold and its mapping

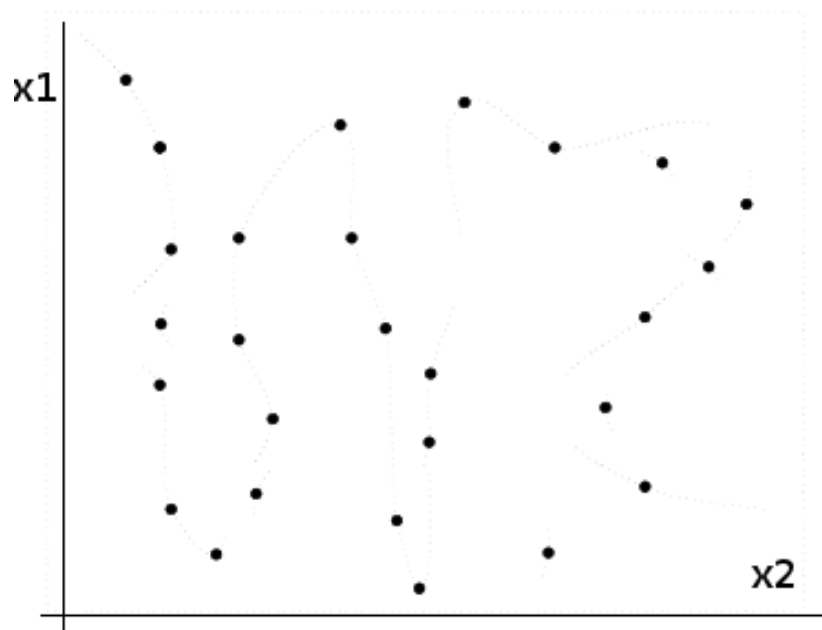
- A spiral of radius R and step s
 - 1D non-linear manifold in \mathbb{R}^3 ,
 - no noise, the given $f(t)$ guarantees zero reconstruction error.



Carreira-Perpinan: A Review of Dimension Reduction Techniques

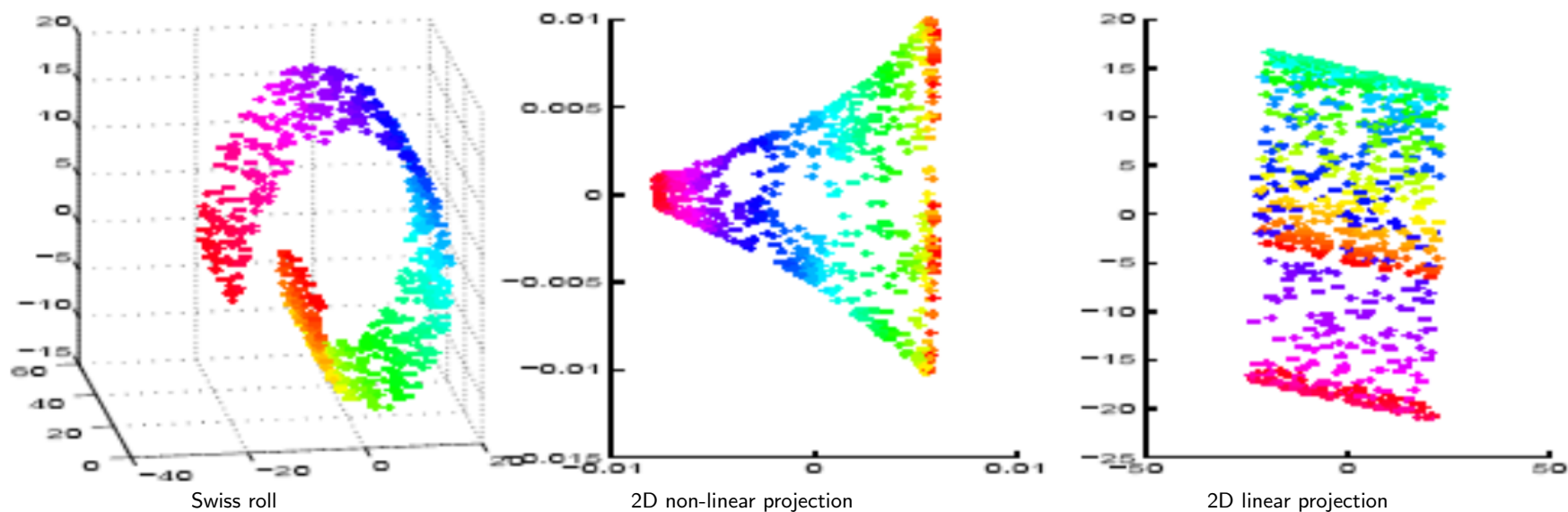
Intrinsic dimension

- the number of variables that **satisfactorilly** describe the phenomenon/data,
- when estimated from data, it is a vague number, it depends on
 - the minimum approximation quality criterion or alternatively,
 - smoothness of the manifold,
- it does not have to be a natural number (can be e.g., 1.4, consider fractals).



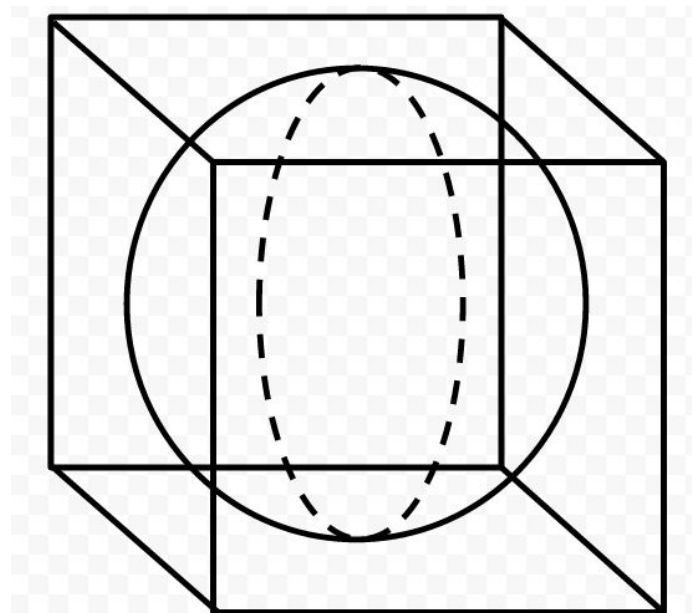
Motivation

- Visualize the data and understand them (typically project into 2 or 3-D),
- compress the data to minimize storage and retrieval cost,
- identify hidden causes/latent variables that govern the process under study,
- learn in the lower-dimensional space
 - possibly obtain better results with fewer training samples in shorter time,
 - a lot of work has already been done during the projection.



The challenges of high-dimensional spaces

- The curse of dimensionality
 - in the absence of **simplifying assumptions**, the sample size needed to estimate a function with D variables to a given **degree of accuracy** grows **exponentially** with D ,
- the geometry of high-dimensional spaces
 - the empty space phenomenon,
 - guess what is the ratio of the volumes of unit hypersphere and unit hypercube . . .



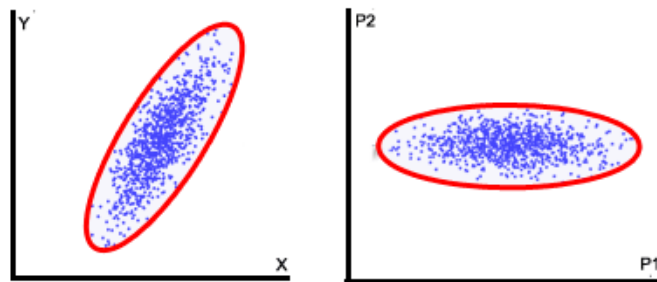


Some data science news ...

- One of the KD Nuggets top news in 2017:
17 More Must-Know Data Science Interview Questions and Answers,
- a great part of the general questions touched in this course, examples:
 - What problems arise if the distribution of the new (unseen) test data is significantly different than the distribution of the training data?
 - What are bias and variance, and what are their relation to modeling data?
 - Why might it be preferable to include fewer predictors over many?
 - What error metric would you use to evaluate how good a binary classifier is? What if the classes are imbalanced? What if there are more than 2 groups?
 - What are some ways I can make my model more robust to outliers?

A brief review of PCA

- PCA could be seen as fitting a (hyper)ellipsoid to the data
 - the new axes have the direction of the highest variance,
 - they match the axes of the encapsulating/confidence ellipsoid.



- What happens in terms of covariances and redundancy?
 - left image: $\sigma_Y^2 > \sigma_{XY}^2 > \sigma_X^2$,
 - right image: $\sigma_{P1}^2 \gg \sigma_{P2}^2$, $\sigma_{P1P2}^2 = 0$,
 - in general, PCA **diagonalizes the covariance matrix**,
 - in other words, PCA removes linear relationship (redundancy) between variables.

A brief review of PCA

- For \mathbf{X} with zero centered variables

$$\sum_{i=1}^m \mathbf{x}_i = 0$$

- the covariance matrix can be computed as follows

$$\mathbf{C}_X = \frac{1}{m} \sum_{i=1}^m \mathbf{x}_i \mathbf{x}_i^T = \frac{1}{m} \mathbf{X}^T \mathbf{X}$$

- by definition, PCA constructs a space transformation matrix $\mathbf{P}_{D \times D}$, such that

$$\mathbf{X}\mathbf{P} = \mathbf{T} \text{ and } \mathbf{C}_T \text{ is a diagonal matrix}$$

- as \mathbf{P} is not known, \mathbf{C}_T cannot be calculated and diagonalized directly,
- instead, \mathbf{C}_T can be expressed in terms of \mathbf{C}_X and \mathbf{P}

$$\begin{aligned} \mathbf{C}_T &= \frac{1}{m} \mathbf{T}^T \mathbf{T} = \frac{1}{m} (\mathbf{X}\mathbf{P})^T (\mathbf{X}\mathbf{P}) = \\ &= \frac{1}{m} \mathbf{P}^T (\mathbf{X}^T \mathbf{X}) \mathbf{P} = \mathbf{P}^T \mathbf{C}_X \mathbf{P} \end{aligned}$$

A brief review of PCA

- any real symmetric matrix is diagonalized by a column matrix of its eigenvectors \mathbf{E} ,
- \mathbf{C}_X is real and symmetric, it follows that

$$\mathbf{C}_X = \mathbf{E}\mathbf{D}\mathbf{E}^T$$

- the only trick is to select \mathbf{P} to be a matrix where each column \mathbf{p}_i is an eigenvector of \mathbf{C}_X

$$\mathbf{P} = \mathbf{E}$$

- we also know that any orthogonal matrix has the same inverse and transpose,
- the above-selected \mathbf{P} is necessarily orthogonal

$$\mathbf{P}^T\mathbf{P} = \mathbf{I} \Rightarrow \mathbf{P}^{-1} = \mathbf{P}^T$$

- then, it is easy to show that \mathbf{P} diagonalizes \mathbf{C}_T

$$\begin{aligned}\mathbf{C}_T &= \mathbf{P}^T\mathbf{C}_X\mathbf{P} = \mathbf{P}^T(\mathbf{E}\mathbf{D}\mathbf{E}^T)\mathbf{P} = \\ &= (\mathbf{P}^T\mathbf{P})\mathbf{D}(\mathbf{P}^T\mathbf{P}) = (\mathbf{P}^{-1}\mathbf{P})\mathbf{D}(\mathbf{P}^{-1}\mathbf{P}) = \mathbf{D}\end{aligned}$$

- PCA is solved by finding the eigenvectors of \mathbf{C}_X .

A brief review of PCA

- what happens when D is large?
 - consider images, the color of each pixel is a feature, megapixel resolution,
 - large \mathbf{C}_X , unfeasible computation of its eigenvectors,
- provided that m is reasonable ($m \ll D$), we can employ the following trick,
- instead of the original eigenvector decomposition

$$\frac{1}{m} \mathbf{X}^T \mathbf{X} \mathbf{u}_k = \lambda_k \mathbf{u}_k$$

- we will consider

$$\frac{1}{m} \mathbf{X} \mathbf{X}^T \mathbf{v}_k = \gamma_k \mathbf{v}_k$$

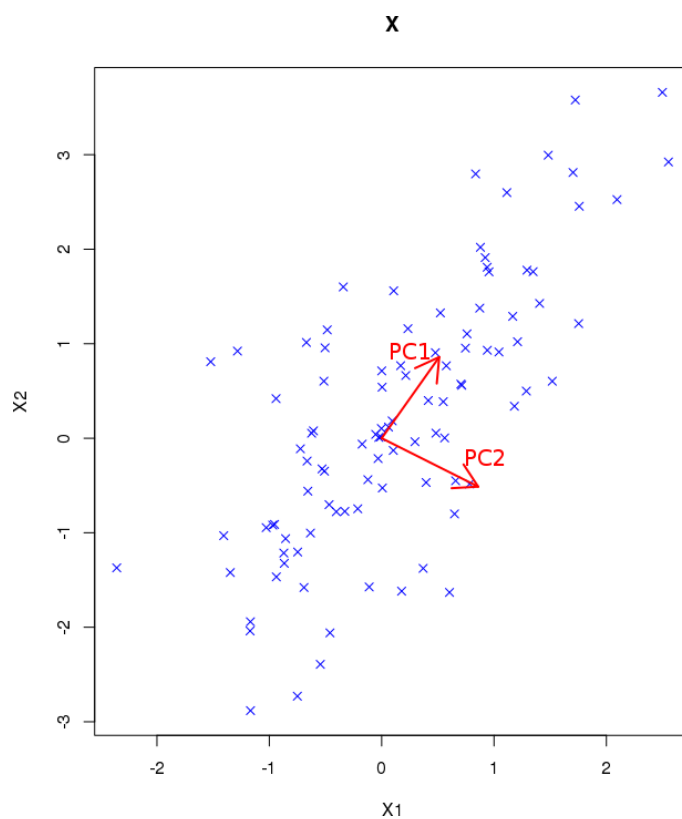
- and multiply both sides by \mathbf{X}^T

$$\frac{1}{m} \mathbf{X}^T \mathbf{X} \mathbf{X}^T \mathbf{v}_k = \gamma_k \mathbf{X}^T \mathbf{v}_k$$

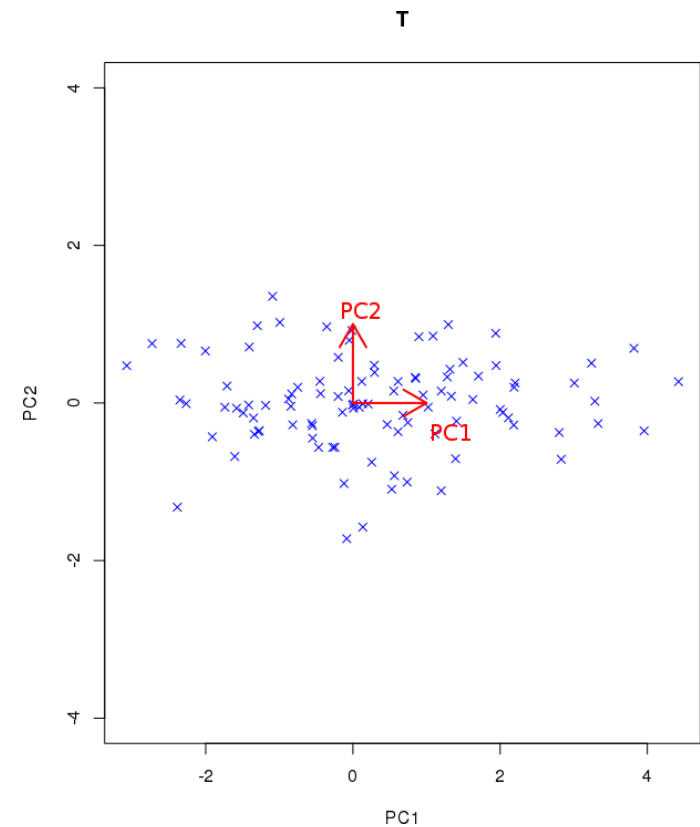
- it is obvious that the substitution $\mathbf{X}^T \mathbf{v}_k = \mathbf{u}_k$ and $\gamma_k = \lambda_k$ matches the original eigenvector decomposition formula,
- PCA can be solved by decomposition of the $m \times m$ scalar-product matrix.

A brief review of PCA

- example: 100 objects in \mathbb{R}^2 space $\rightarrow \mathbf{X}_{100 \times 2}$,
- first, keep all the principal components, no information loss in $\mathcal{X} \xrightarrow{F} \mathcal{T} \xrightarrow{f} \mathcal{X}$
 - reduction mapping $\mathbf{F} : \mathbf{T}_{100 \times 2} = \mathbf{X}_{100 \times 2} \mathbf{P}_{2 \times 2}$,
 - reconstruction mapping $\mathbf{f} : \mathbf{X}_{100 \times 2} = \mathbf{T}_{100 \times 2} \mathbf{P}_{2 \times 2}^T$,

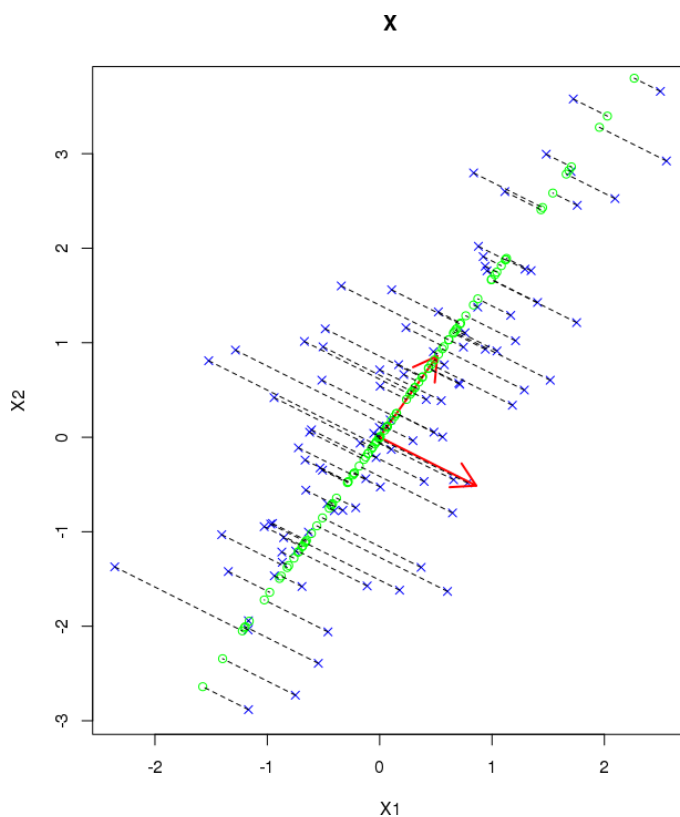


$$\mathbf{P} = \begin{bmatrix} \text{PC1} & \text{PC2} \end{bmatrix}$$
$$= \begin{bmatrix} 0.51 & 0.86 \\ 0.86 & -0.51 \end{bmatrix}$$

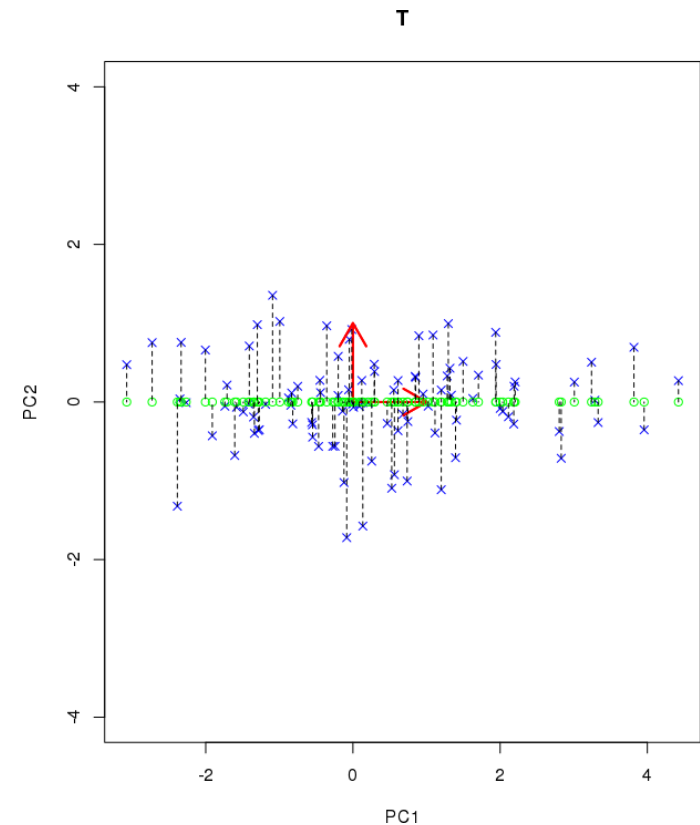


A brief review of PCA

- example: 100 objects in \mathbb{R}^2 space $\rightarrow \mathbf{X}_{100 \times 2}$,
- second, move to 1D, use the first principal component only, dashed lines = information loss,
 - reduction mapping $\mathbf{F} : \mathbf{T}_{100 \times 1} = \mathbf{X}_{100 \times 2} \mathbf{P}_{2 \times 1}$,
 - reconstruction mapping $\mathbf{f} : \mathbf{X}_{100 \times 2} = \mathbf{T}_{100 \times 1} \mathbf{P}_{1 \times 2}^T$,

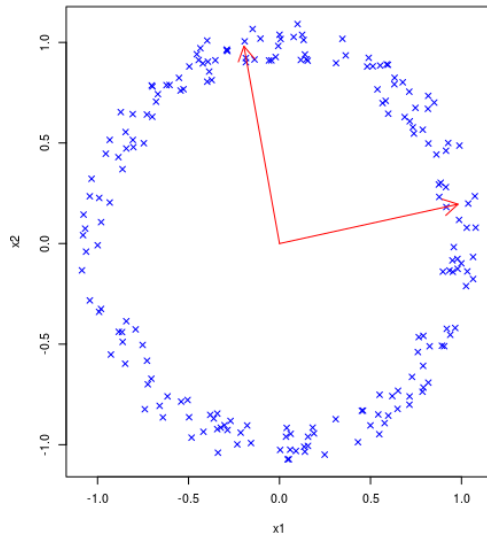


$$\mathbf{P} = \begin{bmatrix} 0.51 \\ 0.86 \end{bmatrix}$$

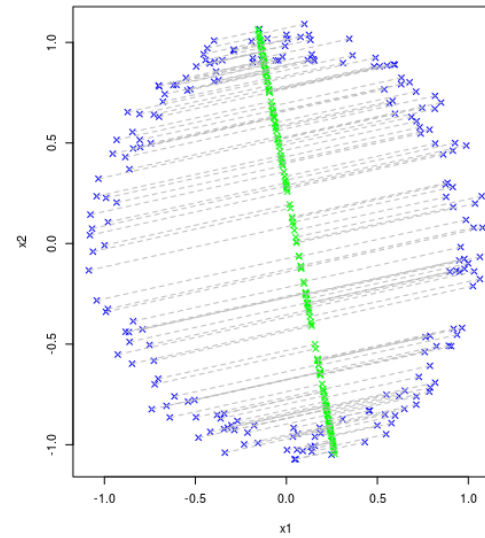


The need for non-linear methods

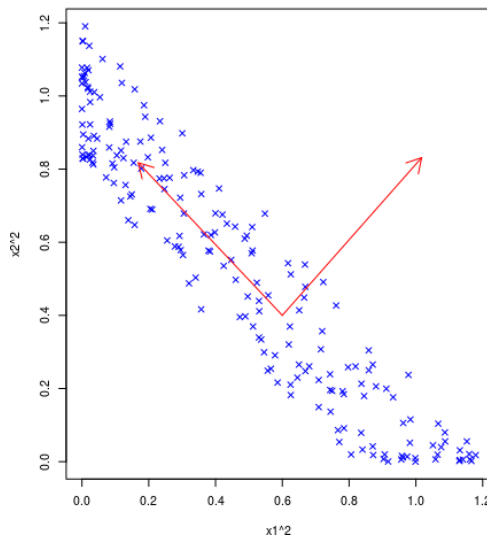
A non-linear manifold, PCA applied directly



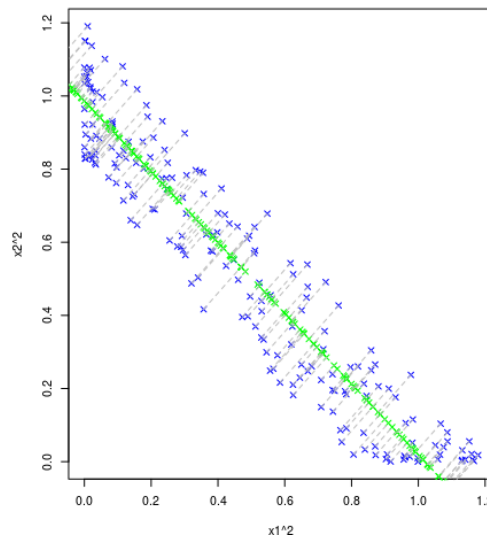
52% of variance captured in PC1



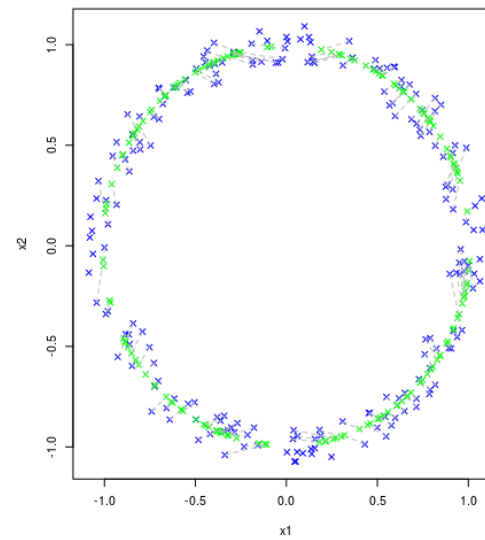
Transform the space before PCA is applied



97% of variance captured in PC1

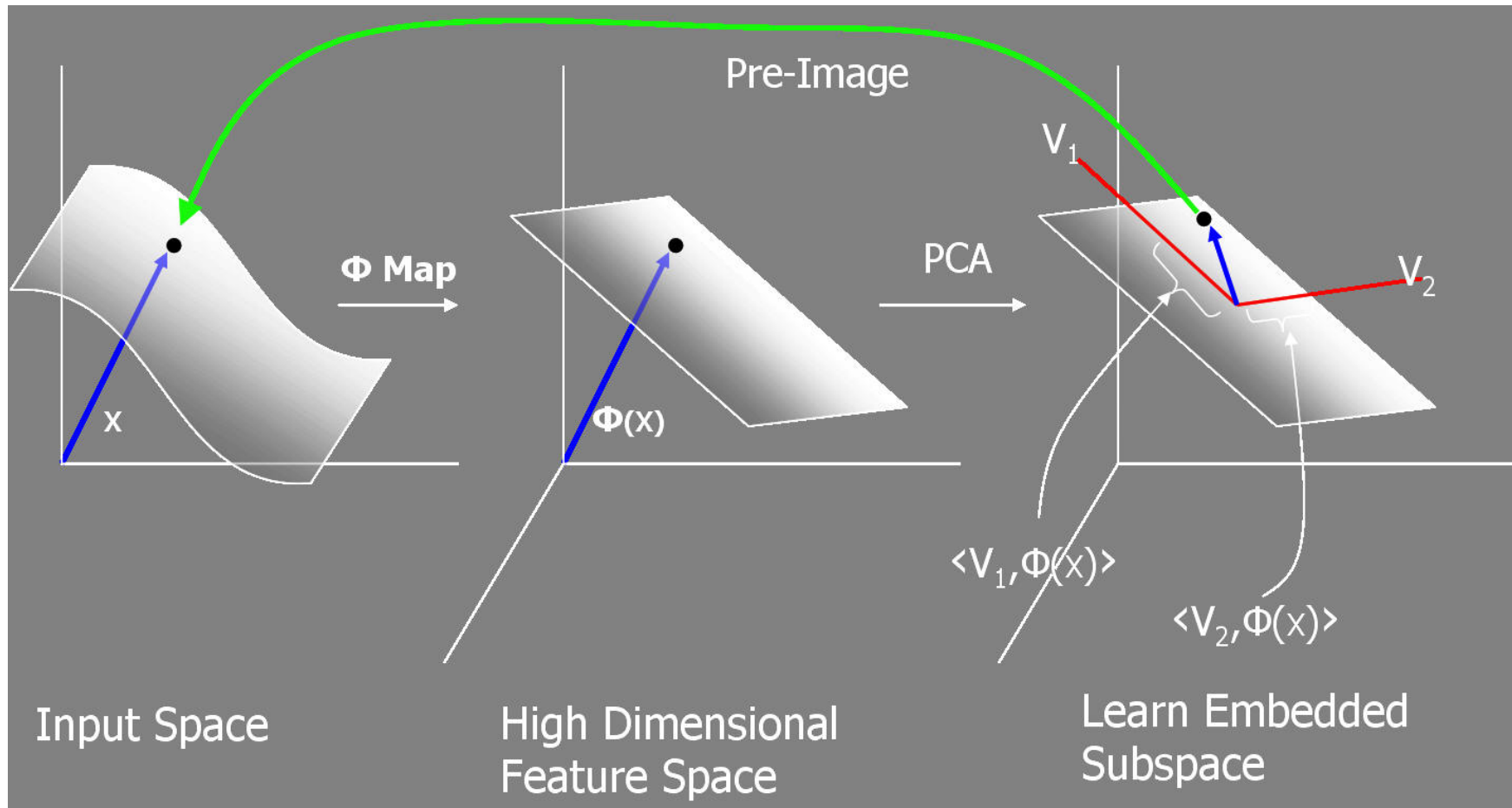


The denoised manifold after the inverse transform



kernel PCA – the idea behind

- Introduce an intermediate feature space \mathcal{U}
 - $\mathcal{X} \rightarrow \mathcal{U} \rightarrow \mathcal{T}$, \mathcal{U} linearizes the original manifold.



<http://www.research.rutgers.edu>

Why kernel in the method name?

- **feature space transformation** can capture non-linearity,
- a domain independent dimensionality reduction, only the transformation tuned for the domain
 - explicit transformation
 - * get the object coordinates in the new feature space,
 - * traditional PCA in the new space,
 - * illustrative, but impractical,
 - implicit transformation
 - * via similarity resp. kernel function $K : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$,
 - * purely a function of object pairs, no object coordinates in the new space,
 - * very natural for clustering, similarity/distance its essential part anyway,

 - * kernel trick analogy (SVM classification),
 - an implicit high-dimensional space, classes potentially easily separable,
 - * very natural in clustering too, see kernel k-means later,

 - * kernel PCA – kernel matrix \rightarrow diagonalize \rightarrow a low-dimensional feature space.

kernel PCA

- We choose a (non-linear) feature space transformation

$$\phi : \mathcal{X} \rightarrow \mathcal{U}$$

- the transformation is implicit, we only know the kernel function

$$\forall \mathbf{x}_i, \mathbf{x}_j \in \mathcal{X} : \mathbf{K}(\mathbf{x}_i, \mathbf{x}_j) := \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$$

- as with the PCA, we will assume the covariance matrix, now in the transformed space

$$\mathbf{C}_U = \frac{1}{m} \sum_{i=1}^m \phi(\mathbf{x}_i) \phi(\mathbf{x}_i)^T$$

- note that the data are assumed to be centered

$$\sum_{i=1}^m \phi(\mathbf{x}_i) = 0$$

- similarly to PCA, we will find \mathbf{C}_U eigenvectors \mathbf{v} to decorrelate variables in \mathcal{T}

$$\mathbf{C}_U \mathbf{v} = \lambda \mathbf{v} \rightarrow \frac{1}{m} \sum_{i=1}^m \phi(\mathbf{x}_i) \phi(\mathbf{x}_i)^T \mathbf{v} = \lambda \mathbf{v}$$

kernel PCA

- $\phi(\mathbf{x}_i)$ are not available, we need to replace them by \mathbf{K} ,
- for $\lambda \geq 0$, \mathbf{v} 's are in the span of $\phi(\mathbf{x}_i)$,
- they can be written as linear combination of the object images

$$\mathbf{v} = \sum_{i=1}^m \alpha_i \phi(\mathbf{x}_i)$$

- we will substitute for \mathbf{v} into the eigenvector formula (the last in the previous slide)

$$\lambda \sum_{j=1}^m \alpha_j \phi(\mathbf{x}_j) = \frac{1}{m} \sum_{i=1}^m \phi(\mathbf{x}_i) \phi(\mathbf{x}_i)^T \sum_{j=1}^m \alpha_j \phi(\mathbf{x}_j)$$

- and use the trick to introduce the dot product, we will multiply by $\phi(\mathbf{x}_k)^T$

$$\lambda \sum_{j=1}^m \alpha_j \phi(\mathbf{x}_k)^T \phi(\mathbf{x}_j) = \frac{1}{m} \sum_{j=1}^m \alpha_j \sum_{i=1}^m (\phi(\mathbf{x}_k)^T \phi(\mathbf{x}_i)) (\phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j))$$

- the kernel function replaces all the occurrences of ϕ , when iterating $\forall k = 1 \dots m$ we obtain

$$\lambda \mathbf{K} \alpha = \frac{1}{m} \mathbf{K}^2 \alpha$$

- to diagonalize \mathbf{C}_T , we solve the eigenvalue problem for the kernel matrix

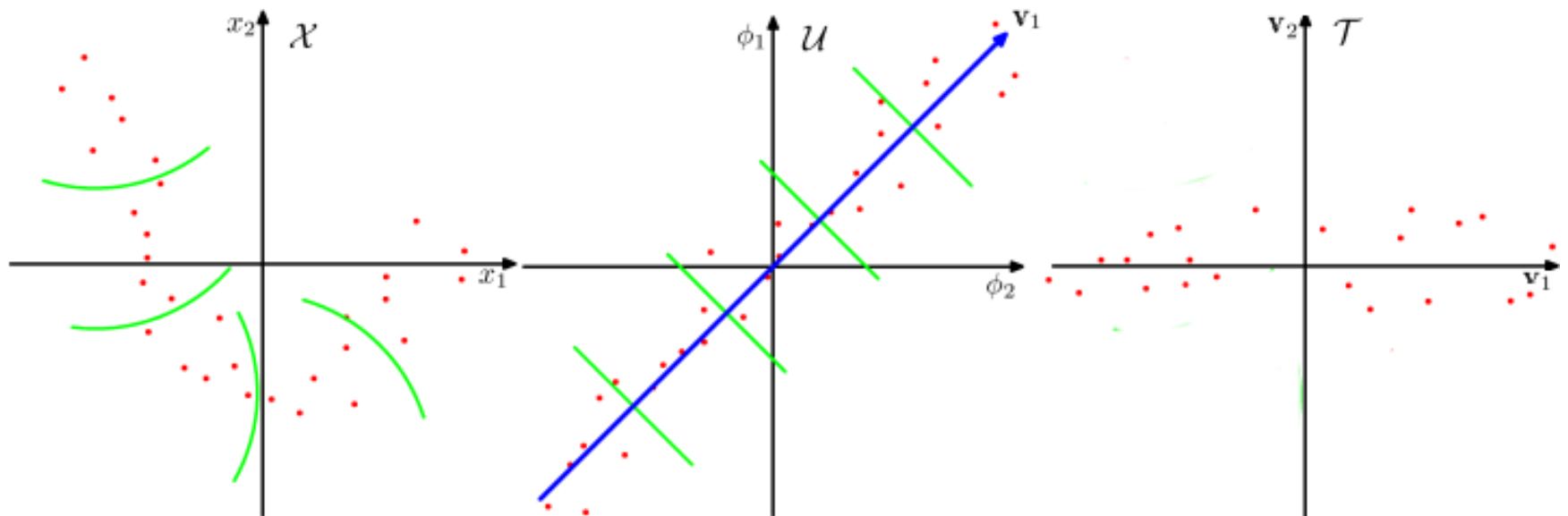
$$m\lambda\alpha = \mathbf{K}\alpha$$

kernel PCA

- the last issue is to extract the principal components, the final object images,
- i.e., the projections of $\phi(\mathbf{x}_i)$ onto the eigenvectors in \mathcal{U}

$$t_{ik} = \mathbf{v}_k^T \phi(\mathbf{x}_i) = \sum_{j=1}^m \alpha_j \phi(\mathbf{x}_j)^T \phi(\mathbf{x}_i) = \sum_{j=1}^m \alpha_j \mathbf{K}(\mathbf{x}_i, \mathbf{x}_j)$$

- note that both the intermediate factors \mathbf{v}_k and $\phi(\mathbf{x}_i)$ remain implicit!



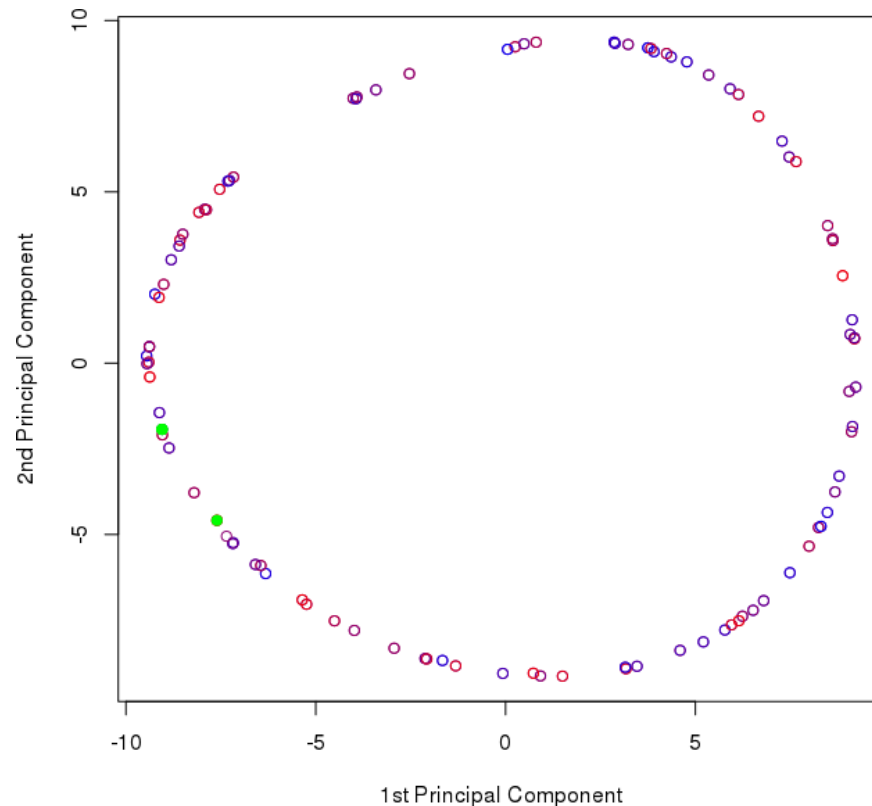
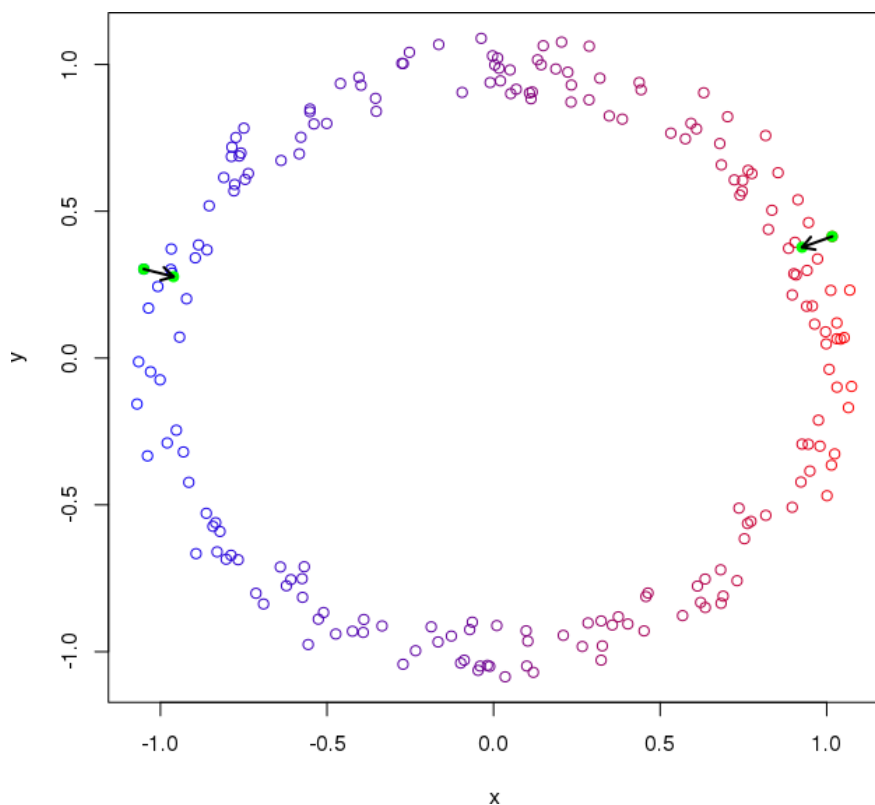
Bishop: Pattern Recognition and Machine Learning, modified

Remarks on kernel PCA

- kernel PCA works for non-linear manifolds
 - effectively compresses them,
- its complexity does not grow with the dimensionality of \mathcal{U}
 - one can work with a large number of components too, i.e., increase the dimension ($L > D$),
 - in PCA $L \leq D$
 - it can pay-off in subsequent classification,
- kernel matrix \mathbf{K} grows quadratically with the number of data points m
 - for large data sets with small dimensionality ($m \gg D$) more expensive than PCA,
 - one may need to (properly) subsample the data,
- from the optimization point of view cannot get trapped in local minima,
- unlike PCA cannot reconstruct objects from their principal components
 - \mathbf{f} is not available.

Remarks on kernel PCA

- example: kernel PCA for denoising,
- can we reconstruct the manifold?
 - remember that f is not available, **pre-images** can be reconstructed for certain kernels,
 - often used e.g. in image processing to obtain the denoised images.



Multidimensional scaling (MDS)

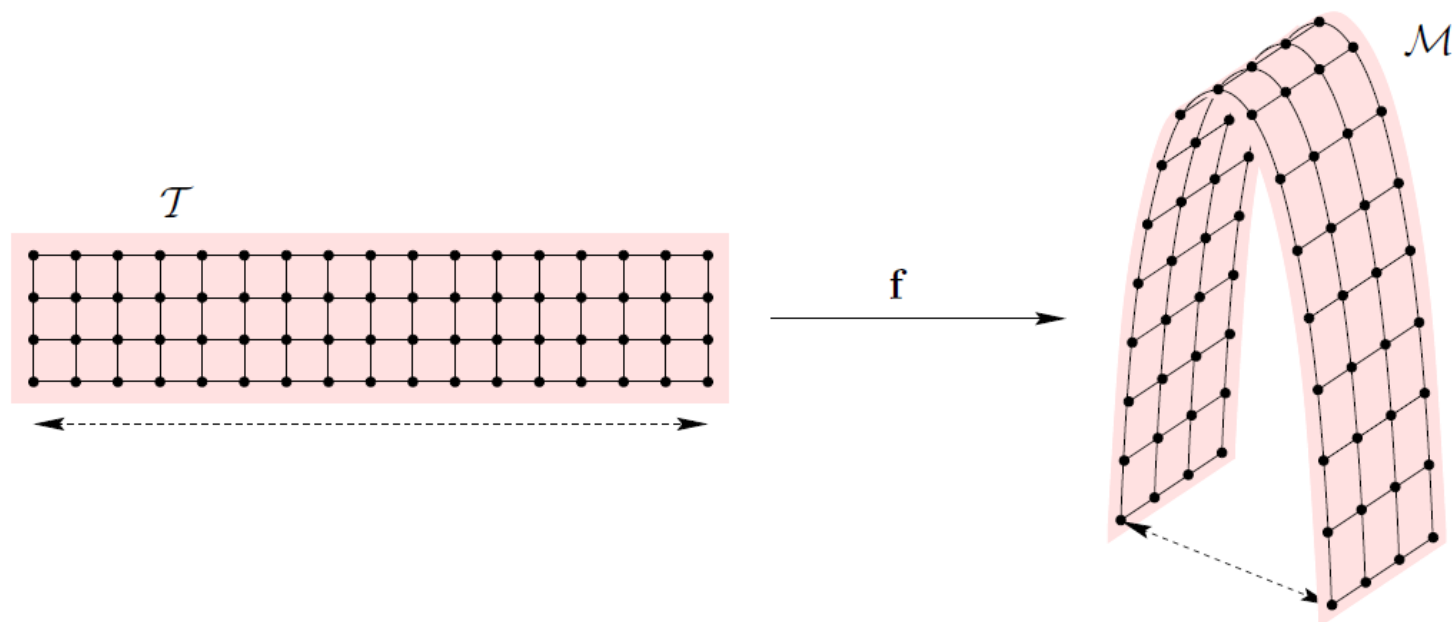
- The main idea
 - points close together in \mathcal{X} should be mapped close together in \mathcal{T} ,
- minimizes the **stress function**

$$stress(\mathbf{T}, f) = \sqrt{\frac{\sum_{i,j=1}^m (f(\delta_{ij}) - d_{ij})^2}{\sum_{i,j=1}^m d_{ij}^2}}$$

- $\delta_{ij} = d_{\mathcal{X}}(\mathbf{x}_i, \mathbf{x}_j)$, $d_{ij} = d_{\mathcal{T}}(\mathbf{t}_i, \mathbf{t}_j)$ – typically Euclidean,
- f is a proximity transformation function (e.g., identity, monotonic \rightarrow metric, ordinal),
- whole class of methods that differ in
 - the method for calculation of proximities δ ,
 - the parametrization of stress function,
 - the method that minimizes the stress function (e.g., gradient descent, Newton).

Geodesic distance

- What does Euclidean distance say about a non-linear manifold?
- We should better preserve **geodesic distance** between points
 - a minimum of the length of a path joining both points that is contained in the manifold,
 - these paths are called geodesics,
- as a result, we unfold the manifold.



Carreira-Perpinan: A Review of Dimension Reduction Techniques

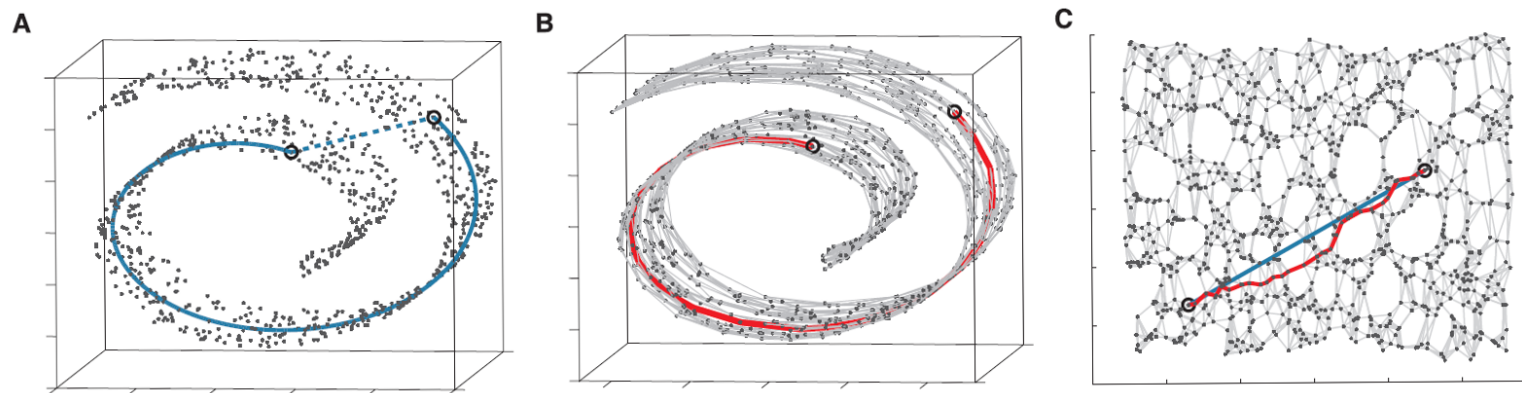
Isomap [Tenenbaum et al., 1998]

- Classical MDS with geodesic distance

(A) Euclidean vs geodesic distance to express intrinsic similarity in the input space,

(B) the neighborhood graph G ($K = 5$, $m = 1000$) and its shortest path (red) as an approximation to the true geodesic path,

(C) 2D embedding, which best preserves the shortest path distances in G , the straight lines in the embedding (blue) represent simpler and cleaner approximations to the true geodesic paths than do the corresponding graph paths.



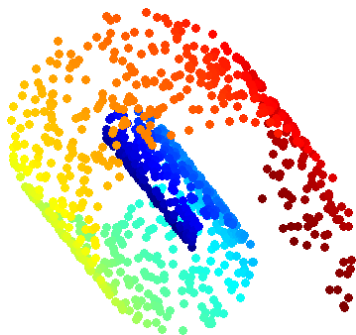
Tenenbaum et al.: A Global Geometric Framework for Nonlinear Dimensionality Reduction

Isomap [Tenenbaum et al., 1998]

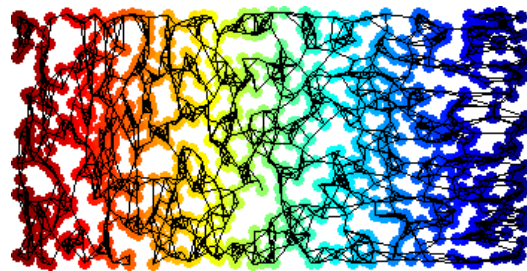
- The Isomap algorithm
 - determine the nearest neighbors
 - * all points in a fixed radius or K-nearest neighbors,
 - construct a neighborhood graph
 - * each point gets connected to its neighbors,
 - * edge length equals the Euclidean distance between the points,
 - compute the shortest paths between all pairs of points
 - * Floyd's or Dijkstra's algorithm ($\mathcal{O}(m^3)$ resp. $\mathcal{O}(Km^2 \log(m))$),
 - * could be time consuming and result in a large non-sparse matrix,
 - * use quantization to compress the graph,
 - construct a lower dimensional embedding
 - * use classical MDS.

Isomap [Tenenbaum et al., 1998]

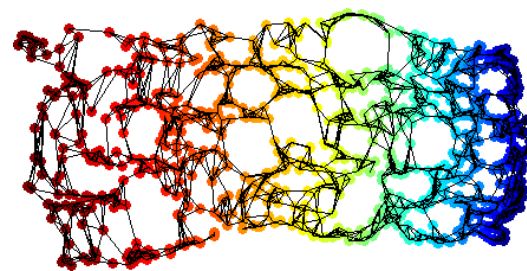
- Isomap has provable convergence guarantees,
- given the infinite (sufficient) input data, the method perfectly recovers the original distances
 - impractical concerning the next property,
 - otherwise, the geodesic distance can overestimate the true distance,
- cubic complexity in the number of objects can be too much
 - selection of M reference vectors can reduce the time to $\mathcal{O}(M^3)$,
 - we have to guarantee that the manifold is uniformly sampled,
- finding a proper value of K is not easy
 - too small/large K : insufficient graph/improper connections originate.



Swiss roll



ideal projection



Isomap outcome

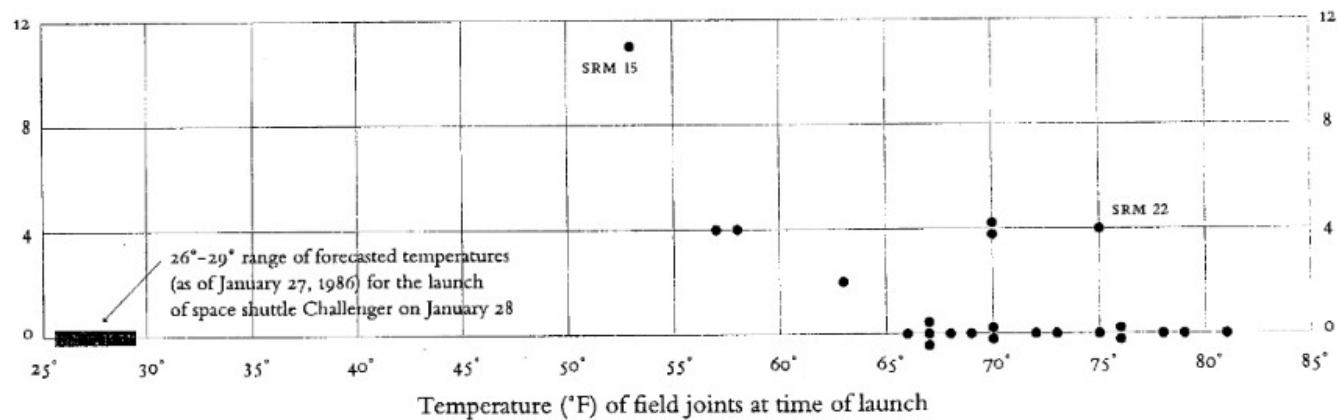


There are right and wrong ways to show data ...

- 1986 Challenger space shuttle disaster

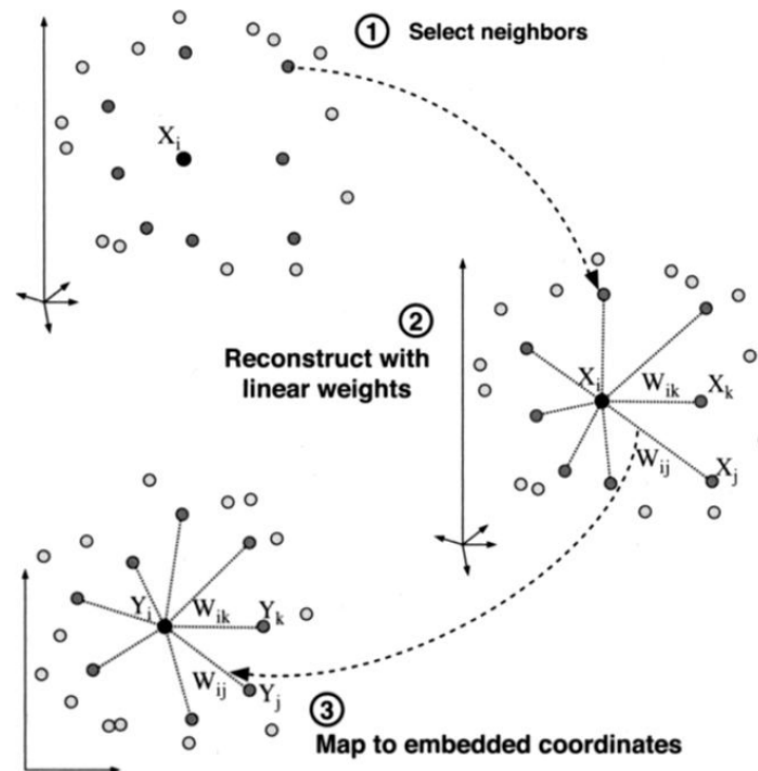


O-ring damage index, each launch



Locally Linear Embedding (LLE) [Roweis, Saul, 2000]

- Remember: manifold is a topological space that is **locally Euclidean**
 - we can locally fit (linear) models using Euclidean distance,
 - when compiled they create a global model.



Roweis, Saul: Nonlinear Dimensionality Reduction by LLE.

Locally Linear Embedding (LLE) [Roweis, Saul, 2000]

- The LLE algorithm

- each data point and its neighbors lie close to a locally linear patch of the manifold,
- each point can be written as a linear combination of its neighbors,
- m local models, the weights chosen to minimize the reconstruction error

$$\hat{\mathbf{x}}_i = \sum_{j \in \mathcal{N}(\mathbf{x}_i)} w_{ij} \mathbf{x}_j \text{ such that } \sum_{i=1}^m \|\mathbf{x}_i - \hat{\mathbf{x}}_i\|^2 \text{ is minimized and } \sum_{j \in \mathcal{N}(\mathbf{x}_i)} w_{ij} = 1$$

- the same weights should reconstruct the point in L dimensions
 - * the weights characterize the intrinsic geometric properties of each neighborhood,
- global embedding fits the positions \mathbf{t}_i in the low-dimensional space
 - * we minimize the embedding cost function, the weights are fixed

$$\hat{\mathbf{t}}_i = \sum_{j \in \mathcal{N}(\mathbf{x}_i)} w_{ij} \mathbf{t}_j \text{ such that } \sum_{i=1}^m \|\mathbf{t}_i - \hat{\mathbf{t}}_i\|^2 \text{ is minimized.}$$

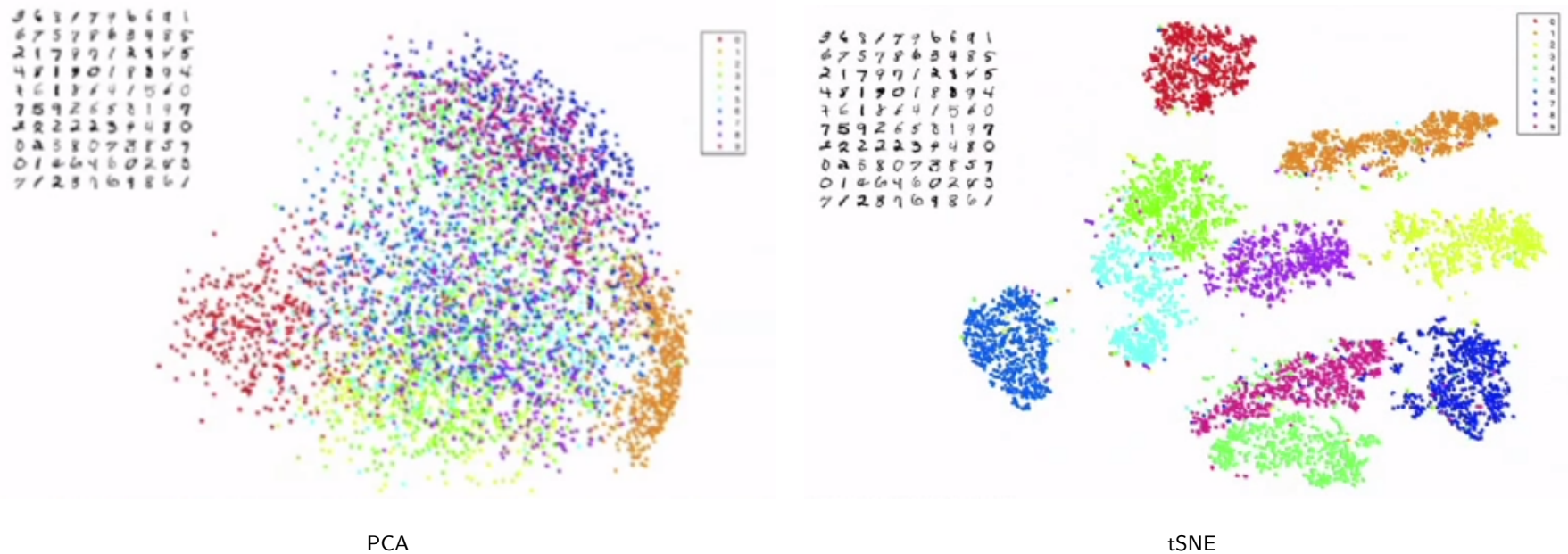
$$\text{subject to } \sum_{i=1}^m \mathbf{t}_i = \mathbf{0} \quad \sum_{i=1}^m \mathbf{t}_i \mathbf{t}_i^T = \mathbf{I}$$

Locally Linear Embedding (LLE) [Roweis, Saul, 2000]

- The ultimate case of piecewise linear modelling
 - approximation of the manifold by a combination of linear models,
 - in here, we have a linear model for each object,
- a special case of kernel PCA constructing a data-dependent kernel matrix
 - for some problems it is difficult to find a kernel for kernel PCA,
- advantages
 - efficient for large datasets, optimization does not involve local minima,
 - single parameter to tune (K – the number of nearest neighbors per object),
 - invariant to scaling, rotation and translation,
- disadvantages
 - improper for representing future data,
 - can be unstable in sparse areas of the input space,
 - tends to collapse a lot of instances near the origin of \mathcal{T} .

t-Distributed Stochastic Neighbor Embedding (t-SNE)

- distance preserving visualization technique [van der Maaten, 2008] (like e.g., MDS),
- puts emphasis on preserving small pairwise distances between objects,
- large distances allowed to be modelled as being larger,
- as a result, two essential characteristics
 - the local data structures retained,
 - ability to reveal global structure such as the presence of clusters at several scales.



van der Maaten: Visualizing Data Using t-SNE, GoogleTechTalks

t-Distributed Stochastic Neighbor Embedding (t-SNE)

- The key ideas are in the design of the stress function driving gradient descent search
 - convert Euclidean distances in both spaces into joint probabilities,
 - p_{ij} in the original space \mathcal{X} and q_{ij} in the reduced space \mathcal{T} ,
 - minimize their Kullback-Leibler divergence

$$S = KL(P||Q) = \sum_i \sum_{j \neq i} p_{ij} \log \frac{p_{ij}}{q_{ij}}$$

- The first “trick” lies in asymmetry of KL divergence
 - large p_{ij} modeled by small q_{ij} \rightarrow big penalty!
 - small p_{ij} modeled by large q_{ij} \rightarrow small penalty!
 - tends to preserve large p_{ij} 's and thus small distances in the original space.

t-Distributed Stochastic Neighbor Embedding (t-SNE)

- The other “tricks” consist in definition of p_{ij} and q_{ij}
 - the empirical probability that an object j is a neighbor of an object i

$$p_{j|i} = \frac{\exp(-\|\mathbf{x}_i - \mathbf{x}_j\|^2/2\sigma_i)}{\sum_{k \neq i} \exp(-\|\mathbf{x}_i - \mathbf{x}_k\|^2/2\sigma_i)}$$

- i.e., it is normally distributed wrt their distance (and decreases quickly with it),
- σ_i is the kernel bandwidth,
- σ_i is locally adjusted so that a fixed number of objects falls in bandwidth around the mean,
- the symmetric joint probability

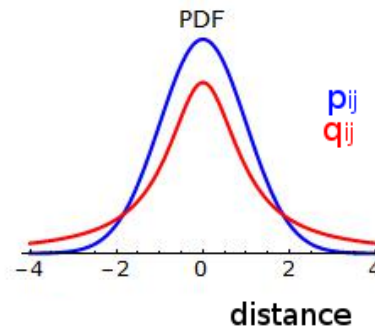
$$p_{ij} = \frac{p_{j|i} + p_{i|j}}{2m}$$

t-Distributed Stochastic Neighbor Embedding (t-SNE)

- In the reduced space \mathcal{T} we permit higher probabilities for larger distances,
 - the normal distribution used in p_{ij} turns into the heavy-tailed t-distribution in q_{ij}

$$q_{ij} = \frac{(1 + \|\mathbf{t}_i - \mathbf{t}_j\|^2)^{-1}}{\sum_{k \neq l} (1 + \|\mathbf{t}_k - \mathbf{t}_l\|^2)^{-1}}$$

- in KL divergence, p_{ij} and q_{ij} should agree as much as possible, but they may map to different distances in both the spaces,
- as a result, a moderate distance in the high-dimensional space can be faithfully modeled by a much larger distance in the map,
- the reduced map gets insensitive to distant points (they can be placed to many places without big changes in q_{ij}).



Tails in normal and student distributions.

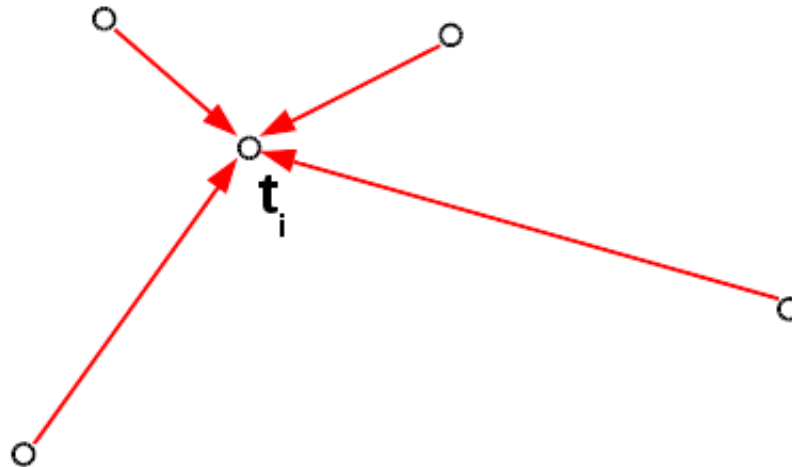
t-Distributed Stochastic Neighbor Embedding (t-SNE)

- The overall picture

- the gradient descent gradually minimizes the stress function for the individual objects

$$\frac{\partial S}{\partial \mathbf{t}_i} \propto \sum_{j \neq i} (p_{ij} - q_{ij})(1 + \|\mathbf{t}_i - \mathbf{t}_j\|^2)^{-1}(\mathbf{t}_i - \mathbf{t}_j)$$

- all the other objects get connected via springs that are either stretched or compressed,
- the resultant summed force tells us where to move the point in every gradient update.



Summary – dimensionality reduction, manifold learning

- Difficult problem namely for the curse of dimensionality
 - huge sample sizes needed to guarantee reasonable parameter estimates, non-empty neighborhoods, etc.,
 - the intrinsic dimensionality estimation is not reliable,
- strong assumptions greatly simplify the task,
- the key role of PCA has not been undermined by any non-linear method yet
 - they work for well-sampled smooth manifolds, but not necessarily for real data,
 - besides the curse of dimensionality, the problems could be caused by insufficiency of objective functions or numerical problems during their optimization,
- there is a large pool of non-linear reduction methods,
- the key properties are effectivity and efficiency including convergence,
- other issues
 - setting hyperparameters?
 - implicit/explicit definition of \mathbf{F} and \mathbf{f} ,
 - additivity – can I drop a coordinate from L mapping to obtain $L - 1$ mapping?

Recommended reading, lecture resources

:: Reading

- Jonathon Shlens: **A Tutorial on Principal Component Analysis.**
 - Google research, 2014,
 - <http://arxiv.org/pdf/1404.1100.pdf>,
- Scholkopf, Smola, Muller: **Kernel Principal Component Analysis.**
 - Artificial Neural Networks, 1997,
 - <http://link.springer.com/chapter/10.1007/BFb0020217>,
- Carreira-Perpinan: **A Review of Dimension Reduction Techniques.**
 - Tech. report, University of Sheffield, 1997,
 - <http://www.pca.narod.ru/DimensionReductionBrifReview.pdf>,
- van der Maaten, Hinton: **Visualizing High-Dimensional Data Using t-SNE.**
 - Journal of Machine Learning Research 9(Nov):2579-2605, 2008,
 - https://lvdmaaten.github.io/publications/papers/JMLR_2008.pdf.