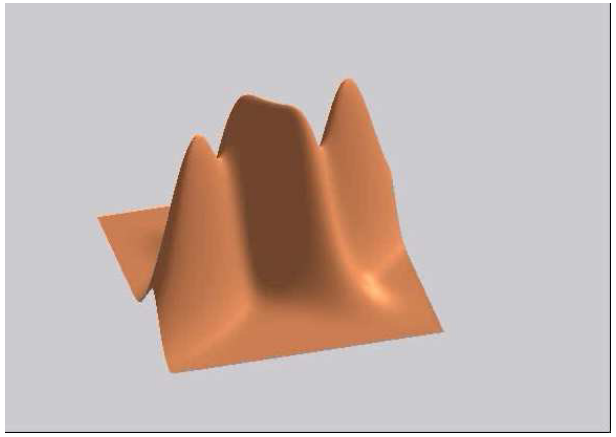
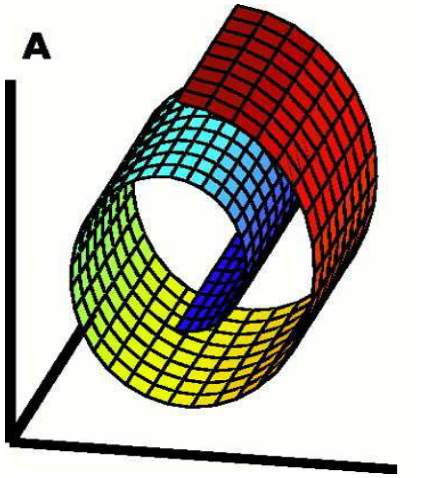
Types of Structure in High Dimension

- Clumps
  - Clustering
  - Density Estimation



- Low Dimensional Manifolds
  - Linear
  - NonLinear



Dimensionality Reduction

- Data representation

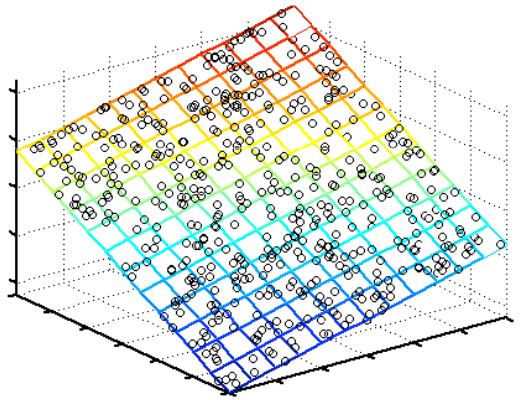
  Inputs are real-valued vectors in a high dimensional space.

- Linear structure

  Does the data live in a low dimensional subspace?

- Nonlinear structure

  Does the data live on a low dimensional submanifold?



Dimensionality Reduction

- Question

  How can we detect low dimensional structure in high dimensional data?

- Applications
  - Digital image and speech processing
  - Analysis of neuronal populations
  - Gene expression microarray data
  - Visualization of large networks

Notations

- Inputs (**high dimensional**)

  $\boldsymbol{x_1, x_2, \ldots, x_n}$ points in R^D

- Outputs **(low dimensional)**

  $\boldsymbol{y_1, y_2, \ldots, y_n}$ points in R^d ($d << D$)

- Goals

  Nearby points remain nearby.

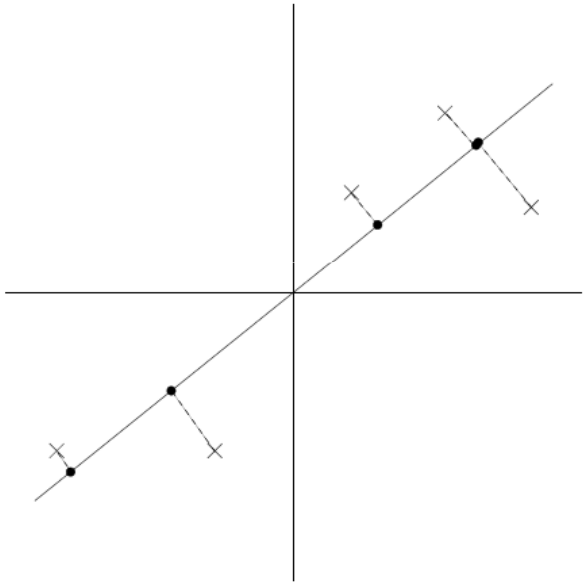
  Distant points remain distant.

Linear Methods

- PCA
- MDS

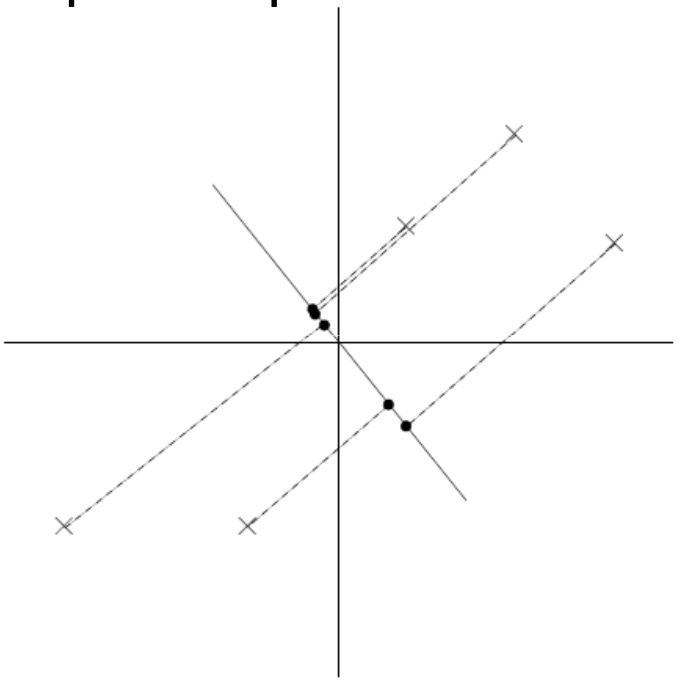
Principle Component Analysis

good representation



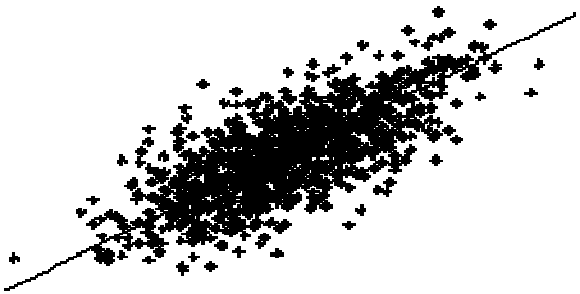
the projected data has a fairly large variance, and the points tend to be far from zero.

poor representation

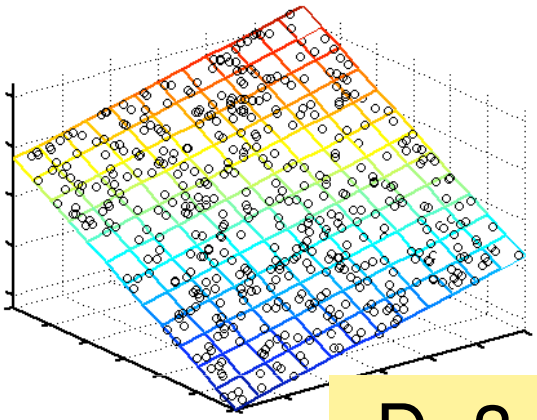


the projections have a significantly smaller variance, and are much closer to the origin.

Principle Component Analysis



D=2, d=1



D=3, d=2

- Seek most accurate data representation in a lower dimensional space.
- The good direction/subspace to use for projection lies in the direction of largest variance.

Maximum Variance Subspace

- Assume inputs are centered: $\sum_i x_i = 0$
- Given a unit vector u and a point x, the length of the projection of x onto u is given by $x^T u$
- Maximize projected variance:

$$\text{var}(y) = \frac{1}{n}\sum_i \left(x_i^T u\right)^2 = \frac{1}{n}\sum_i u^T x_i x_i^T u$$

$$= u^T \left(\frac{1}{n}\sum_i x_i x_i^T\right) u$$

1D Subspace

- Maximizing $u^T C u$ subject to $\|\mathbf{u}\| = 1$

  where $C = n^{-1} \sum_i x_i x_i^T$ is the empirical

  covariance matrix of the data,

  gives the principle eigenvector of C.

d-dimensional Subspace

- to project the data into a d-dimensional subspace (d <<D), we should choose

  $u_1,\ldots,u_d$ to be the top d eigenvectors of C.

- $u_1,\ldots,u_d$ now form a new, orthogonal basis for the data.
- The low dimensional representation of x is given by

$$y_i = \begin{bmatrix} u_1^T x_i \\ u_2^T x_i \\ \vdots \\ u_k^T x_i \end{bmatrix} \in \Re^d .$$

Interpreting PCA

- Eigenvectors:

  principal axes of maximum variance subspace.

- Eigenvalues:

  variance of projected inputs along principle axes.

- Estimated dimensionality:

  number of significant (nonnegative) eigenvalues.

PCA summary

Input: $z_i \in R^D,\ i = 1,..,n$ Output: $y_i \in R^d,\ i = 1,..,n$

1. Subtract sample mean from the data

$$x_i = z_i - \hat{\mu}, \quad \hat{\mu} = 1/n \sum_i z_i$$

2. Compute the covariance matrix

$$C = 1/n \sum_{i=1}^{n} x_i x_i^t$$

3. Compute eigenvectors $\boldsymbol{e_1}, \boldsymbol{e_2}, \ldots, \boldsymbol{e_d}$ corresponding to the $\boldsymbol{d}$ largest eigenvalues of C (d<<D).

4. The desired y is

$$y = P^t x, \ P = [e_1, ..., e_d]$$

Equivalence

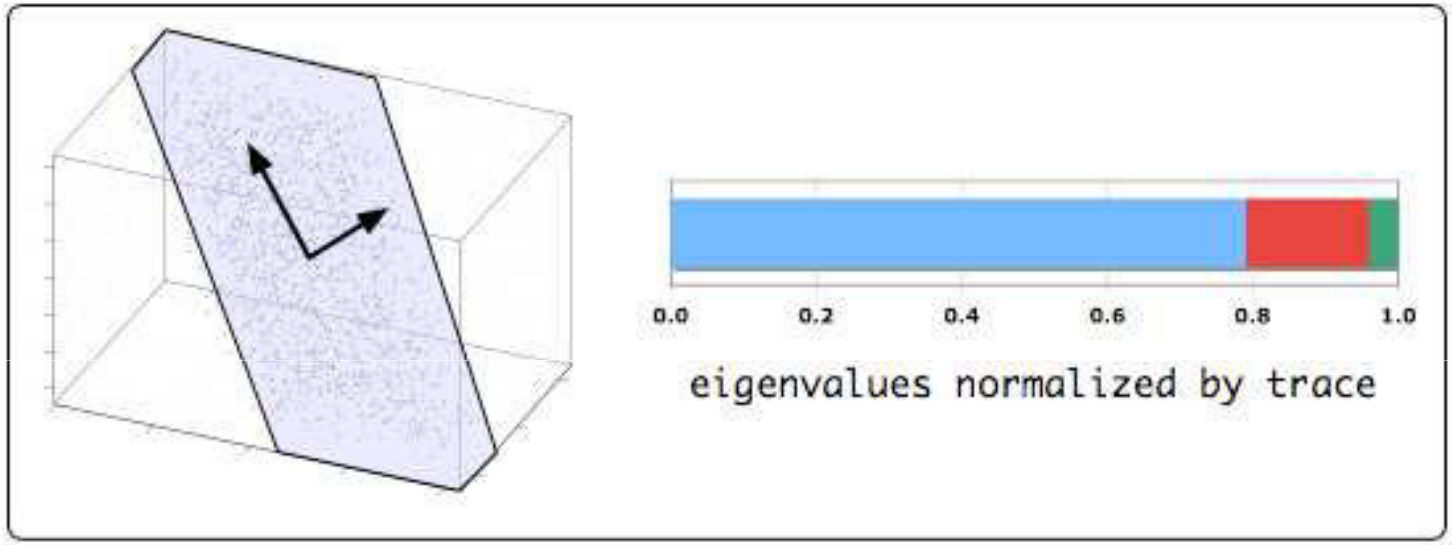
- PCA finds the directions that have the most variance.

$$\mathrm{var}(y) = \frac{1}{n}\sum_i \left\| P^T x_i \right\|^2$$

- Same result can be obtained by minimizing the squared reconstruction error.

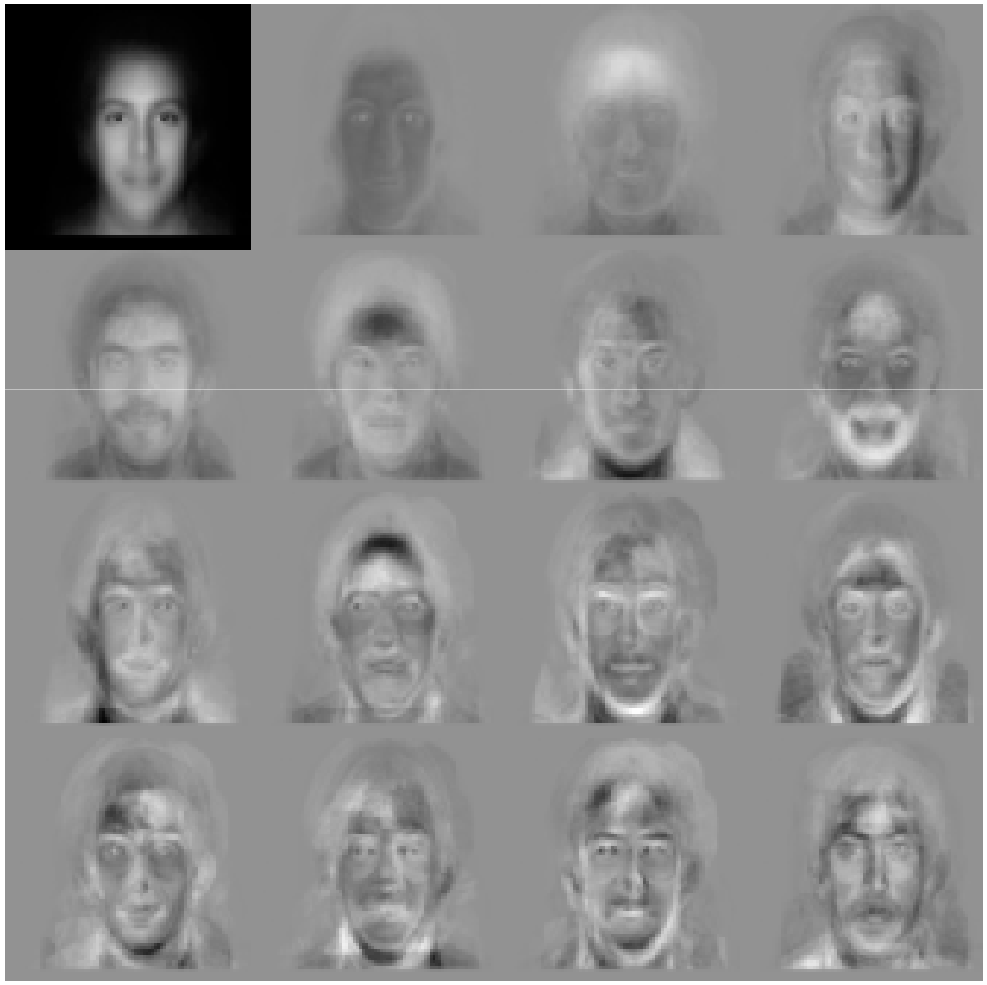
$$err(y) = \frac{1}{n}\sum_i \left\| x_i - PP^T x_i \right\|^2$$

Example of PCA



Eigenvectors and eigenvalues of covariance matrix for *n=1600* inputs in *d=3 dimensions.*

Example: faces

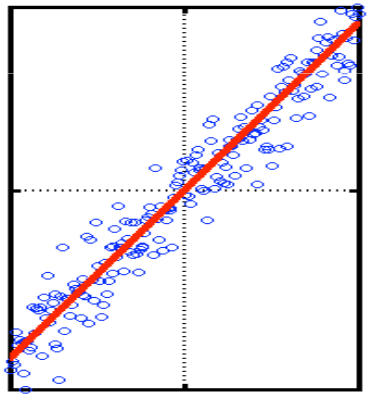


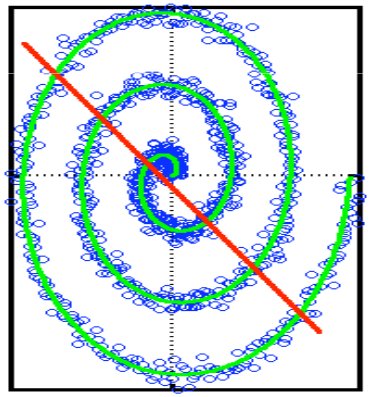
Eigenfaces from 7562 Images:
top left image
is linear
combination
of the rest.

Sirovich & Kirby (1987)
Turk & Pentland (1991)

Properties of PCA

- Strengths:
  - Eigenvector method
  - No tuning parameters
  - Non-iterative
  - No local optima





- Weaknesses:
  - Limited to second order statistics
  - Limited to linear projections

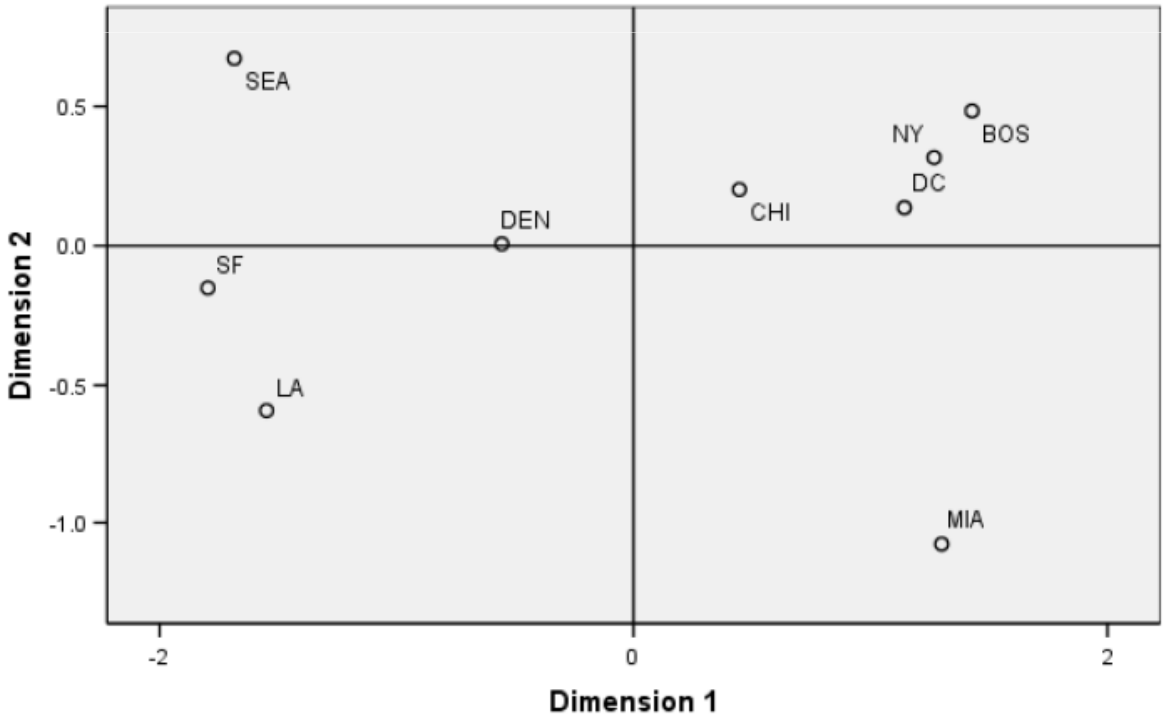
Multidimensional Scaling (MDS)

- MDS attempts to preserve pairwise distances.

- Attempts to construct a configuration of n points in Euclidian space by using the information about the distances between the n patterns.

Example : Distances between US Cities

	BOS	CHI	DC	DEN	LA	MIA	NY	SEA	SF
BOS	0	963	429	1,949	2,979	1,504	206	2,976	3,095
CHI	963	0	671	996	2,054	1,329	802	2,013	2,142
DC	429	671	0	1,616	2,631	1,075	233	2,684	2,799
DEN	1,949	996	1,616	0	1,059	2,037	1,771	1,307	1,235
LA	2,979	2,054	2,631	1,059	0	2,687	2,786	1,131	379
MIA	1,504	1,329	1,075	2,037	2,687	0	1,308	3,273	3,053
NY	206	802	233	1,771	2,786	1,308	0	2,815	2,934
SEA	2,976	2,013	2,684	1,307	1,131	3,273	2,815	0	808
SF	3,095	2,142	2,799	1,235	379	3,053	2,934	808	0



Multidimensional Scaling (MDS)

- A $n \times n$ matrix $\mathcal{D}$ is called a distance or affinity matrix if it is symmetric, $\mathtt{d}_{ii} = 0$, and $\mathtt{d}_{ij} > 0, \quad i \neq j$.

- Given a distance matrix $\mathcal{D}^{(X)}$, MDS attempts to find n data points $y_1, ..., y_n$ in d dimensions, such that if $d_{ij}^{(Y)}$ denotes the Euclidean distance between y_i and y_j, then $\mathcal{D}^Y$ is similar to $\mathcal{D}^{(X)}$.

Metric MDS

- Metric MDS minimizes

$$\min_{Y} \sum_{i=1}^{n} \sum_{j=1}^{n} (d_{ij}^{(X)} - d_{ij}^{(Y)})^2$$

where

$$d_{ij}^{(X)} = \left\| x_i - x_j \right\| \quad \text{and} \quad d_{ij}^{(Y)} = \left\| y_i - y_j \right\|.$$

Metric MDS

- The distance matrix $D^{(X)}$ can be converted to a Gram matrix K by

$$K = -\frac{1}{2} H (D^{(X)})^2 H$$

where $H = I - \frac{1}{n} e e^T$ and e is the vector of ones.

Metric MDS

- K is *p.s.d*, thus it can be written as $K = X^T X$
- $\min_Y \sum_{i=1}^{n} \sum_{j=1}^{n} (d_{ij}^{(X)} - d_{ij}^{(Y)})^2$ is equivalent to

$$\min_Y \sum_{i=1}^{n} \sum_{j=1}^{n} (x_i^T x_j - y_i^T y_j)^2$$

- The norm can be converted to a trace:

$$\min_Y Tr\left(X^T X - Y^T Y\right)^2$$

Non-Metric MDS

- Transform pairwise distances: $\delta_{ij} \rightarrow g(\delta_{ij})$
  - Transformation: nonlinear, but monotonic.
  - Preserves rank order of distances.

- Find vectors y_i such that $\left\| y_i - y_j \right\| \approx g(\delta_{ij})$

$$Cost = \min_{y} \sum_{ij} \left(g(\delta_{ij}) - \left\| y_i - y_j \right\| \right)^2$$

Non-Metric MDS

- Possible objective function:

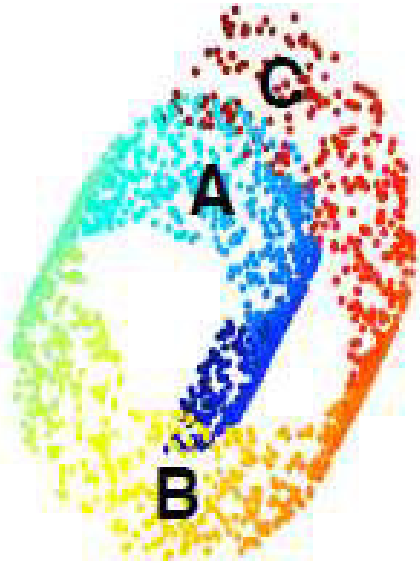
$$Cost = \sum_{i\ j} \left(\frac{\| \mathbf{x}_i - \mathbf{x}_j \| - \| \mathbf{y}_i - \mathbf{y}_j \|}{\| \mathbf{x}_i - \mathbf{x}_j \|} \right)^2$$

Properties of non-metric MDS

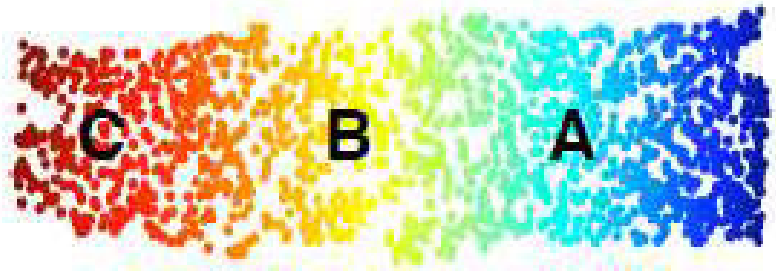
- Strengths
  - Relaxes distance constraints.
  - Yields nonlinear embeddings.
- Weaknesses
  - Highly nonlinear, iterative optimization with local minima.
  - Unclear how to choose distance transformation.

Non-metric MDS for manifolds?

Rank ordering of Euclidean distances is NOT preserved in “manifold learning”.

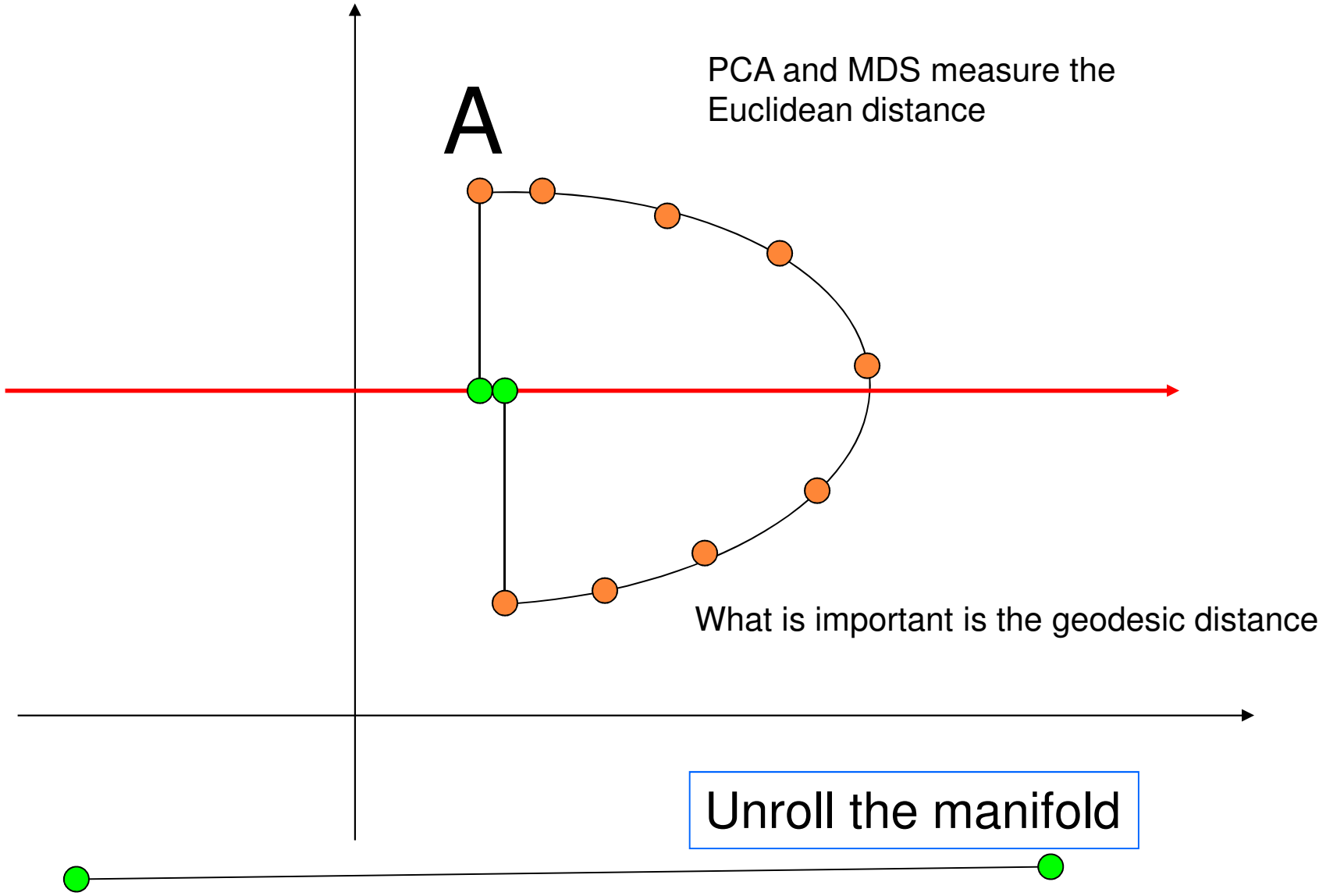


$d(A,C) < d(A,B)$

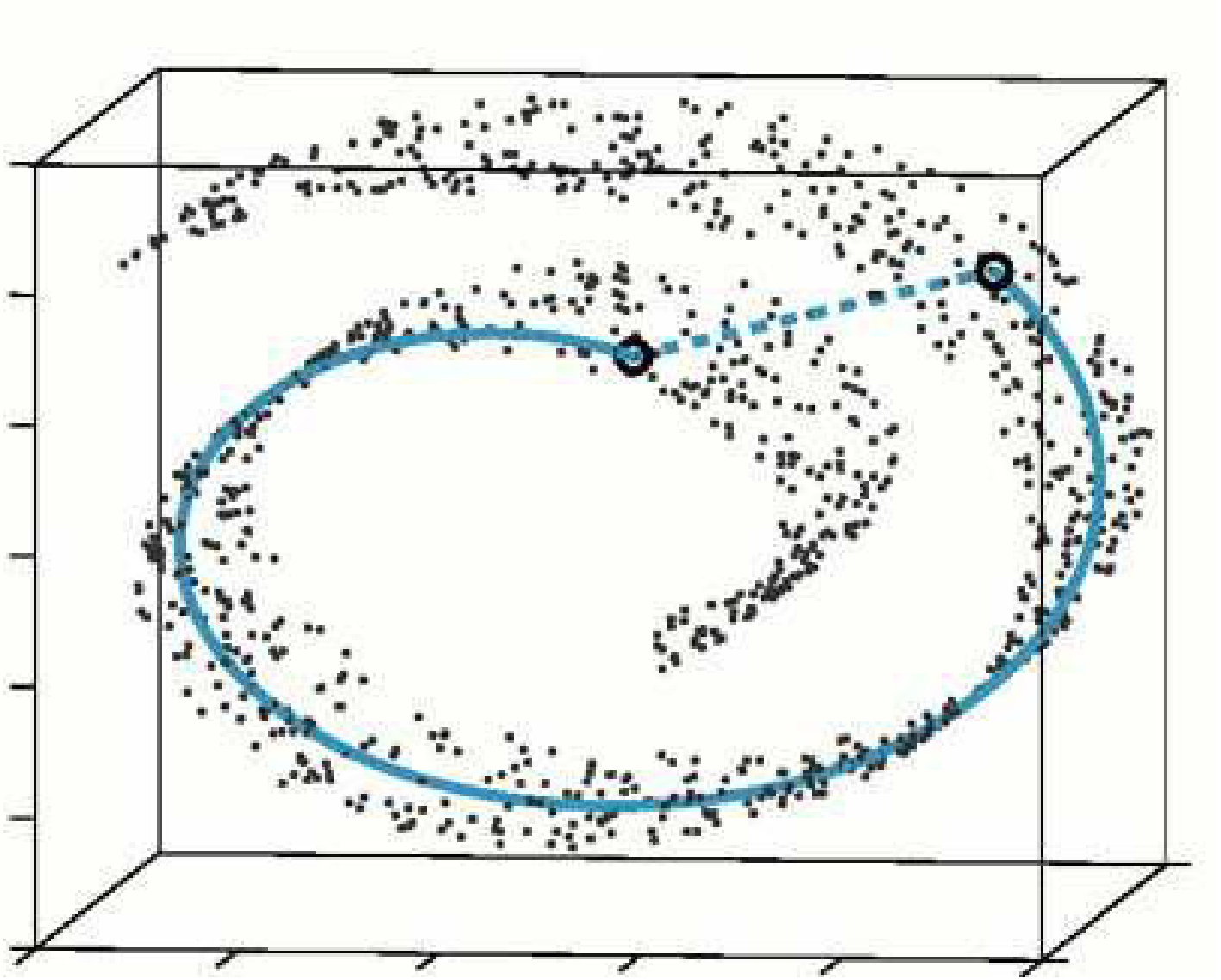


$d(A,C) > d(A,B)$

Nonlinear Manifolds



To preserve structure preserve the geodesic distance and not the euclidean distance.



Graph-Based Methods

- Tenenbaum et.al's Isomap Algorithm
  - Global approach.

    Preserves global pairwise distances.

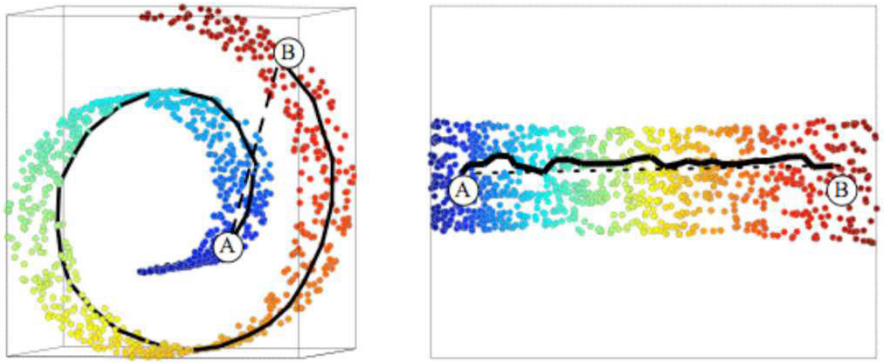
- Roweis and Saul's Locally Linear Embedding Algorithm
  - Local approach

    Nearby points should map nearby

- Belkin and Niyogi Laplacian Eigenmaps Algorithm
  - Local approach
  - minimizes approximately the same value as LLE

Isomap - Key Idea:

- Use **geodesic** instead of Euclidean distances in MDS.
  - For neighboring points Euclidean distance is a good approximation to the geodesic distance.
  - For distant points estimate the distance by a series of short hops between neighboring points. Find shortest paths in a graph with edges connecting neighboring data points.



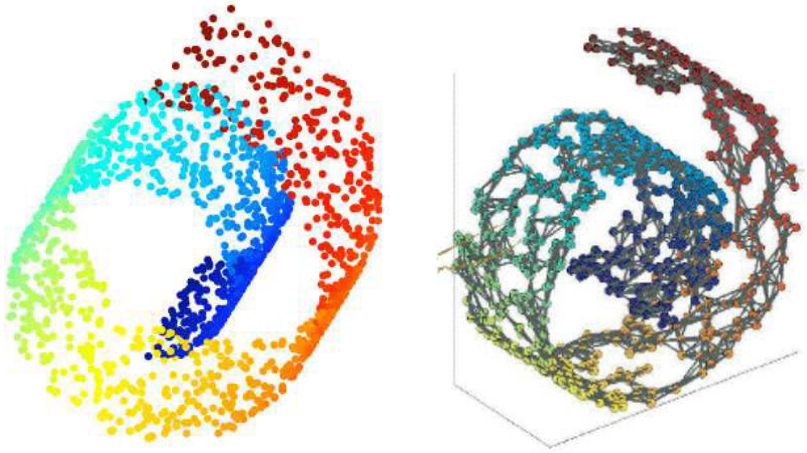
Step 1. Build adjacency graph.

- Adjacency graph

  Vertices represent inputs. Undirected edges connect neighbours.

- Neighbourhood selection

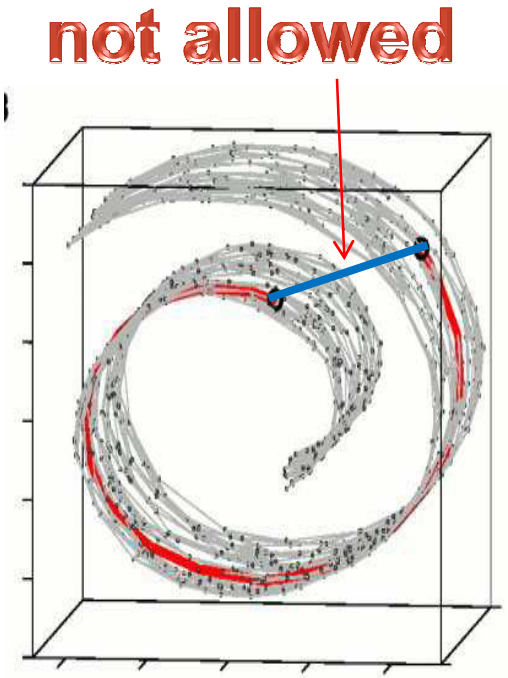
  Many options: k-nearest neighbours, inputs within radius r, prior knowledge.



Graph is discretized approximation of submanifold.

Building the graph

- Computation
  - kNN scales naively as $O(n^2 D)$
  - Faster methods exploit data structures.
- Assumptions
  1. Graph is connected.
  2. Neighbourhoods on graph reflect neighbourhoods on manifold.



Step 2. Estimate geodesics

- Dynamic programming
  - Weight edges by local distances.
  - Compute shortest paths through graph.
- Geodesic distances
  - Estimate by lengths of shortest paths:

    denser sampling = better estimates.
- Computation
  - Djikstra's algorithm for shortest paths $O(n^2 \log n + n^2 k)$.

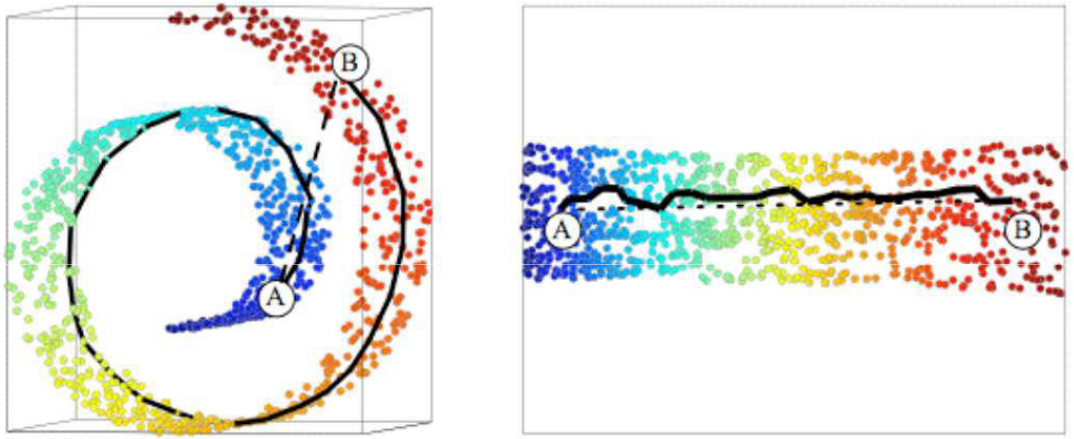
Step 3. Metric MDS

- Embedding
  - Top d eigenvectors of Gram matrix yield embedding.
- Dimensionality
  - Number of significant eigenvalues yield estimate of dimensionality.
- Computation
  - Top d eigenvectors can be computed in $O(n^2d)$.

Summary

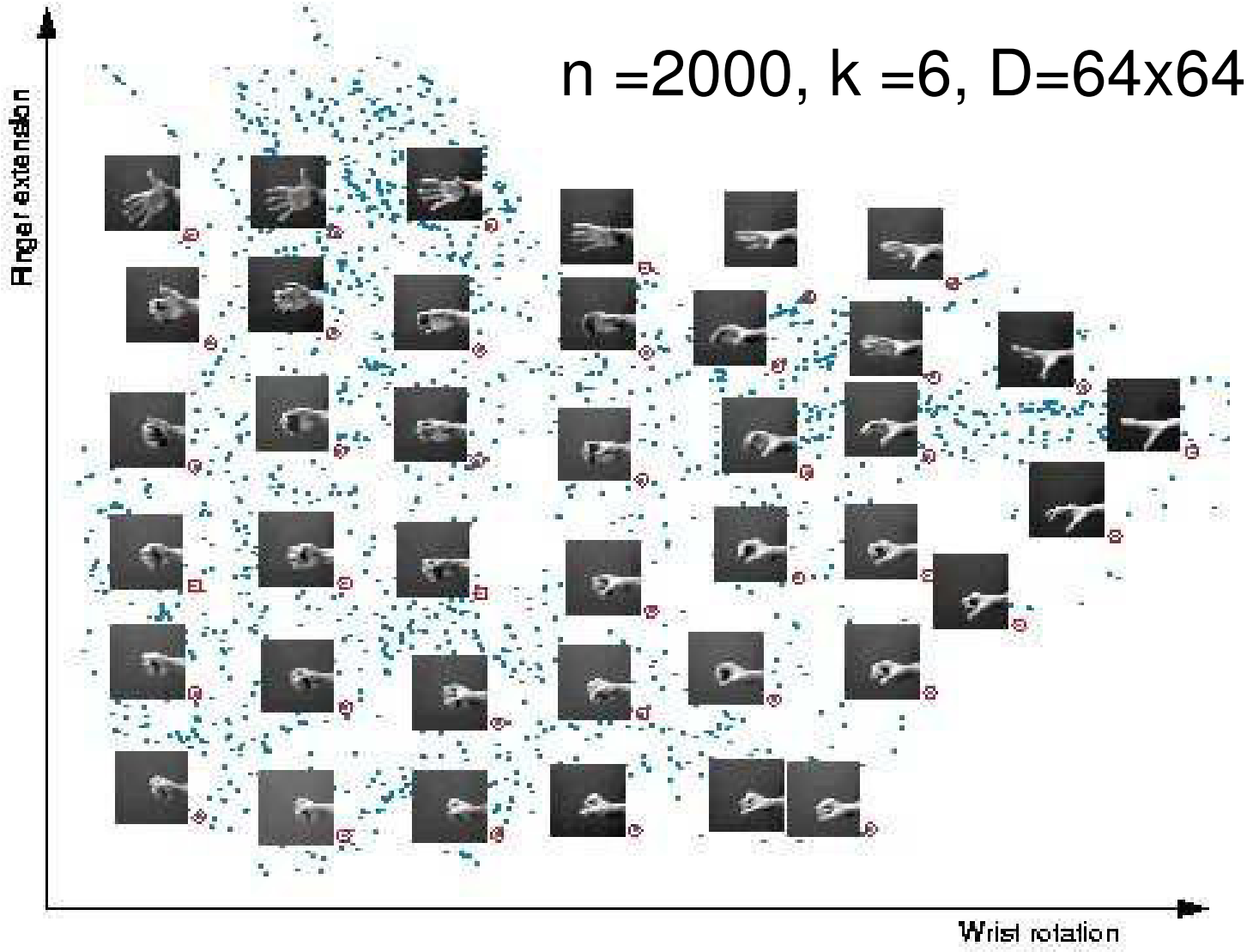
- Algorithm
  1. k nearest neighbours
  2. shortest paths through graph
  3. MDS on geodesic distances

Swiss Roll

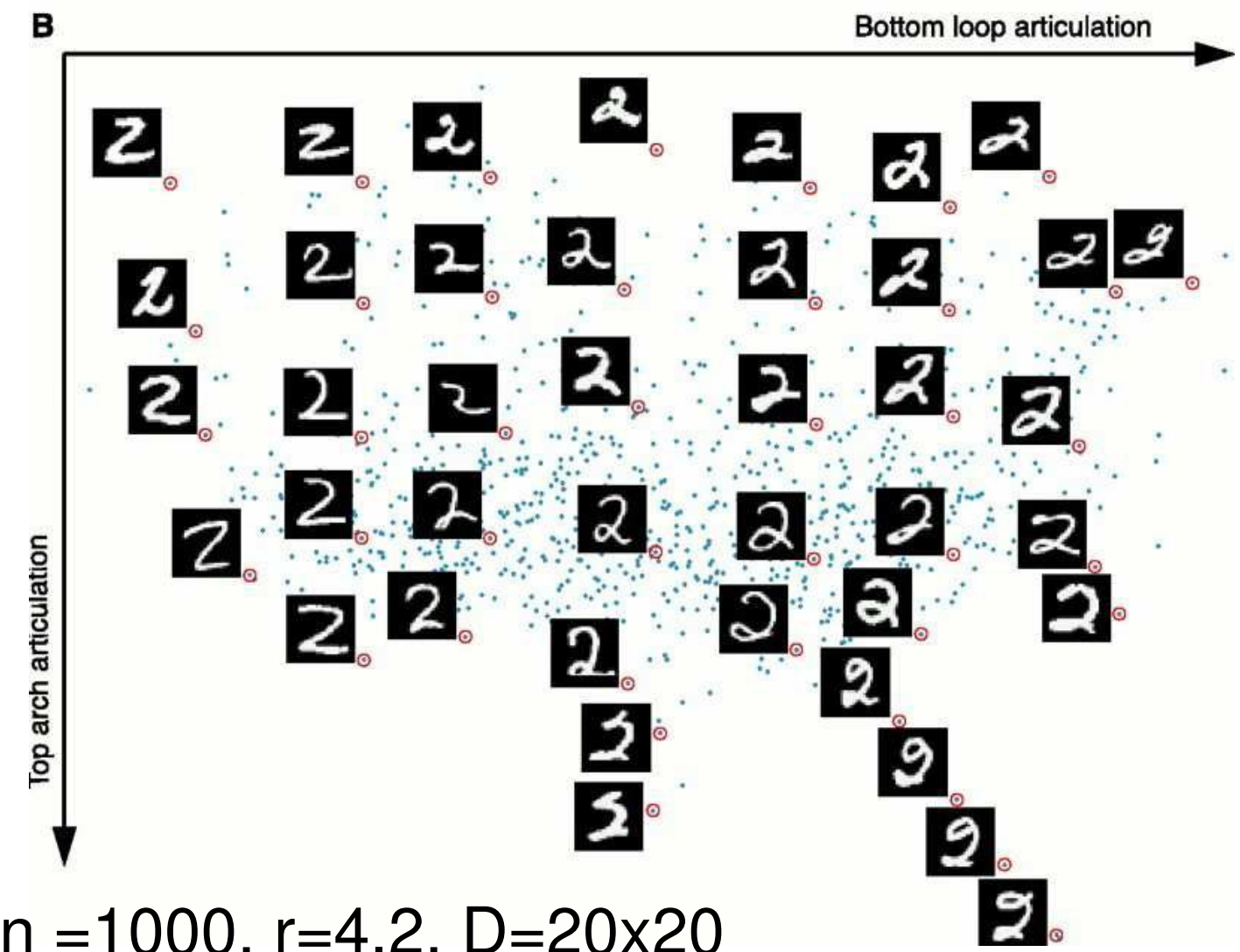


n (points) =1024
k (neighbors) =12

Isomap: Two-dimensional embedding of hand images (from Josh. Tenenbaum, Vin de Silva, John Langford 2000)



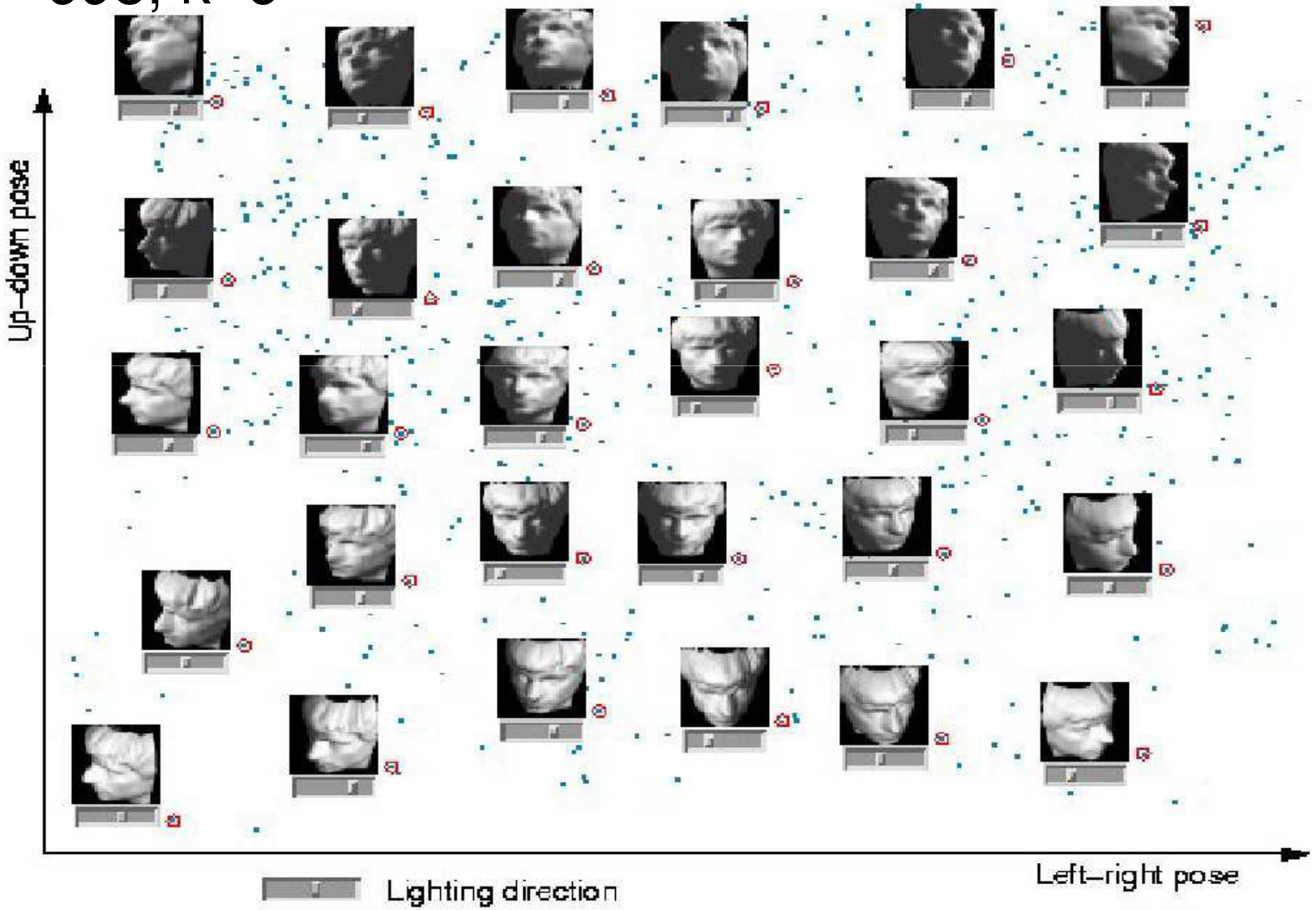
Isomap: two-dimensional embedding of hand-written ‘2’ (from Josh. Tenenbaum, Vin de Silva, John Langford 2000)



n =1000, r=4.2, D=20x20

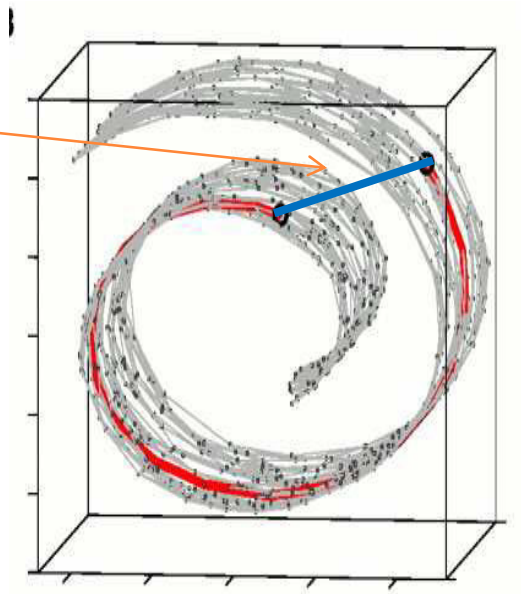
Isomap: three-dimensional embedding of faces (from Josh. Tenenbaum, Vin de Silva, John Langford 2000)

n =698, k=6



Properties of Isomap

- Strengths :
  - Preserves the global data structure
  - Performs global optimization
  - Non-parametric (Only heuristic is neighbourhood size)
- Weaknesses :
  - Sensitive to “shortcuts”
  - Very slow



Spectral Methods

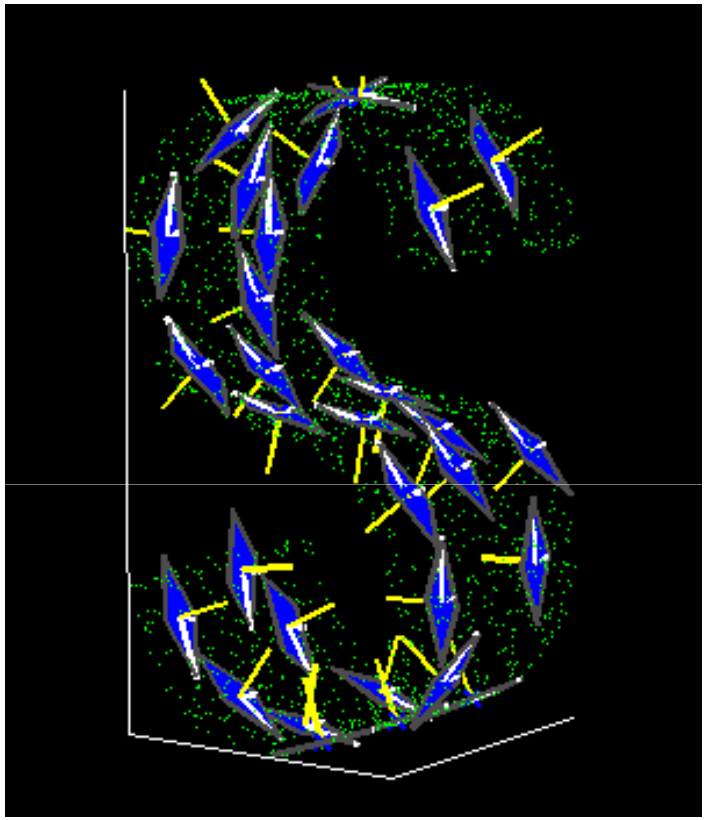
- Common framework
  1. Derive sparse graph from *kNN.*
  2. Derive matrix from graph weights.
  3. Derive embedding from eigenvectors.

- Varied solutions

  Algorithms differ in step 2. Types of optimization: shortest paths, least squares fits, semidefinite programming.

Locally Linear Embedding (LLE)

- Assume that data lies on a manifold: each sample and its neighbors lie on approximately linear subspace
- Idea:
  1. Approximate data by a set of linear patches
  2. Glue these patches together on a low dimensional subspace s.t. neighborhood relationships between patches are preserved.



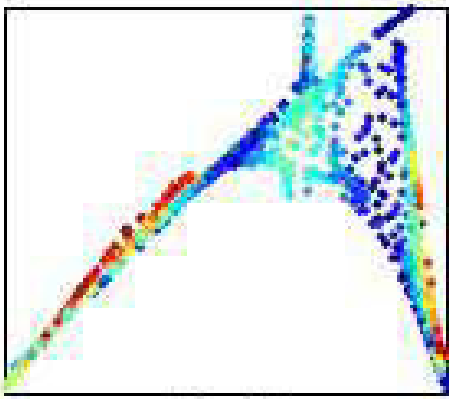
Algorithm: http://cs.nyu.edu/~roweis/lle/algorithm.html

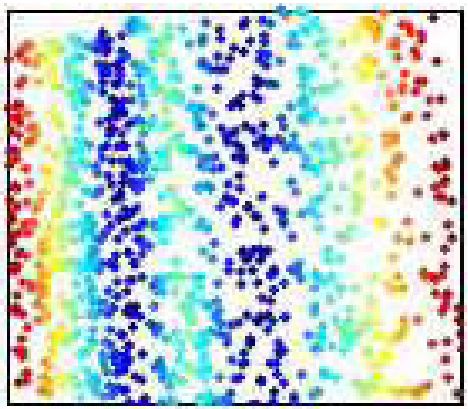
LLE at glance

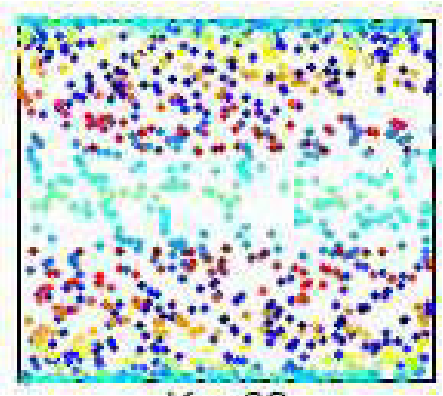
- Steps
  1. Nearest neighbour search.
  2. Least squares fits.
  3. Sparse eigenvalue problem.
- Properties
  - Obtains highly nonlinear embeddings.
  - Not prone to local minima.
  - Sparse graphs yield sparse problems.

Step 1. Nearest neighbours search

Effect of Neighbourhood Size



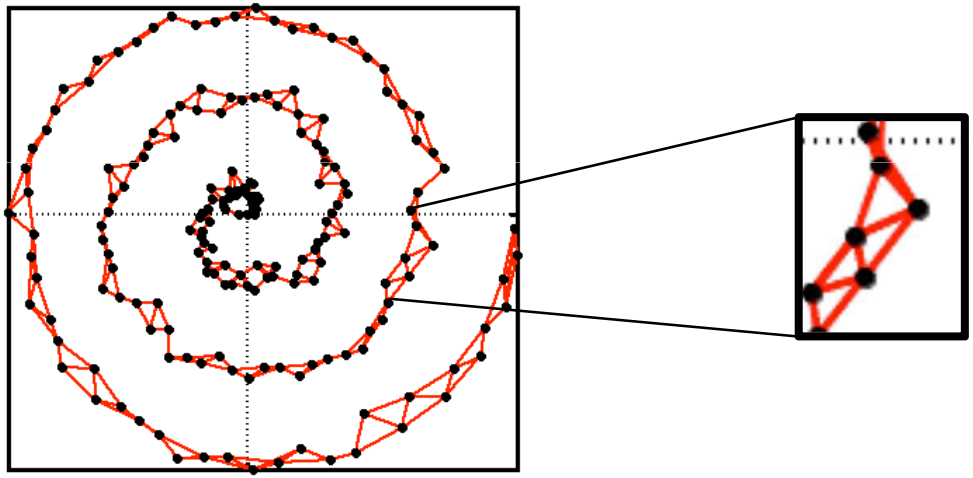




✗ ✓ ✗

Step 2. Compute weights

- Characterize local geometry of each neighbourhood by weights Wij.



- Compute weights by reconstructing each input (linearly) from neighbours.

Linear reconstructions

- Local linearity
  - Assume neighbours lie on locally linear patches of a low dimensional manifold.
- Minimize reconstruction error
  - Each point can be written as a linear combination of its neighbors.
  - The weights chosen to minimize the reconstruction error:

$$\min_{W} \sum_{i} \left| x_i - \sum_{j} W_{ij} x_j \right|^2$$

Least squares fits (Computing W_{ij})

- Local reconstructions
  - Choose weights to minimize: $\Phi(W) = \sum_i \left| x_i - \sum_j W_{ij} x_j \right|^2$
- Constraints
  - Set $W_{ij} = 0$ if x_j is not a neighbor of x_i
  - Weights must sum to one: $\sum_j W_{ij} = 1$

invariance to translation

- Local invariance
  - Optimal weights W_{ij} are invariant to rotation, translation, and scaling.

Step 3. Finding the Embedding

- Low dimensional representation

  Map inputs to outputs: $x_i \in R^D \rightarrow y_i \in R^d$

- Minimize reconstruction errors

  Optimize outputs for fixed weights:

$$\Psi(y) = \sum_i \left| y_i - \sum_j W_{ij} y_j \right|^2$$

- Constraints:
  - Center outputs on origin $\sum_i y_i = 0$
  - Impose unit covariance matrix $\frac{1}{N} \sum_i y_i y_i = I_d$

Minimization

- Quadratic form:

$$\Psi(y) = \sum_{ij} M_{ij}\left(y_i \cdot y_j\right)$$

$$M_{ij} = \delta_{ij} - W_{ij} - W_{ji} + \sum_k W_{ki} W_{kj},$$

$$\delta_{ij} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}$$

It can be shown that

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

Sparse eigenvalue problem

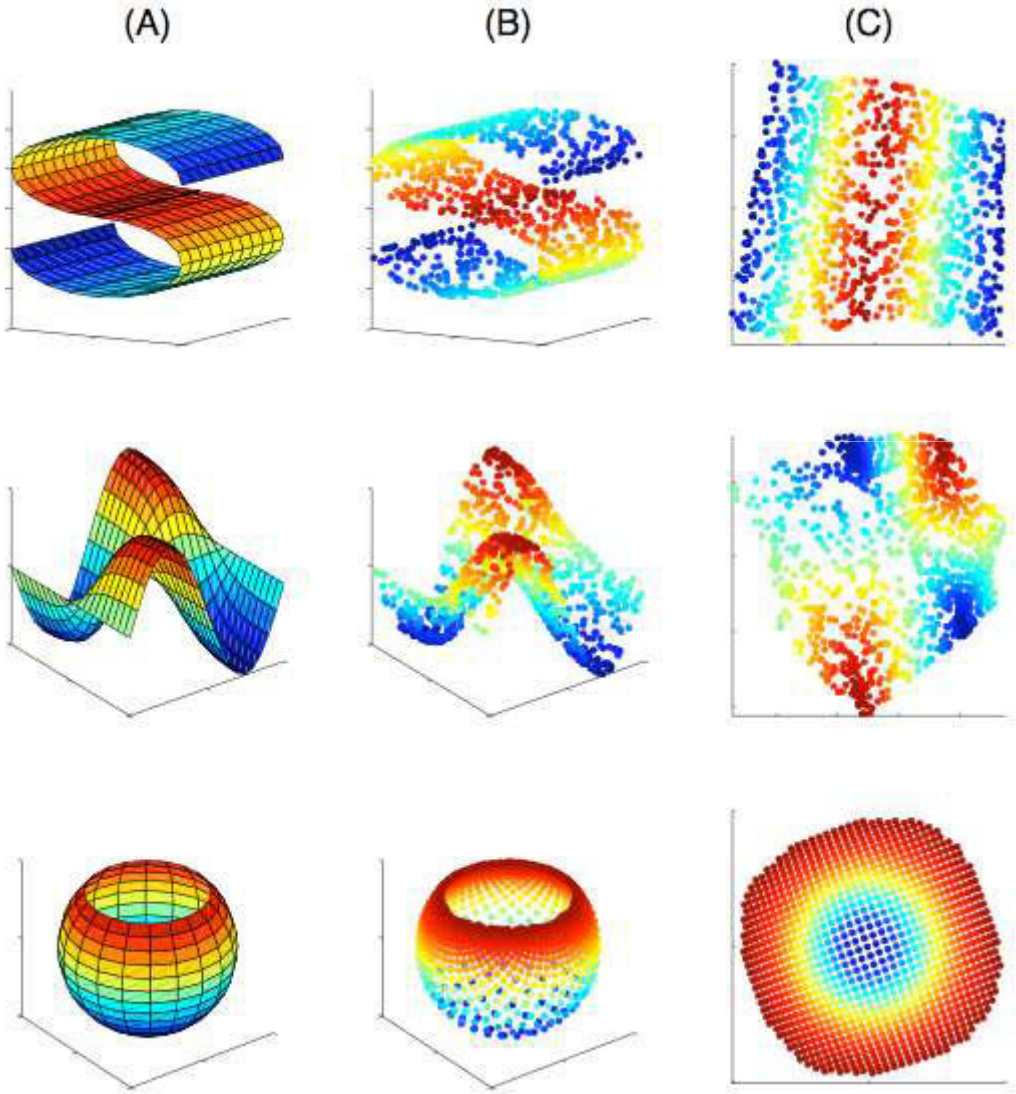
- Optimal embedding

  given by bottom d+1 eigenvectors, corresponding to the d+1 smallest eigenvalues (Rayleigh-Ritz theorem).

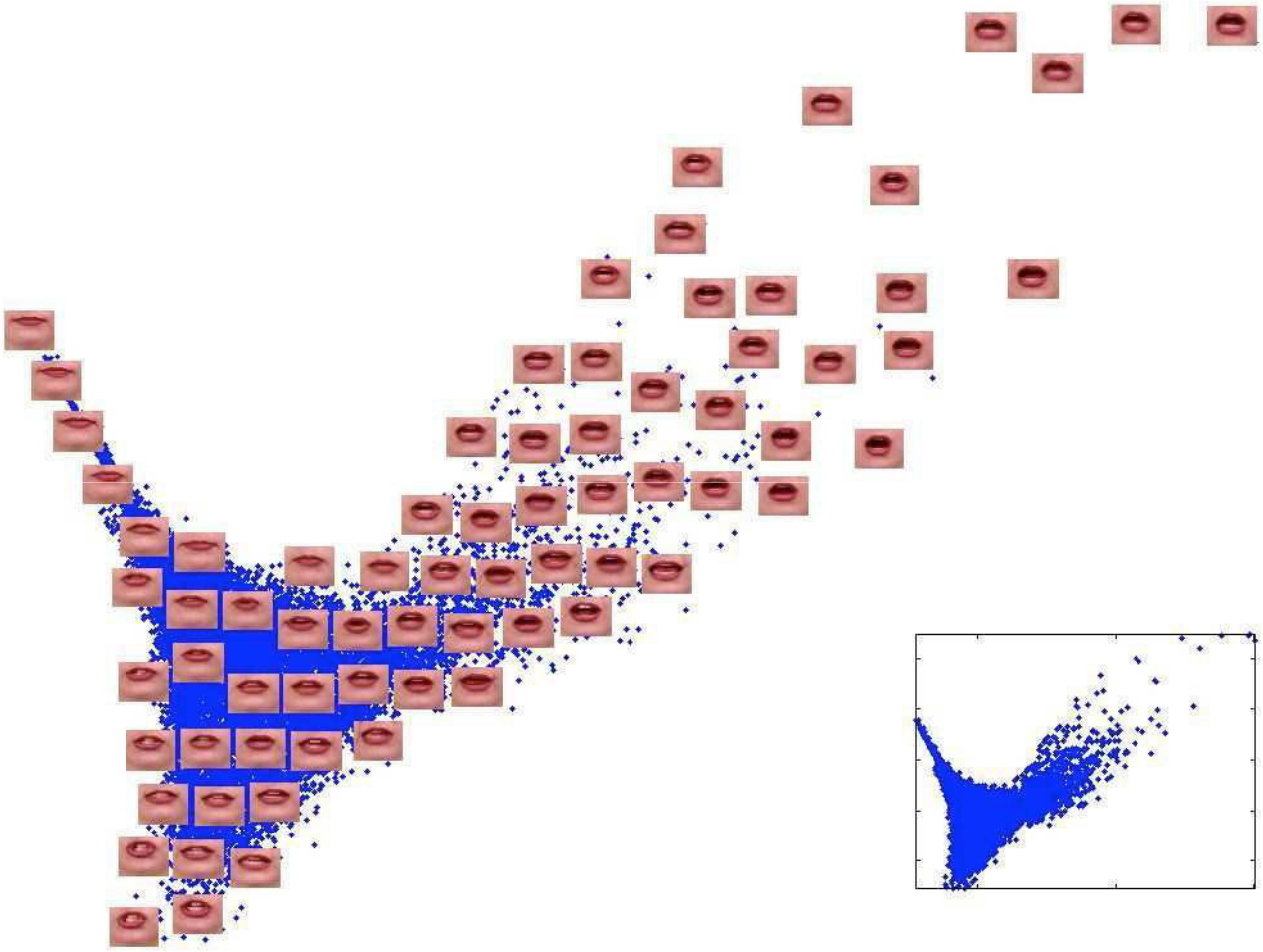
- Solution
  - Discard bottom eigenvector [1 1 … 1] (with eigenvalue zero).
  - Other eigenvectors satisfy constraints.

Surfaces

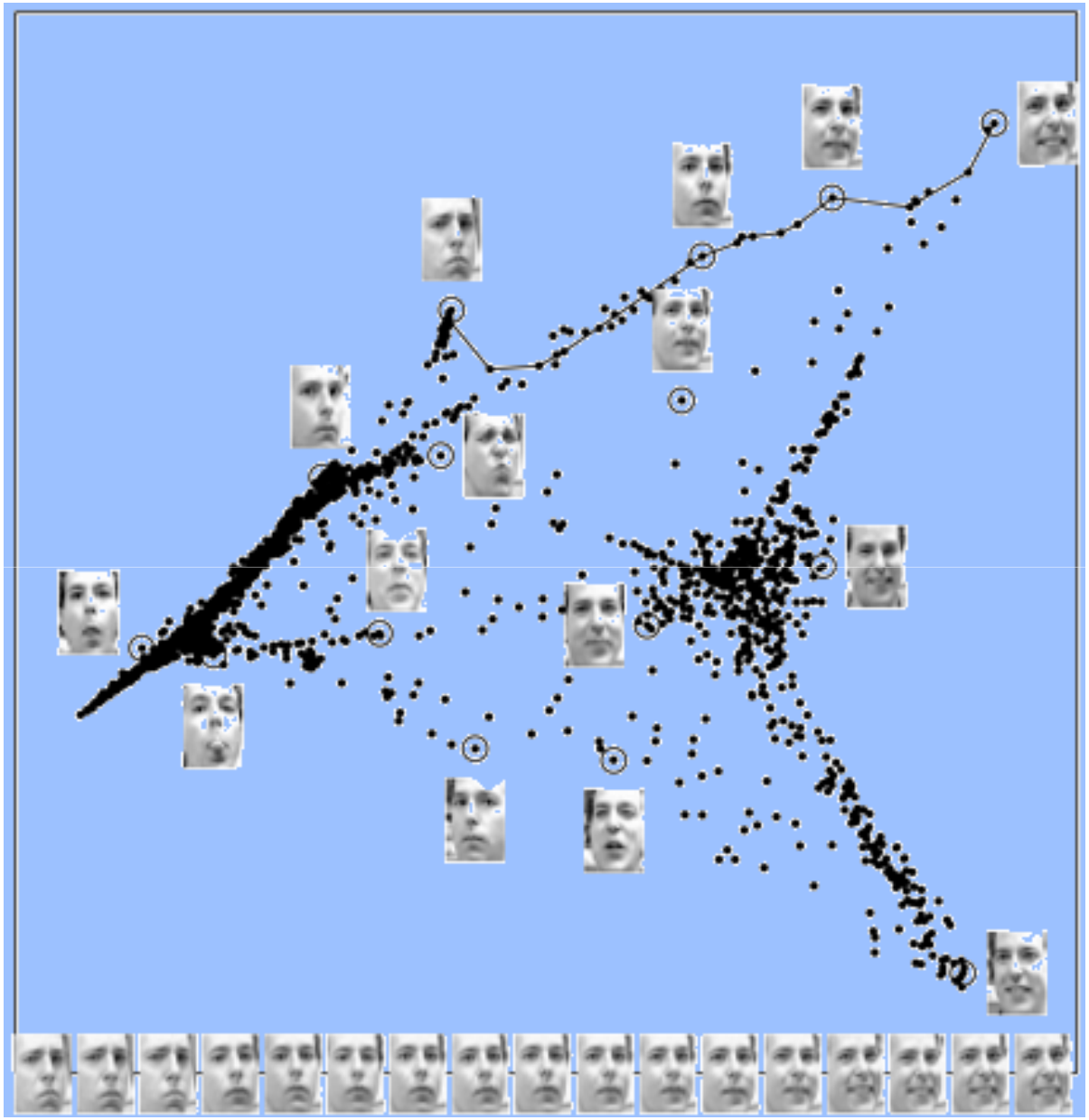
N=1000
inputs
k=8
nearest
neighbors



Lips
N=15960
images
K=24
neighbors
D=65664
pixels
d=2
(shown)



Pose and expression
N=1965 images
k=12 nearest neighbors
D=560 pixels
d=2 (shown)



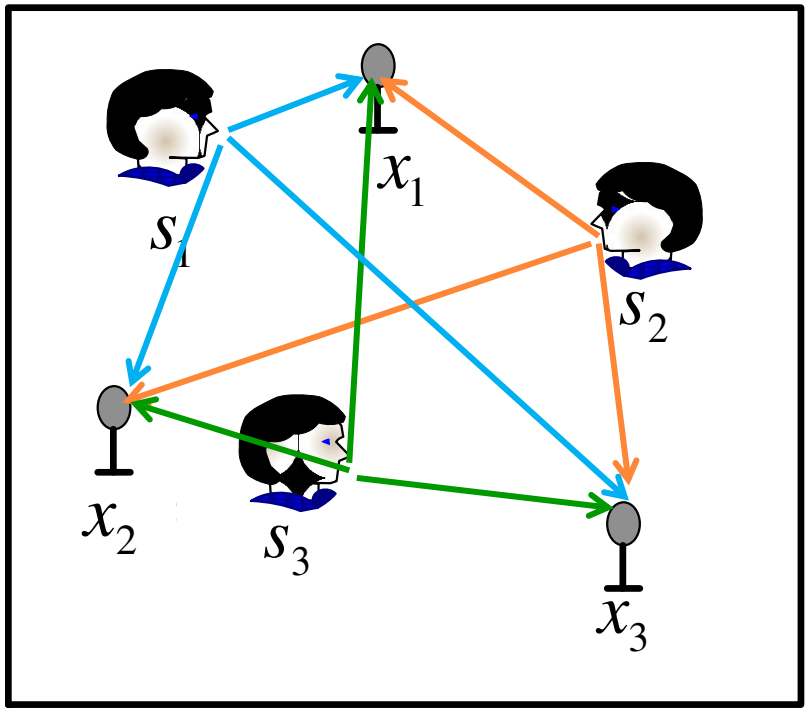
Properties of LLE

- Strengths:
  - Fast
  - No local minima
  - Non-iterative
  - Non-parametric (only heuristic is neighbourhood size).
- Weaknesses:
  - Sensitive to “shortcuts”
  - No estimate of dimensionality

LLE versus Isomap

- Many similarities
  - Graph-based, spectral method
  - No local minima
- Essential differences
  - Does not estimate dimensionality ☹
  - No theoretical guarantees ☹
  - Constructs sparse vs. dense matrix ☺
  - Preserves weights vs. distances
  - Much faster ☺

Cocktail Party



- microphone signals are mixed speech signals

$$x_1(t) = a_{11}s_1(t) + a_{12}s_2(t) + a_{13}s_3(t)$$
$$x_2(t) = a_{21}s_1(t) + a_{22}s_2(t) + a_{23}s_3(t)$$
$$x_3(t) = a_{31}s_1(t) + a_{32}s_2(t) + a_{33}s_3(t)$$

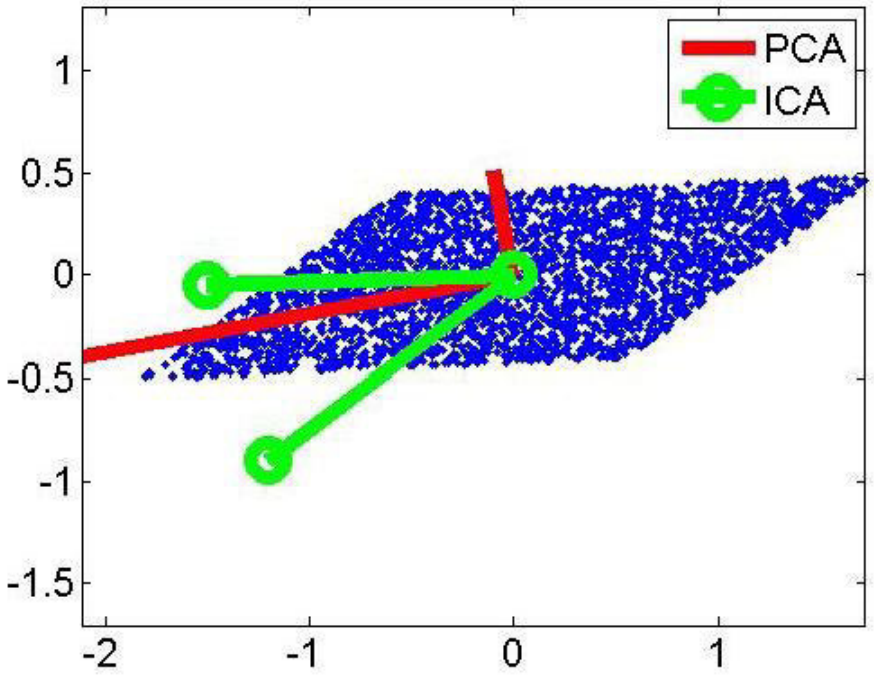
- Input: microphone signals x_1, x_2, x_3
- Goal: recover the speech signals s_1, s_2, s_3

http://research.ics.tkk.fi/ica/cocktail/cocktail_en.cgi

ICA vs. PCA

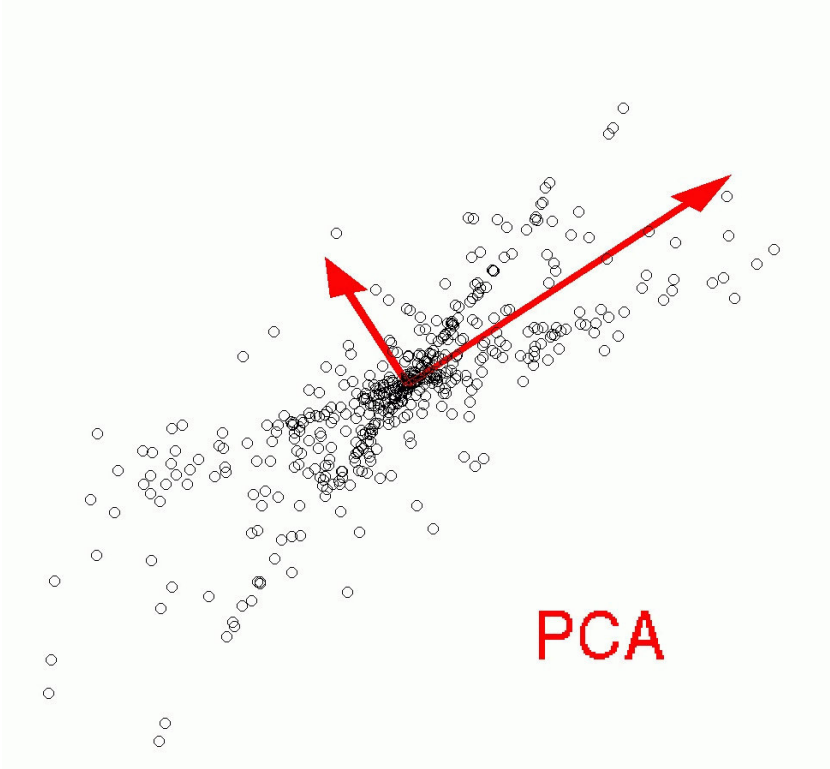
- Similar to PCA
  - Finds a new basis to represent the data
- Different from PCA
  - PCA removes only correlations**,** ICA removes correlations, **and higher order dependence.**
  - In PCA some components are **more important than others** (based on eigenvalues) in ICA components are **equally important.**

ICA vs. PCA

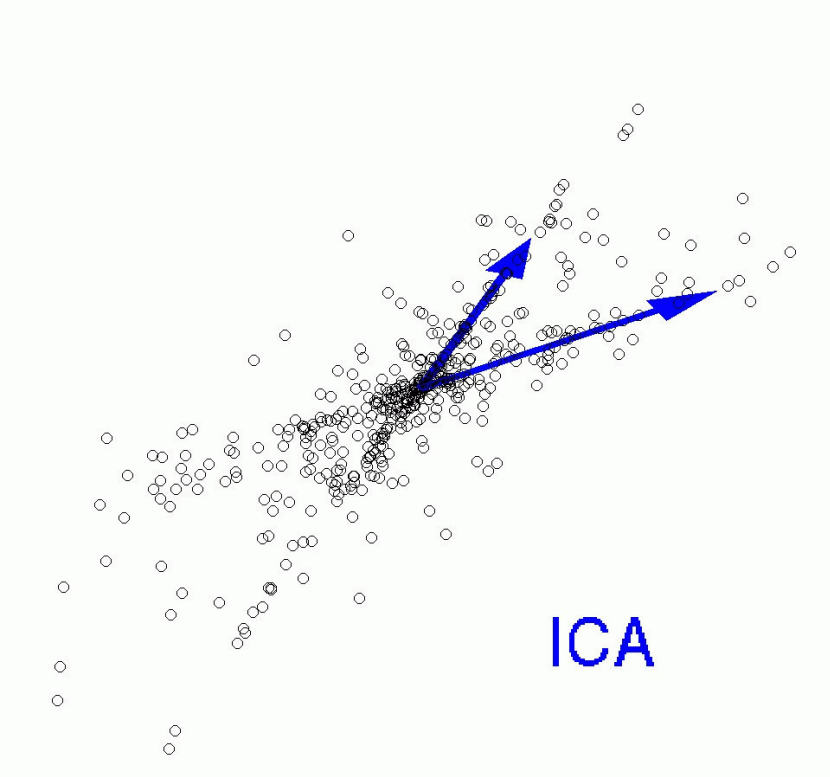


- PCA: principle components are orthogonal.
- ICA: independent components are not!

ICA vs. PCA



maximal variance directions



independent components

Model

- Assume data $s \in R^n$, generated by n independent sources.

- We assume:

$$x = As,$$

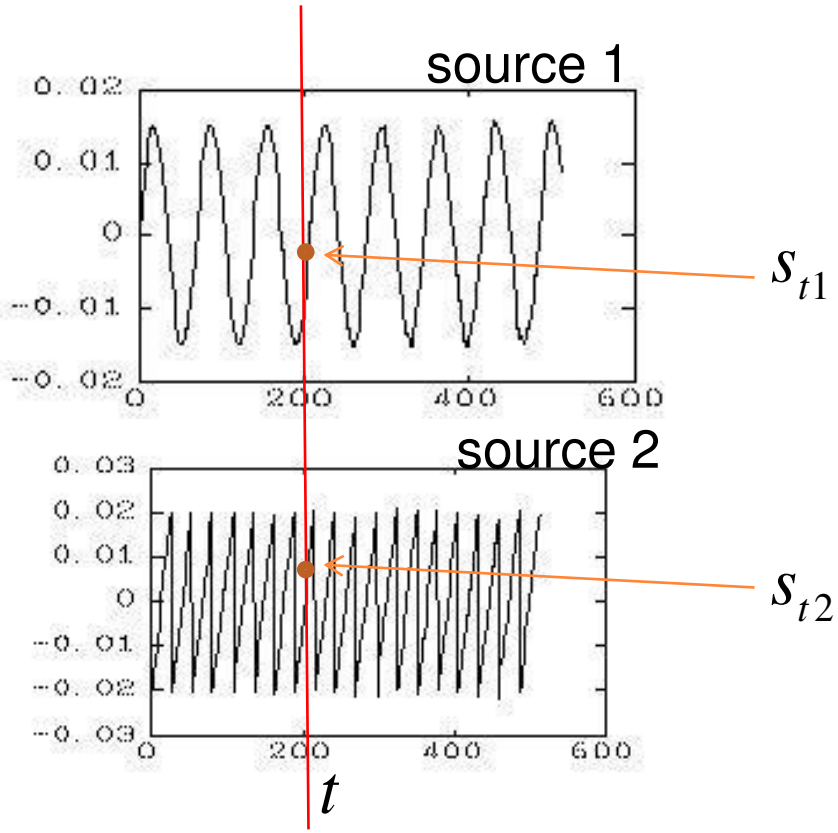
mixing matrix

$A \in R^{n \times n}$ is unknown

Model

- Assume data $s \in R^n$, generated by n independent sources.

s_{ij} signal from source j at time i.



mic. j at time i

$$x_{ij} = \sum_{k=1}^{n} A_{jk} s_{ik}$$

sum over sources

Problem Definition

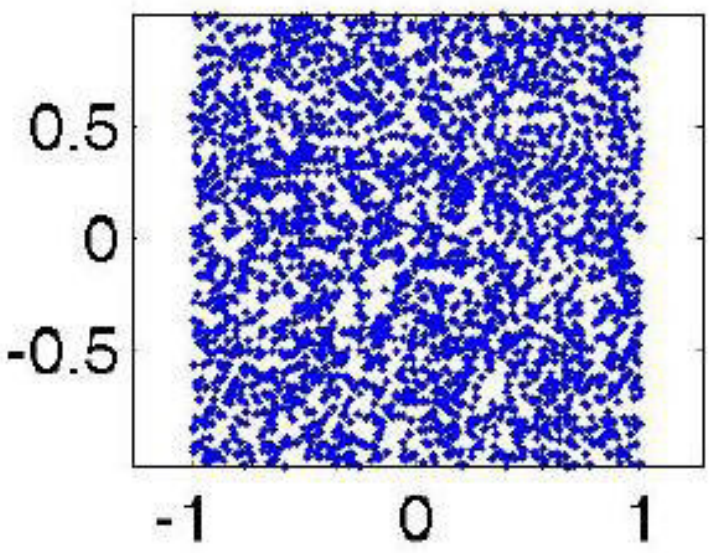
- We observe $\{x_i;\ i = 1,..,m\}$ *i* denotes time
- Goal: recover the sources s_j, that generated the data $(x = As)$.

- Let $W = A^{-1}$ unmixing matrix
- Goal is to find W, such that $s_i = Wx_i$
- Denote

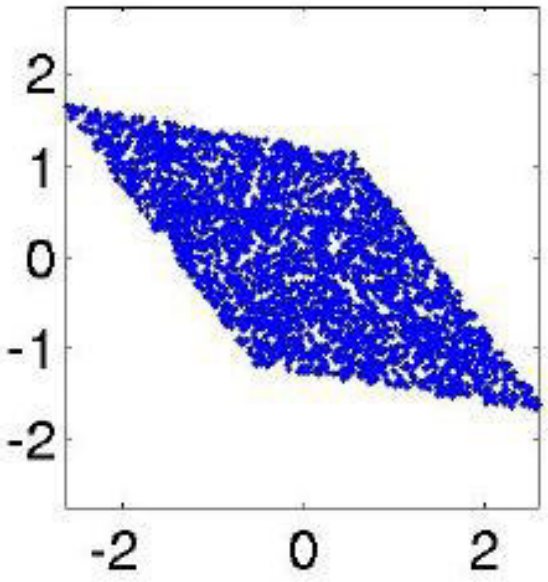
$$W = \begin{bmatrix} - w_1^T - \\ \vdots \\ - w_n^T - \end{bmatrix}$$

then the *j*-th source can be recovered by $s_{ij} = w_j^T x_i$

ICA Intuition



original



mixed

$s_j \in \text{Uniform}[-1,1]$

ICA Ambiguities

- If we have no prior knowledge about the mixing matrix, then there are inherent ambiguities in A that are impossible to recover.
- The sources can be recovered up to
  - Permutation
  - Scaling
  - Sign

Permutation Ambiguity

Assume that P is a nxn permutation matrix.

Examples: $P = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$; $P = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$;

$$W = \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{bmatrix}; \quad P = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}; \quad PW = \begin{bmatrix} w_{21} & w_{22} \\ w_{11} & w_{12} \end{bmatrix}$$

Given only the x_i's , we cannot distinguish between W and PW.

The permutation of the original sources is ambiguous.

Not important in most applications

Scaling Ambiguity

$$x_i = As_i$$

$$A \to 2, \quad s_i \to (0.5 s_i) \quad \Rightarrow \quad x_i = 2A(0.5 s_i)$$

$$A \to \begin{bmatrix} | & & | & \\ a_1 & \cdots & \alpha a_j & \cdots \\ | & & | & \end{bmatrix}, \; s_j \to 1/\alpha \, s_j \quad \Rightarrow \quad x_i = \begin{bmatrix} | & & | & \\ a_1 & \cdots & \alpha a_j & \cdots \\ | & & | & \end{bmatrix} \begin{bmatrix} s_{i1} \\ \vdots \\ 1/\alpha s_{ij} \\ \vdots \end{bmatrix}$$

We cannot recover the "correct" scaling of the sources.

Not important in most applications

Scaling a speaker's speech signal s_j by some positive factor affects only the volume of that speaker's speech.

Also, sign changes do not matter: s_j and $-s_j$ sound identical when played on a speaker.

Gaussian sources are problematic

$$n = 2, s \sim N(0, I), \quad x = As$$

$$x \sim N(0, AA^T)$$

$$E[xx^T] = E[Ass^T A^T] = AA^T$$

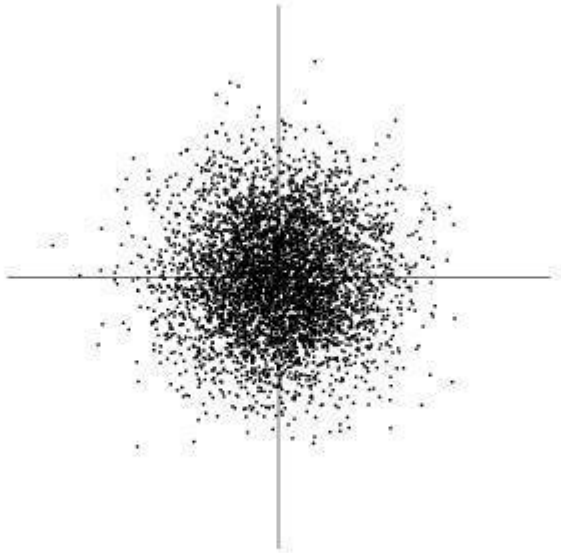


Figure 7: The multivariate distribution of two independent gaussian variables.

Let R be an arbitrary orthogonal matrix, such that $RR^T = R^T R = I$.

Let $A' = AR$, then $x' = A's \Rightarrow x' \sim N(0, AA^T)$

$$E[x'x'^T] = E[A'ss^T A'^T] = E[ARss^T (AR)^T] = ARR^T A = AA^T$$

Gaussian Sources are Problematic

- Whether the mixing matrix is A or A', we would observe data from a $N(0, AA^T)$ distribution.
- Thus, there is no way to tell if the sources were mixed using A or A'.
- There is an arbitrary rotational component in the mixing matrix that cannot be determined from the data, and we cannot recover the original sources.
- Reason: The Gaussian distribution is **spherically symmetric.**
- For non-Gaussian data, it is possible, given enough data, to recover the n independent sources.

Densities and linear transformations

Suppose s is a r.v drawn according to $p_s(s)$.

Let $x \in R$ be a r.v. defined by $x = As$. The density of x is given by:

$$p_x(x) = p_s(Wx) \cdot |W|$$

where $W = A^{-1}$ (A is squared invertible matrix)

Example: $s \sim \text{Uniform}[0,1] : p_s(s) = 1 (0 \le s \le 1)$

Let $A = 2$, then $x = 2s$. Clearly, $x \sim \text{Uniform}[0,2]$

Thus , $p_x(x) = 0.5 (0 \le x \le 2)$.

ICA algorithm

- Assume that the distribution of s_i is $p_s(s_i)$.
- The joint distribution is

$$p(s) = \prod_{j=1}^{n} p_s(s_j)$$

sources are independent

- Using the previous formulation, we can derive

$$p(x) = \prod_{j=1}^{n} p_s(w_j^T x)|W|$$

$$x = As = W^{-1}s$$

$$p(x) = p_s(Wx) \cdot |W|$$

- We must specify a density for the individual sources p_s.

ICA algorithm

- A cumulative distribution of a real r.v. z is defined by

$$F(z_0) = P(z \le z_0) = \int_{-\infty}^{z_0} p_z(z)dz$$

- The density of z can be found by $p_z(z) = F'(z)$.

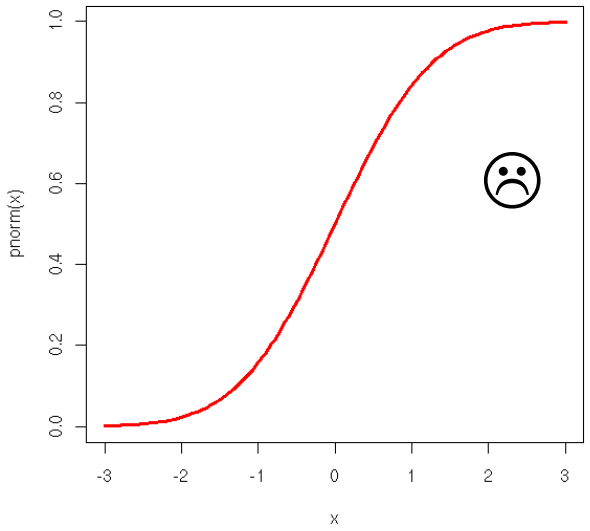
Specify a density for the s_j ⇒ specify its cdf.

If you have a prior knowledge that the sources' densities take a certain form, then use it here, otherwise make an assumption about cdf.

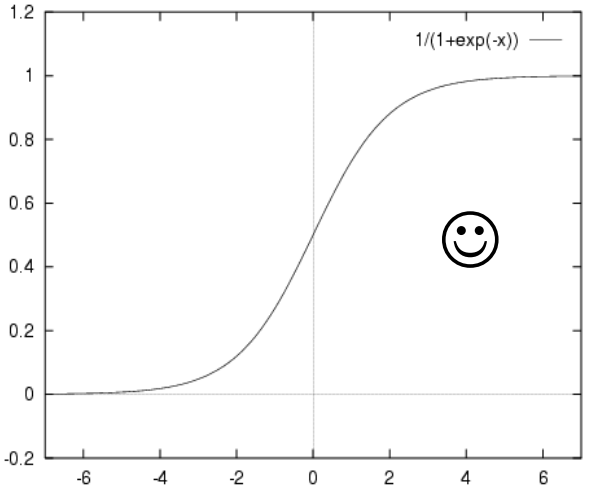
Density of s

cdf is has to be a monotonic function that increases from zero to one.

Gaussian CDF



sigmoid



$$g(s) = 1/(1+e^{-s})$$

$$p(s) = g'(s)$$

We assume that the data x_i has zero mean. This is necessary because our assumption that $p(s) = g'(s)$ implies $E(s) = 0$. Thus $E(x) = E(As) = 0$

ICA algorithm

- W is a parameter of our model that we want to estimate.
- Given a training set $\{x_i; i = 1,..., m\}$, the log likelihood is:

$$l(W) = \sum_{i=1}^{m} \left(\sum_{j=1}^{s} \log g'\left(w_j^T x_i\right) + \log|W| \right).$$

- Maximize $l(W)$ using gradient ascent:

$W \leftarrow W + \eta \nabla l(W)$, where η is the learning rate.

Equivalently, $w_j \leftarrow w_j + \eta \boxed{\frac{\partial}{\partial w_j} l(W)}$?

ICA algorithm

- By taking the derivatives of $l(W)$ using:

$$g(x) = 1/(1+e^{-x});\ \ g'(x) = g(x)(1-g(x))$$

$$\nabla_W |W| = |W|\left(W^{-1}\right)^T$$

we obtain the update rule:

$$W \leftarrow W + \eta\left(\begin{bmatrix} 1-2g\left(w_1^T x_i\right) \\ 1-2g\left(w_2^T x_i\right) \\ \vdots \\ 1-2g\left(w_n^T x_i\right) \end{bmatrix} x_i^T + \left(W^T\right)^{-1}\right)$$

- When the algorithm converges, compute $s_i = Wx_i$.

Remarks

- We assumed that $\{x_i; i = 1,..., m\}$ are independent of each other.
- This assumption is incorrect for time series where the x_i's are dependent (e.g. speech data).
- it can be shown, that having correlated training examples will not hurt the performance of the algorithm if we have sufficient data.
- Tip: run stochastic gradient ascent on a randomly shuffled copy of the training set.

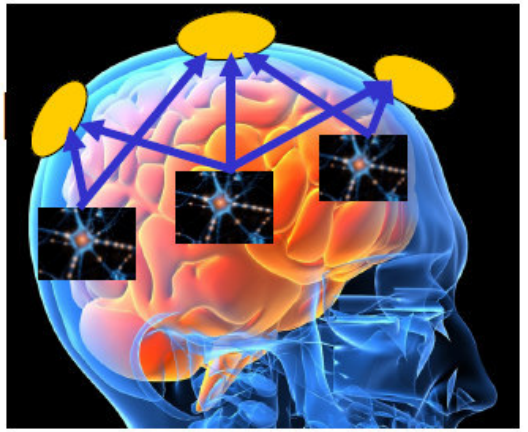
Application domains of ICA

- Blind source separation
- Image denoising
- Medical signal processing – fMRI, ECG, EEG
- Modelling of the hippocampus and visual cortex
- Feature extraction, face recognition
- Compression, redundancy reduction
- Watermarking
- Clustering
- Time series analysis (stock market, microarray data)
- Topic extraction
- Econometrics: Finding hidden factors in financial data

Slide due B. Poczos

ICA Application, Removing Artifacts from EEG

- EEG ~ *Neural cocktail party*
- Severe ***contamination*** of EEG activity
  - eye movements
  - blinks
  - muscle
  - heart, ECG artifact
  - vessel pulse
  - electrode noise
  - line noise, alternating current (60 Hz)



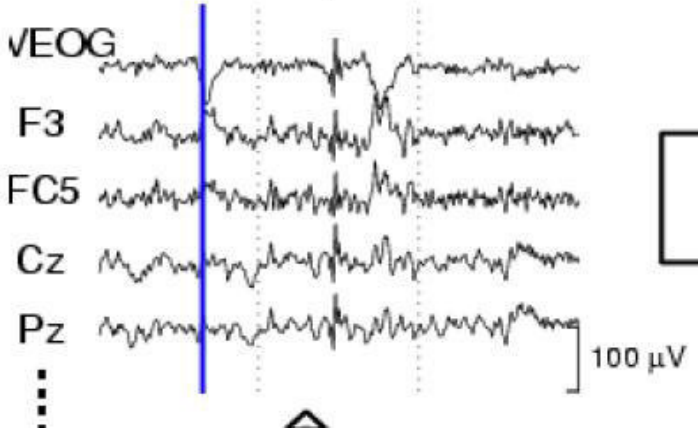
- ICA can improve signal
  - effectively ***detect, separate and remove*** activity in EEG records from a wide variety of artifactual sources. (Jung, Makeig, Bell, and Sejnowski)

- ICA weights help find **location** of sources

Slide due B. Poczos

ICA decomposition

EEG Scalp Channels



unmixing
(W)

Independent Comp

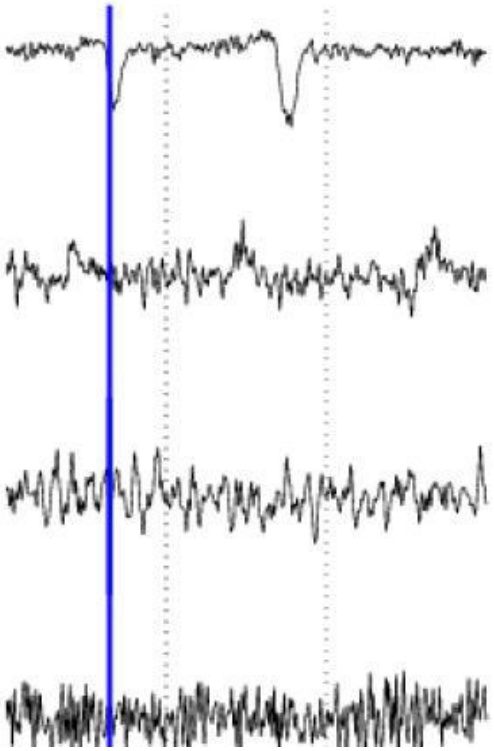


Fig. from Jung

Summed Projection of Selected Components

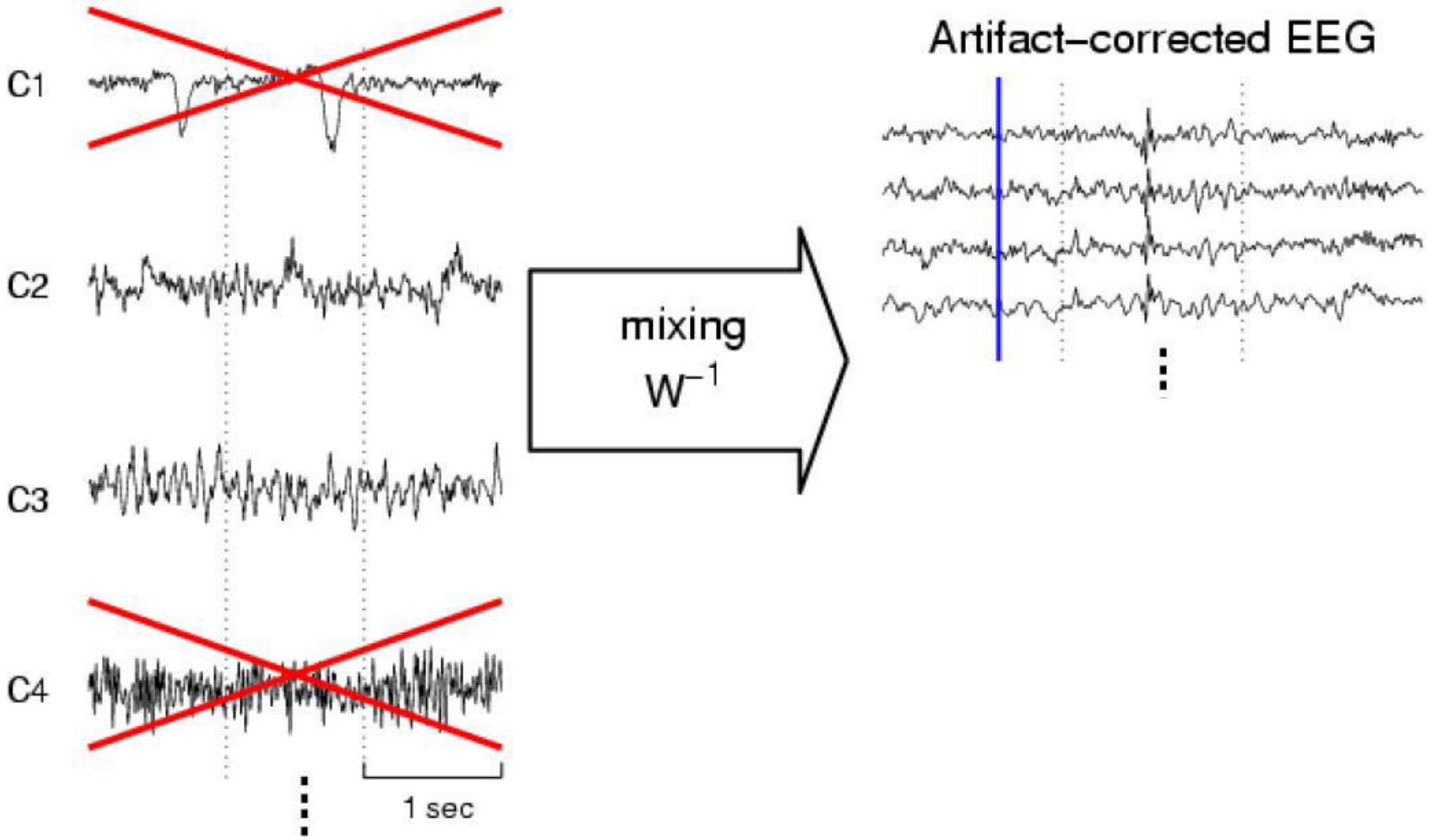


Fig from Jung

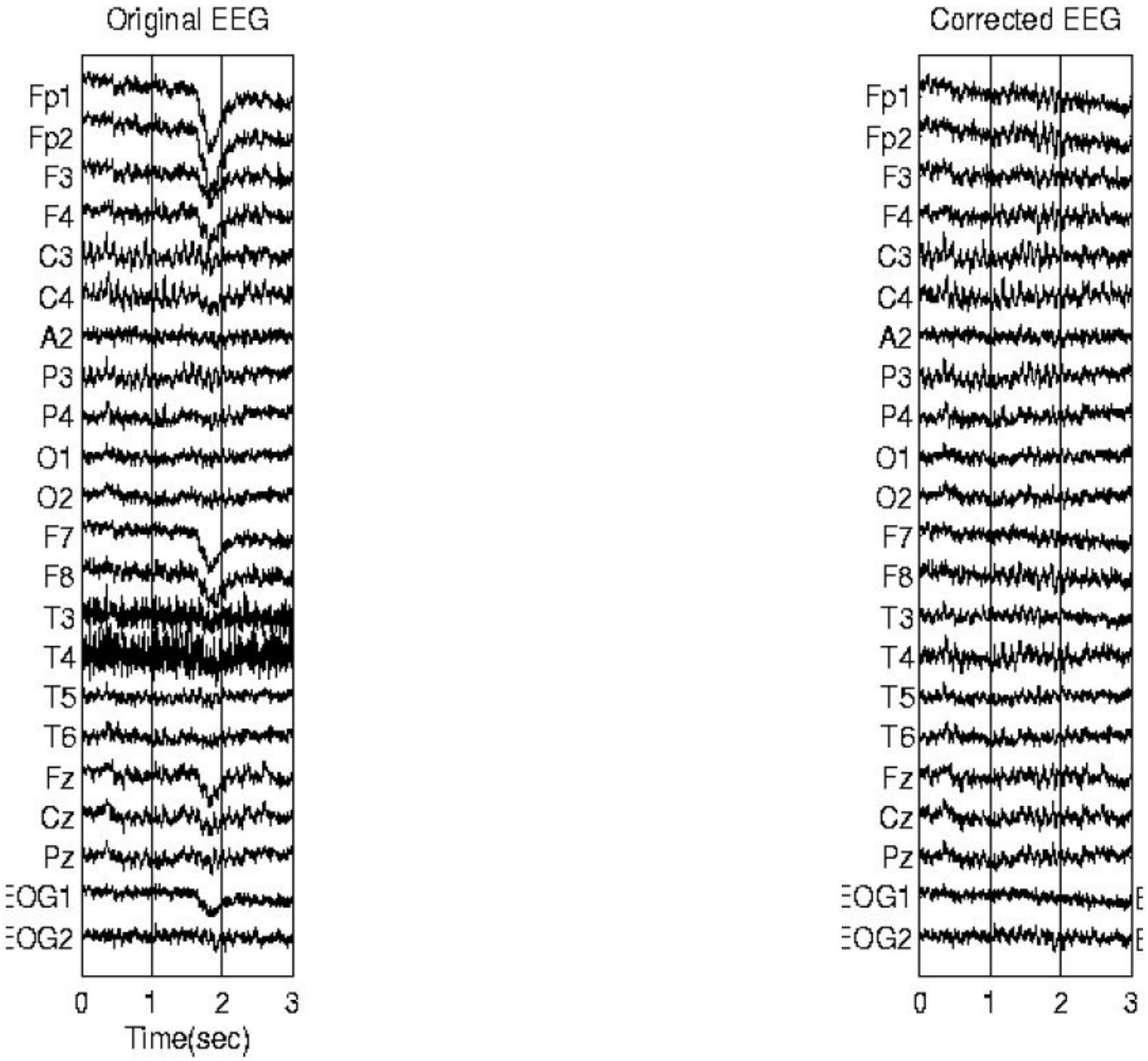
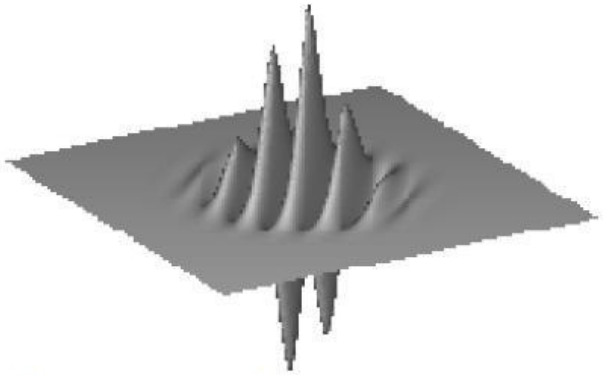
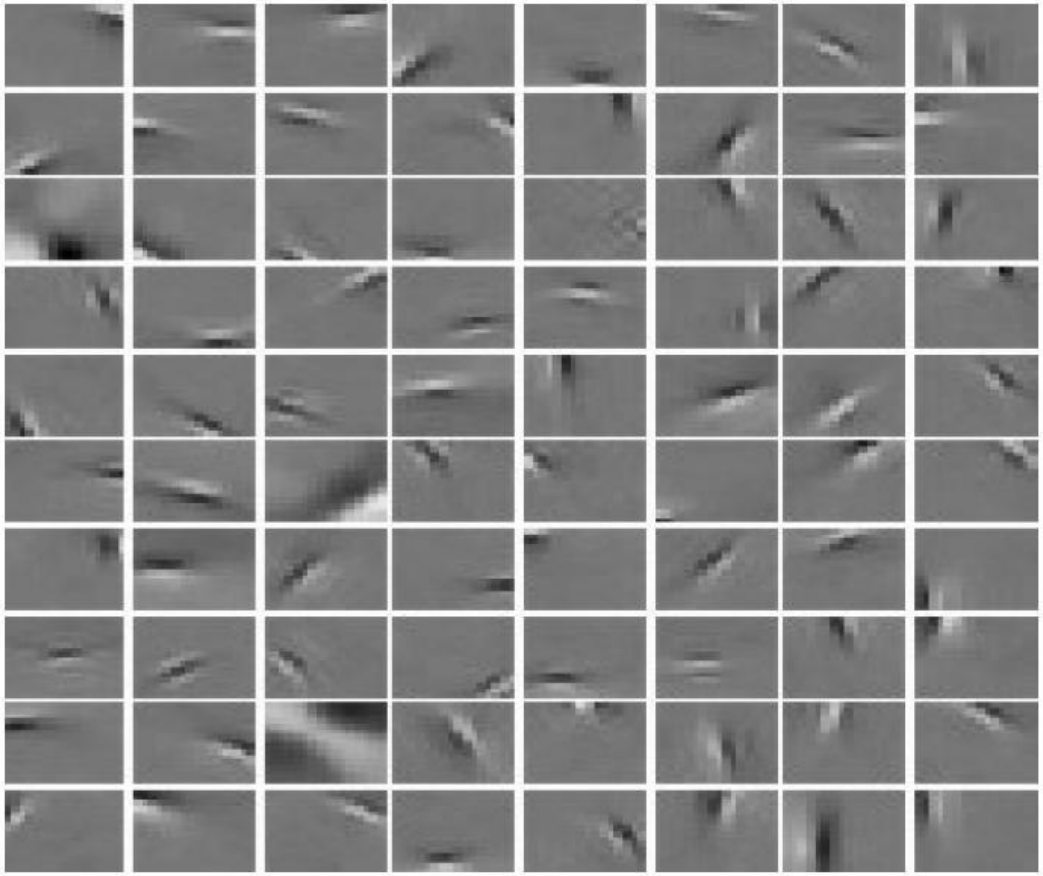


Fig from Jung

ICA basis vectors extracted from natural images

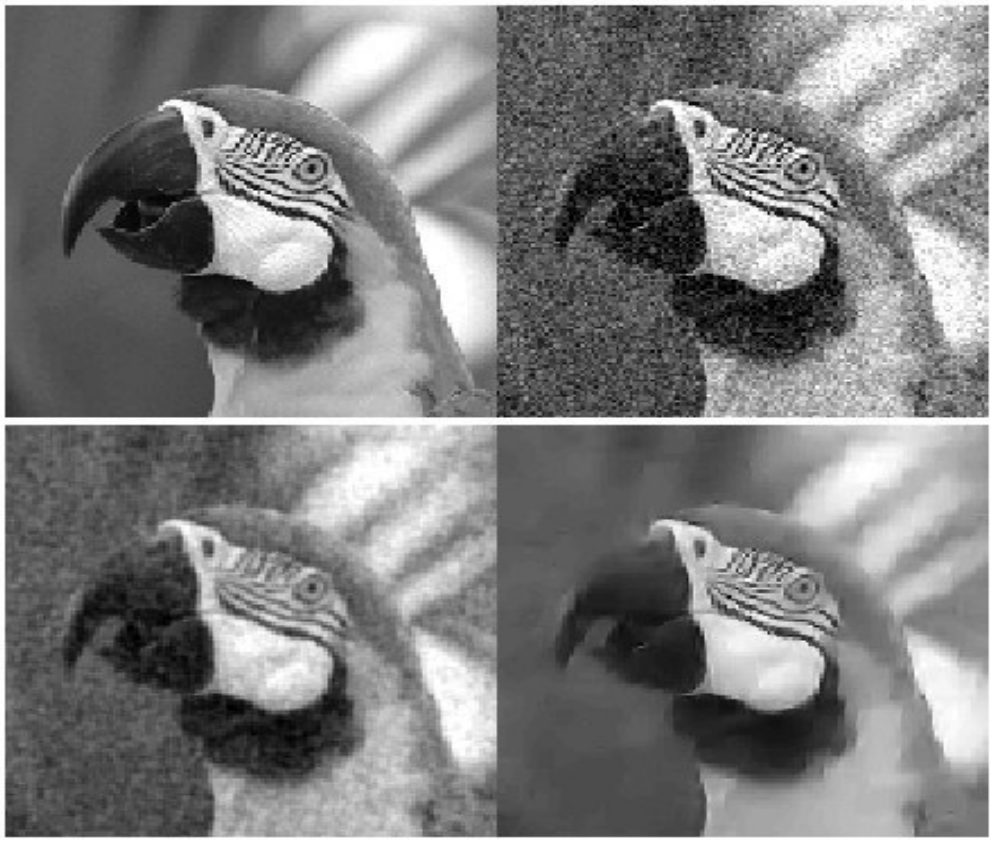




Gabor wavelets,
edge detection,
receptive fields of V1 cells...

Image denoising

Original image



Noisy image

Wiener filtering

ICA filtering

Doing Machine Learning

1. Get new data
2. Select the right technique to build a model out of it
   - Question Time: is there any structure into the data?
   - Soon after, you start guessing about the best technique to be used...
3. Train the model
4. Evaluate predictions

3rd party hints about data structure are always very welcome, but looking for insights on your own is the suggested approach!

Random guessing is usually not the best approach...

CAMLIN

Doing Machine Learning

In the words of Laurens Van Der Maaten:

> *"Always visualize your data first, before you start to train predictors on the data! Oftentimes, visualizations such as the ones I made provide insight into the data distribution that may help you in determining what types of prediction models to try."*

Doing Machine Learning

Trying to visualize input data makes sense, but:

- plotting 2D data is a fairly easy task
- plotting 3D data is easy, but grasping data structures while navigating the plot might be tricky!
- how to visualize 4D data? ...
- how to visualize N-D data? ...

The natural choice is to project input data onto a 2D space so to easily visualize it!

How? Reducing data dimensionality, **obvious**!

CAMLIN

Dimensionality Reduction - Simplified Statement

Given as input a set of high-dimensional data vectors

$$X = \{\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_k\}, \quad \vec{x}_i \in \mathbb{R}^m$$

determine a set of output vectors

$$Y = \{\vec{y}_1, \vec{y}_2, \ldots, \vec{y}_k\}, \quad \vec{y}_i \in \mathbb{R}^n, \quad n < m$$

with *lower* dimensionality in a way to **retain as much information (data structure) as possible**.

Dimensionality Reduction - Simplified Statement

The problem can be also formulated as a minimization problem:

$$\arg\min_{Y} f(P, Q)$$

where P is a square matrix which elements

$$p_{ij} = g(\vec{x}_i, \vec{x}_j)$$

express a *similarity* measure for all pairs of input samples, Q is a square matrix which elements

$$q_{ij} = h(\vec{y}_i, \vec{y}_j)$$

express a similarity measure for all pairs of output samples, and f judges whether Y shows a data structure different to the one of X or not.

Dimensionality Reduction - Examples of Known Techniques

Linear techniques:

- Principal Component Analysis (PCA)
- Independent Component Analysis (ICA)

Non-Linear techniques:

- Elastic Map
- Deep Belief Networks, and Auto-encoders in general
- **t-Student Stochastic Neighborhood Embedding (tSNE)**

Dimensionality Reduction (a side note)

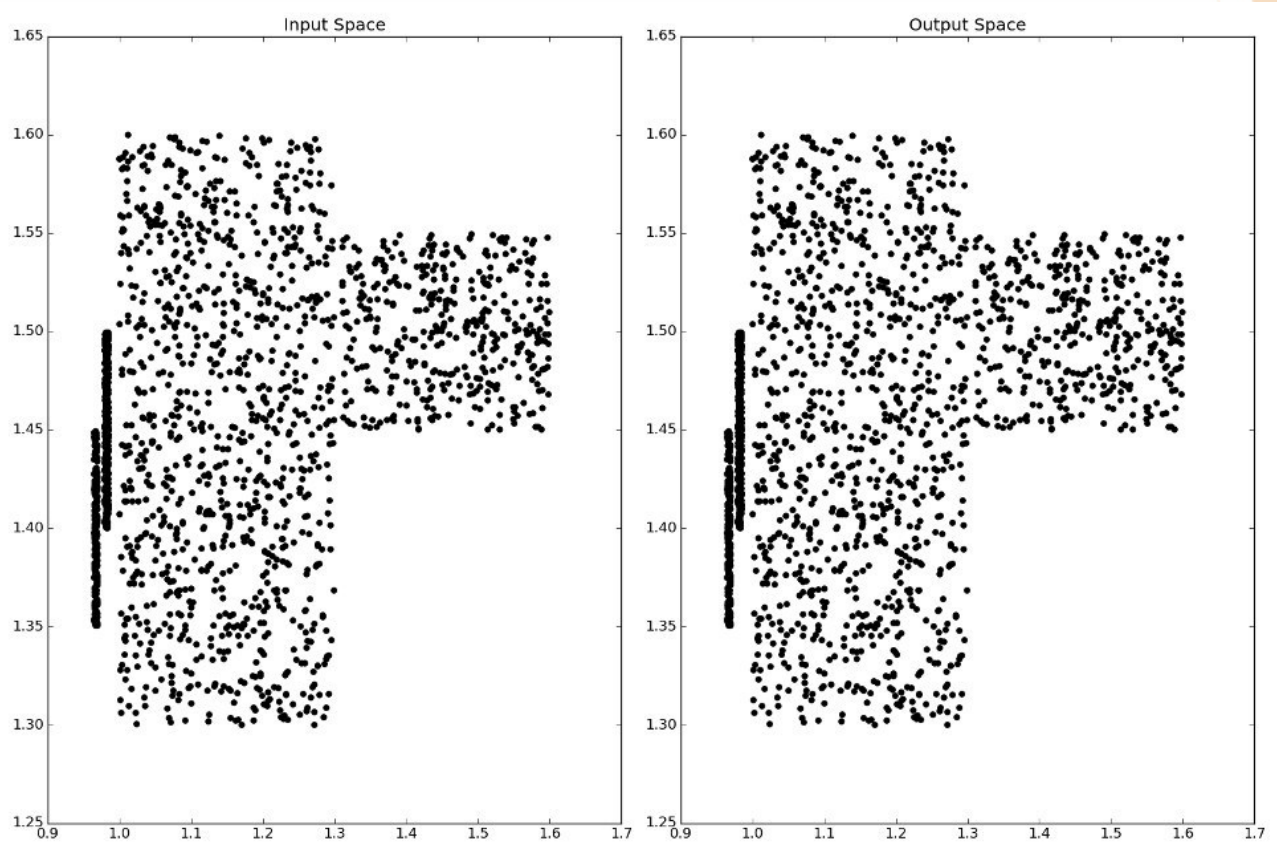
The *course of dimensionality* is a well known drawback in data analysis for which the increase of the feature space dimension is associated with a decrease on the performances of all techniques *based on distance computation*. A few examples are:

- K-Nearest Neighbor
- K-Means
- Hierarchical Clustering

For this reason *dimensionality reduction* is often the first processing step applied to input data so to improve the performances of the following steps.

Dimensionality Reduction - Dummy Example

From 2D to 2D using $P \equiv Q$



In this case, requiring to get keep the very same data structure unchanged should provide trivial results.

The Idea Behind SNE

Abandon exact distance measures or exact relationships (no more springs!)

Input space: building P using probability distributions so that picking two similar points is much more probable than picking two points which are well far apart.

Output space: building Q using probability distributions to judge distances between points.

SNE

Input Space Activations: Gaussian

$$p_{ij} = \frac{\exp\left(-\delta_{ij}^2/\sigma\right)}{\sum_k \sum_{l \neq k} \exp\left(-\delta_{kl}^2/\sigma\right)}, \quad \text{for } \forall i \forall j : i \neq j$$

neighborhoods' dimension.

Output Space Activations: Gaussian

$$q_{ij} = \frac{\exp(-\|\mathbf{y}_i - \mathbf{y}_j\|^2)}{\sum_k \sum_{l \neq k} \exp(-\|\mathbf{y}_k - \mathbf{y}_l\|^2)}, \quad \text{for } \forall i \forall j : i \neq j$$

Despite the clever idea behind, SNE still suffer from the so called *crowding problem*: it is impossible to preserve the whole structure due to the distortions introduced by the dimensionality reduction.

CAMLIN

Local Structure - A Different Perspective

Landscape observed from *Colle dell'Infinito*, Recanati (Italy)

Local Structure - A Different Perspective

Local Structure - A Different Perspective

Local Structure - A Different Perspective

Local Structure - A Different Perspective

Local Structure - A Different Perspective

Local Structure - A Different Perspective

Local Structure - A Different Perspective

Local Structure - A Different Perspective

CAMLIN

Local Structure - A Different Perspective

Local Structure - A Different Perspective

Local Structure - A Different Perspective

Local Structure - A Different Perspective

Local Structure - A Different Perspective

Local Structure - A Different Perspective

The Idea Behind t-SNE

In order to overcome the so called *crowding problem* lets be sensible to **local structure** only by:

- Input space: using a Gaussian as for SNE but selecting neighborhood size σ independently for each point
- Output space: building Q using a **heavy-tailed distribution**: Student's t.
- Cost Function: **judging information loss** going from X to Y using KL-Divergence to ignore low pairwise activations

$$\lim_{x \to +\infty} \frac{tStudent(x)}{Gaussian(x)} = +\infty$$

Input Space Activations: Gaussian

$$p_{ij} = \frac{\exp\left(-\delta_{ij}^2/\sigma_i\right)}{\sum_k \sum_{l \neq k} \exp\left(-\delta_{kl}^2/\sigma_k\right)}, \quad \text{for } \forall i \forall j : i \neq j$$

σ_i is selected for each point by initially setting a target **perplexity**.

Output Space Activations: Student's t

$$q_{ij} = \frac{(1 + \|\mathbf{y}_i - \mathbf{y}_j\|^2)^{-1}}{\sum_k \sum_{l \neq k} (1 + \|\mathbf{y}_k - \mathbf{y}_l\|^2)^{-1}}, \quad \text{for } \forall i \forall j : i \neq j$$

CAMLIN

Kullback-Leibler Divergence

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

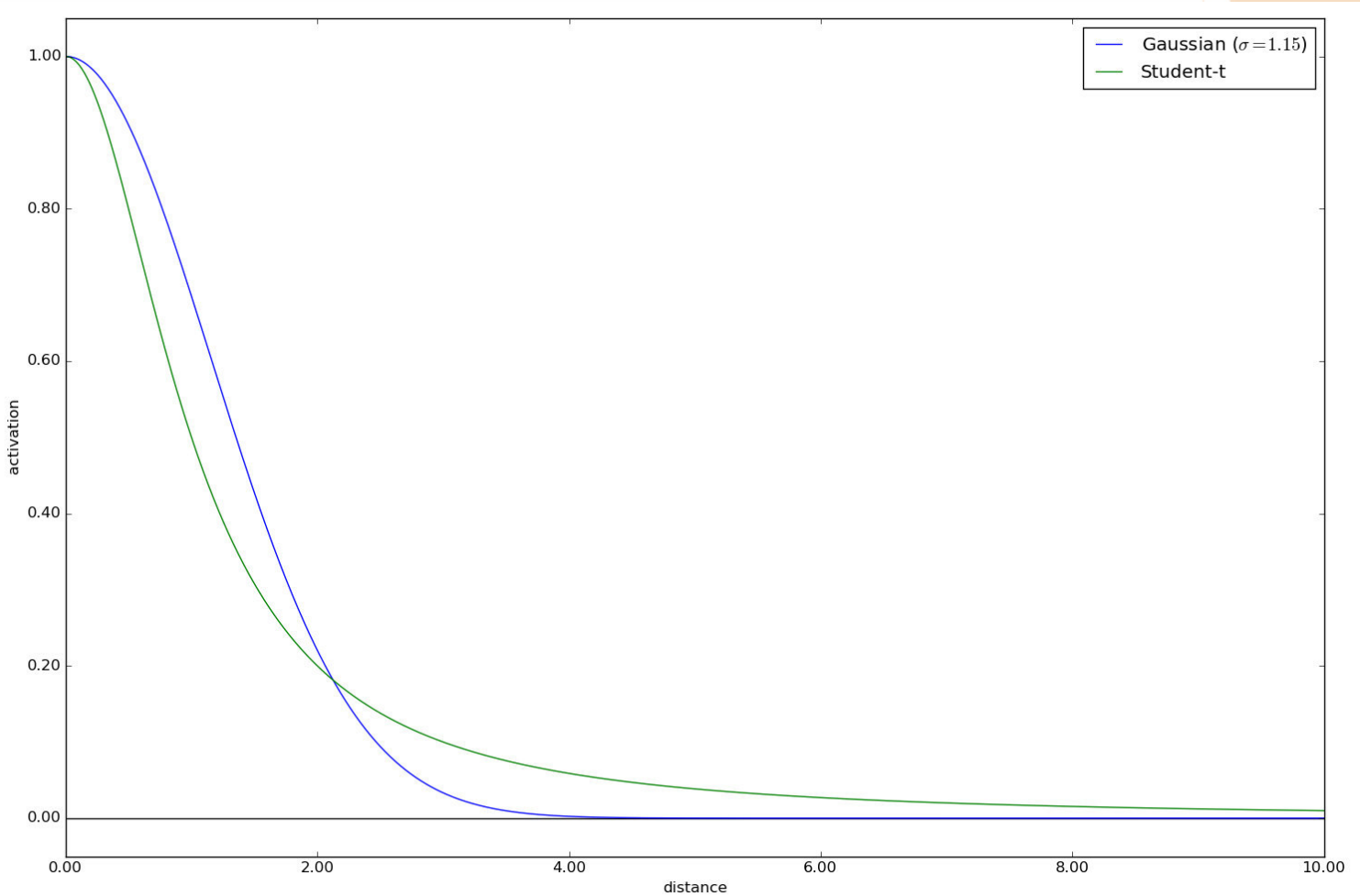
A natural way to measure how two probability distributions compare.

It also fits well with the t-SNE idea: “similar” points in the input space will add a big penalty if in the output space they result “dissimilar”

- large p_{ij} modelled with small q_{ij} result in a big penalty
- small p_{ij} modelled with large q_{ij} result in a not so big penalty

t-SNE - The Trick Explained

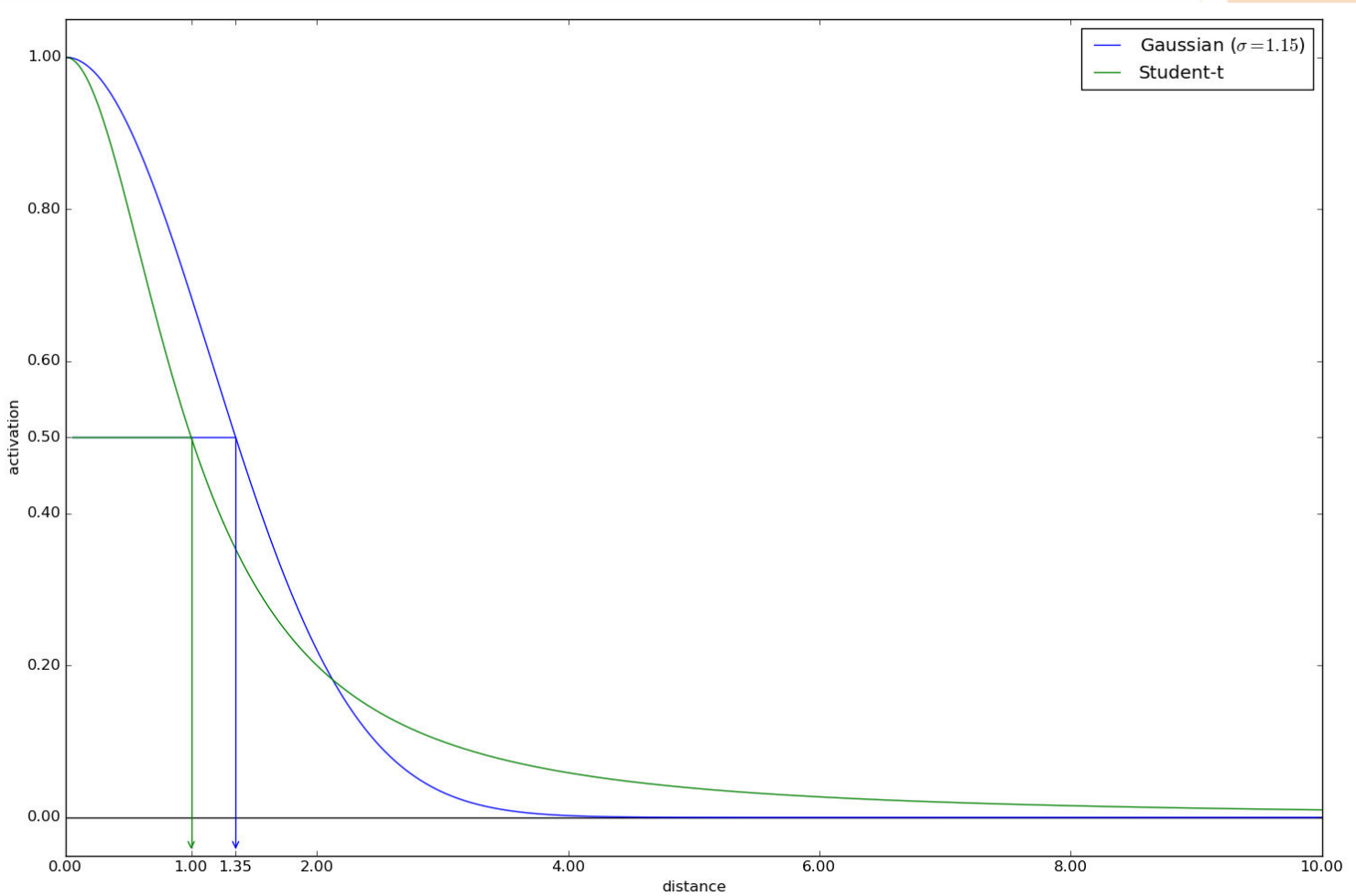
Gaussian activation vs. Student's t activation



See the *heavy-tail*?!

t-SNE - The Trick Explained

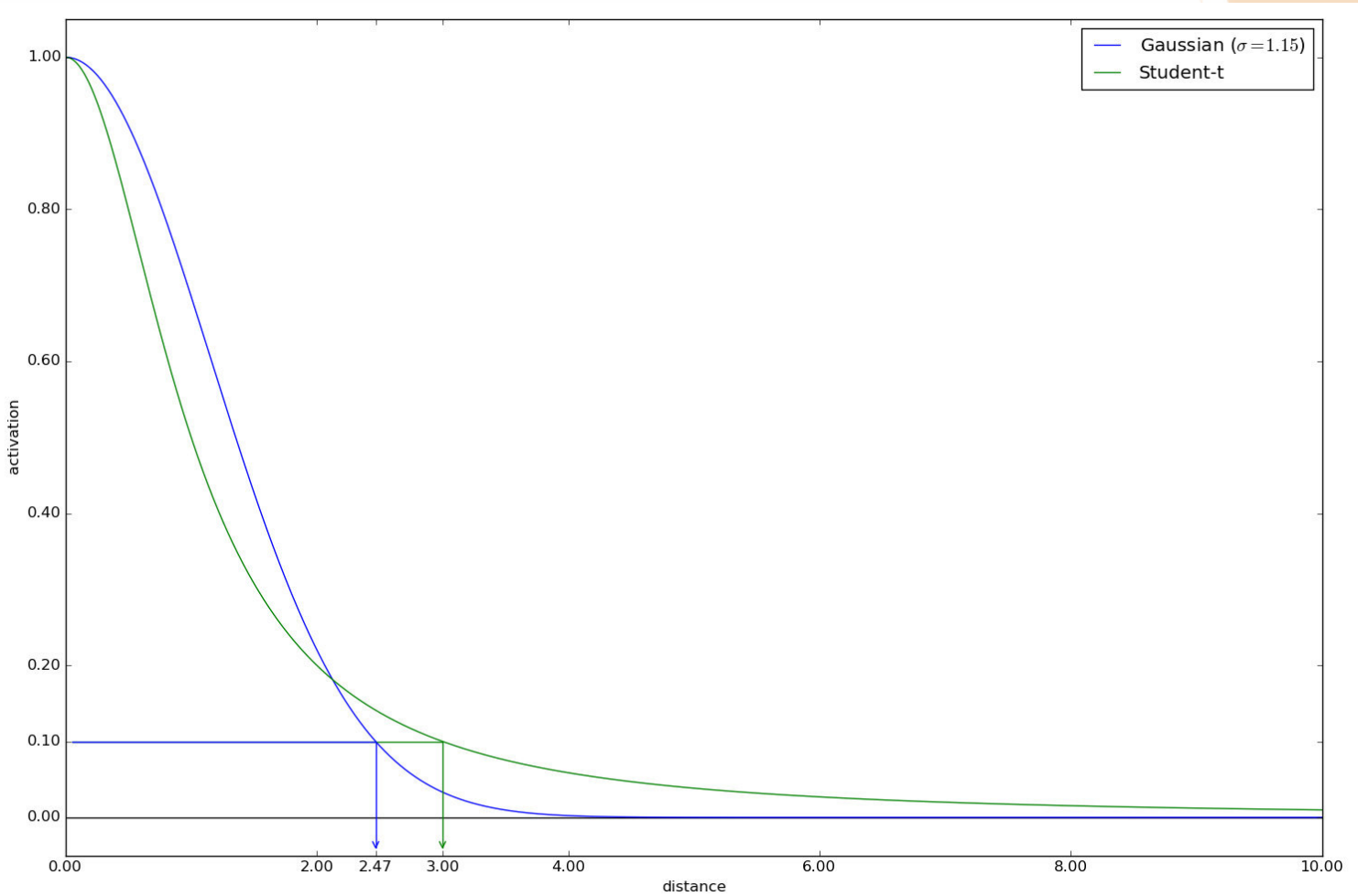
Gaussian activation vs. Student's t activation



Local neighbors are attracted to a little closer position.

t-SNE - The Trick Explained

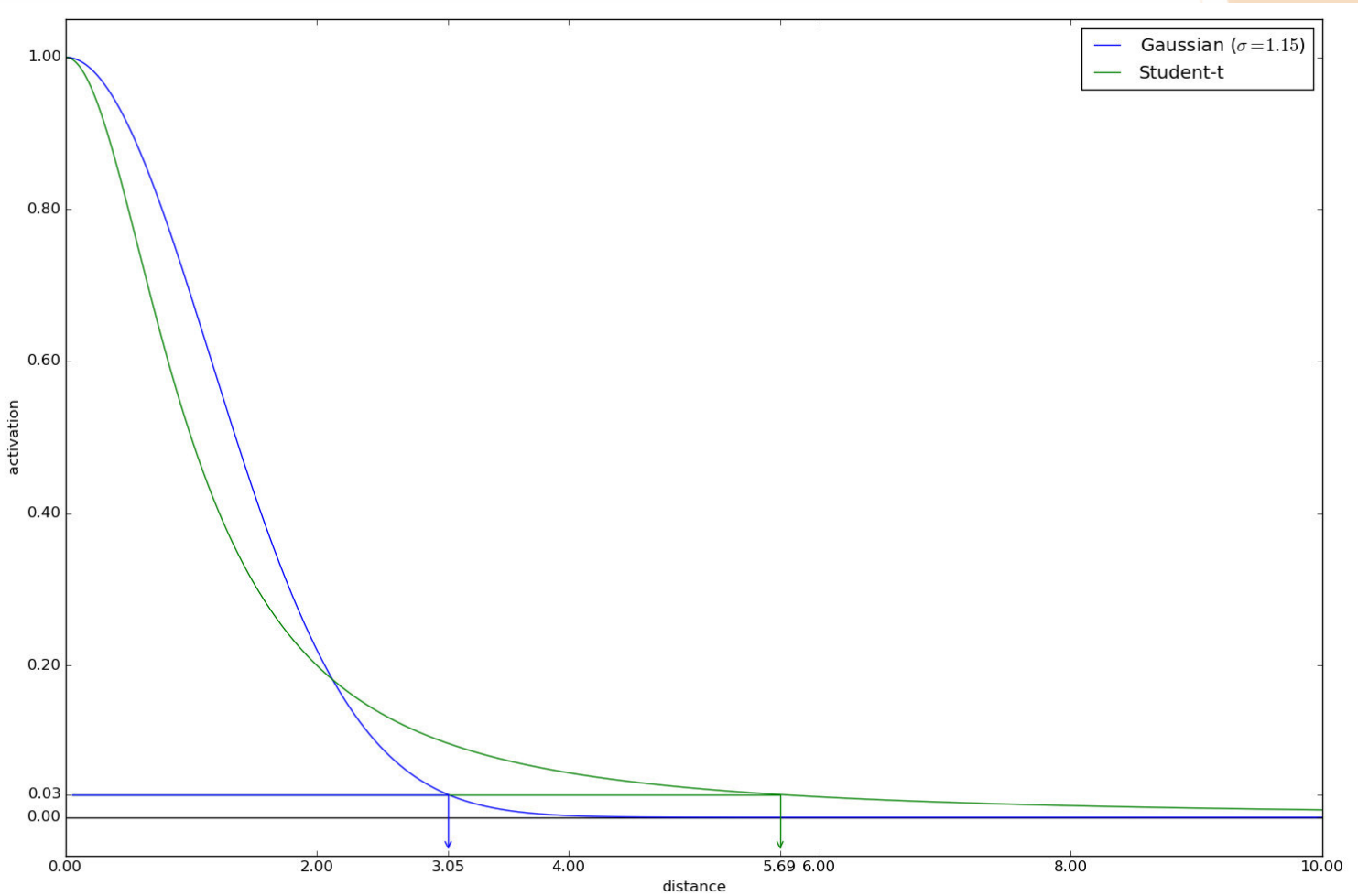
Gaussian activation vs. Student's t activation



Non-local neighbors might get a little repulsed.

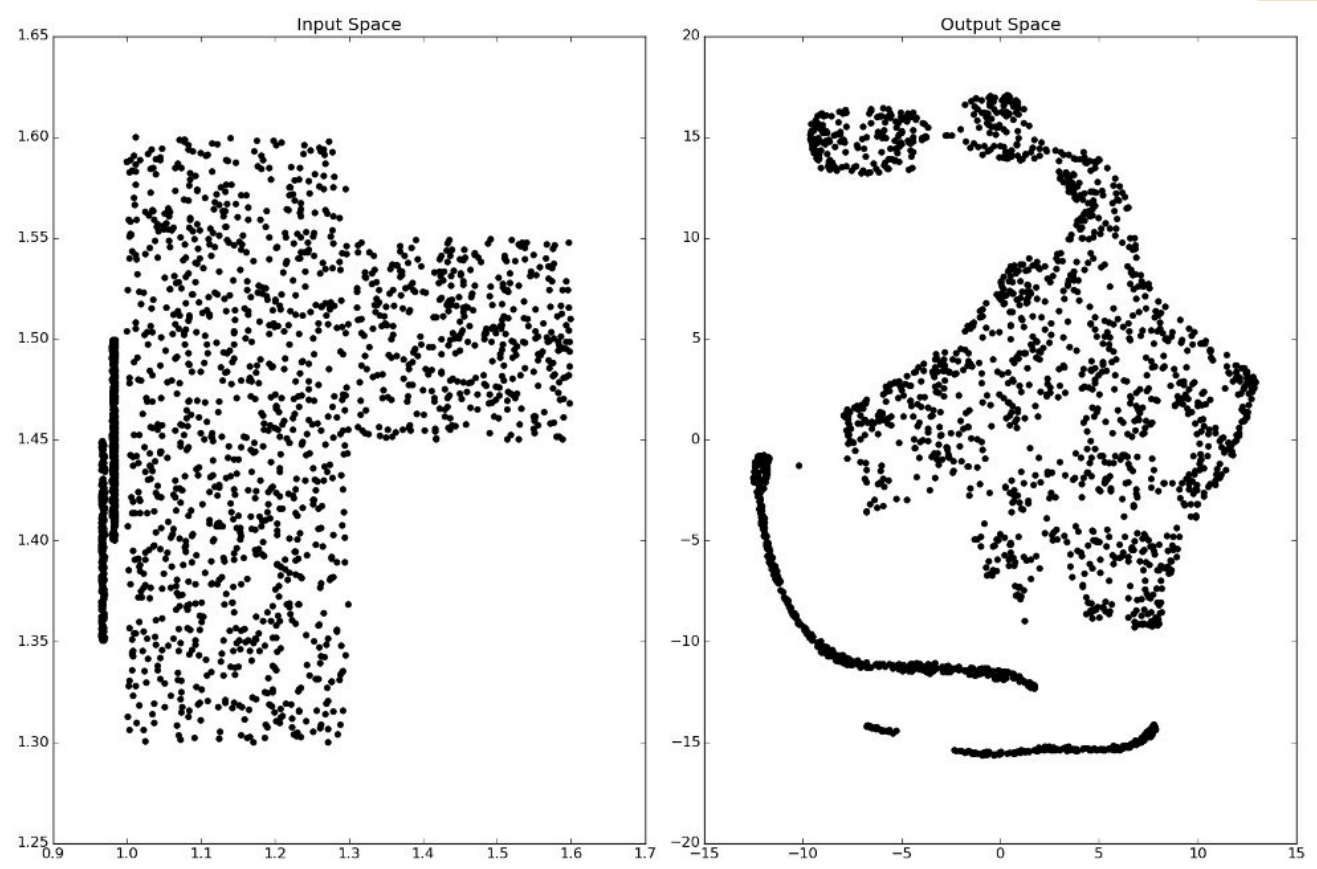
t-SNE - The Trick Explained

Gaussian activation vs. Student's t activation



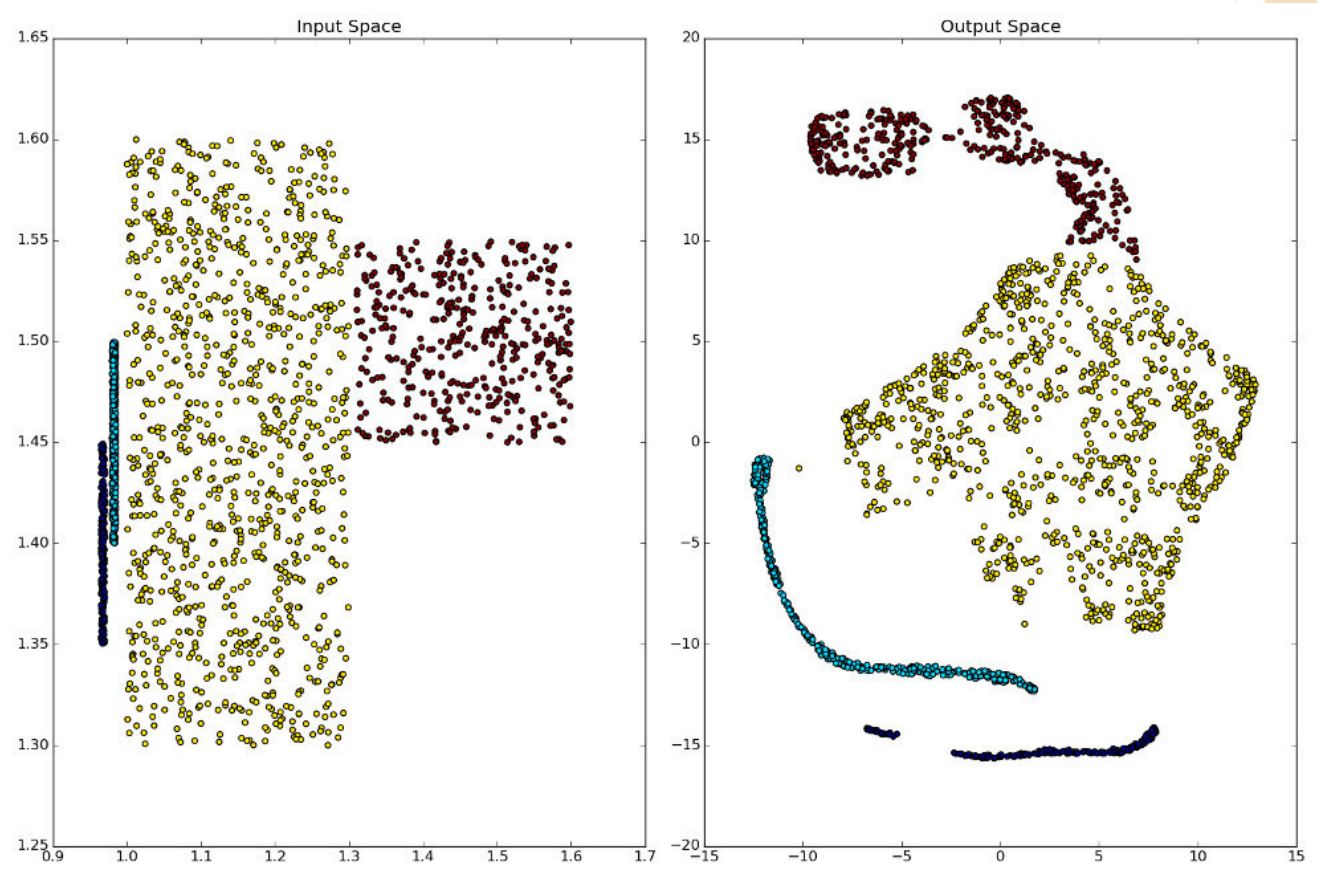
Non-neighbors are potentially pushed away.

Examples - Toy Problem (1)



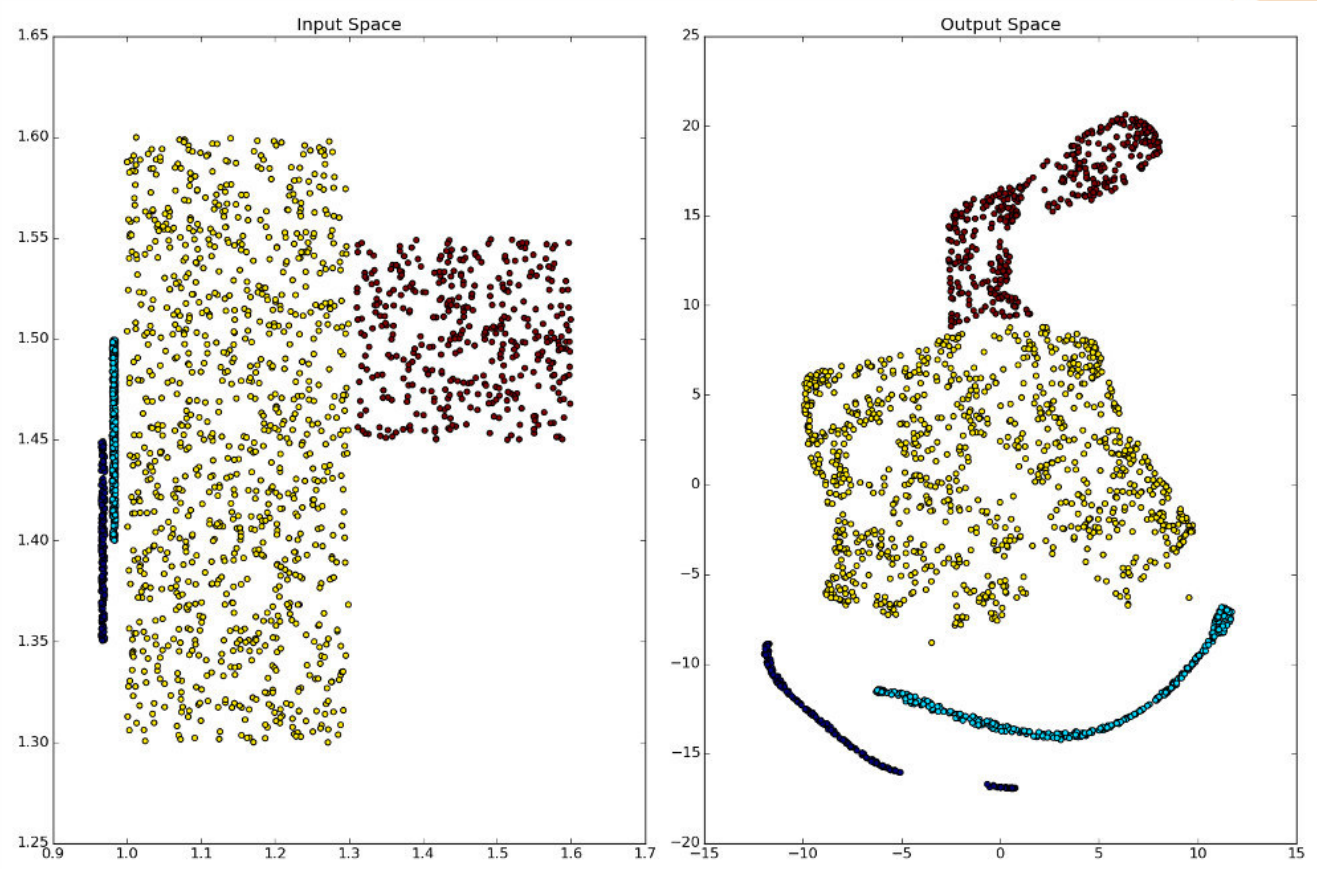
Different local densities were recognized an reflected in the output map!

Examples - Toy Problem (2)



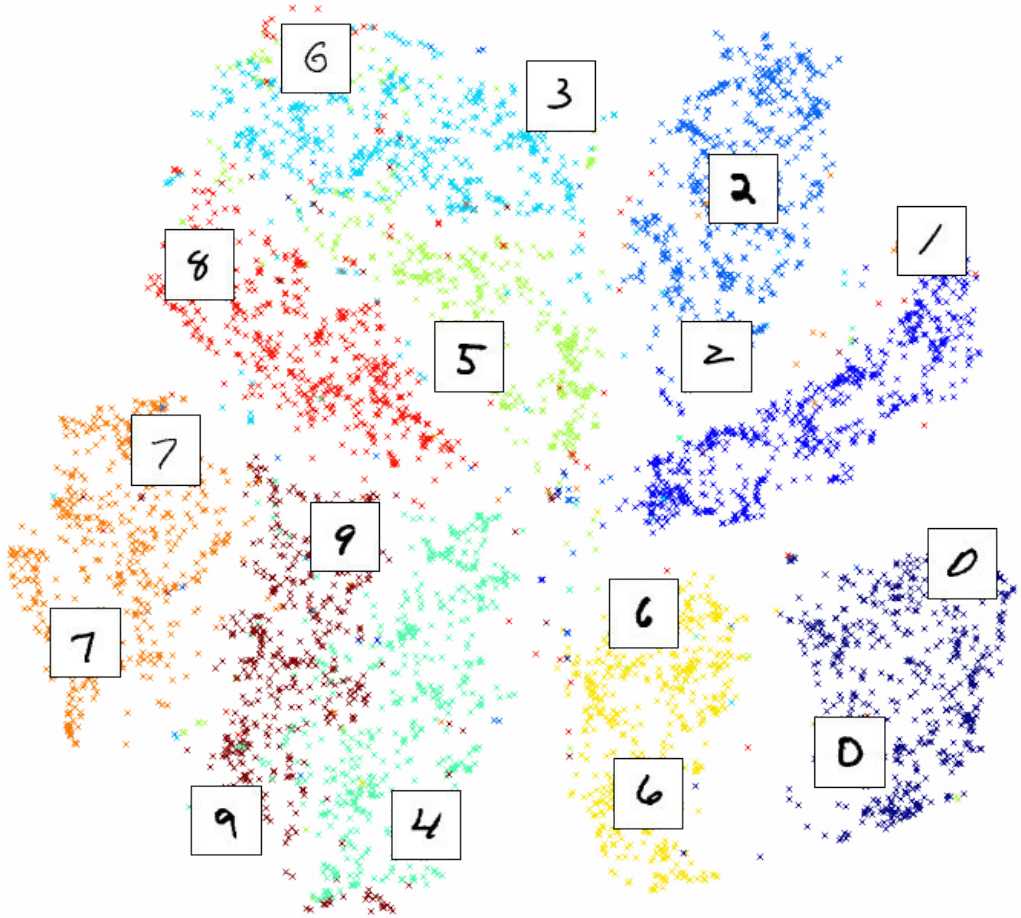
Input data actually comes from four different distributions...

Examples - Toy Problem (3)



A different run of the t-SNE on the very same input data potentially provides a very different result!

Examples - MNIST



Projection of a subset of the famous MNIST dataset collecting a very big number of hand written digits.

Examples - Waveforms

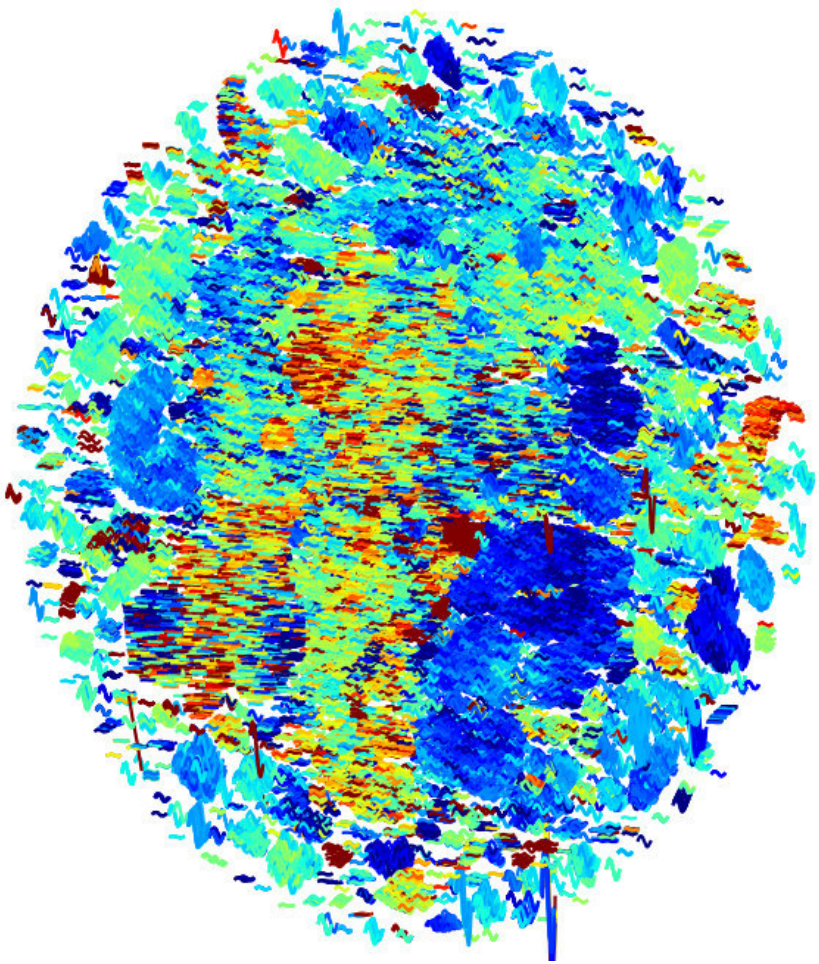
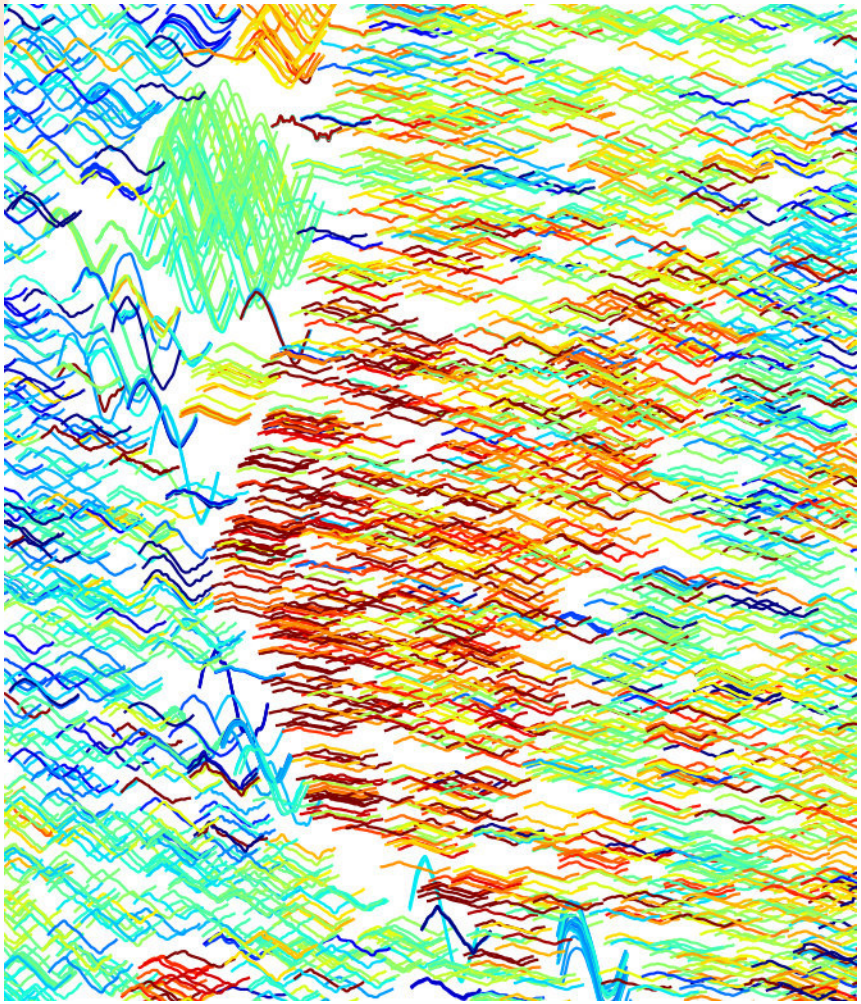


Figure : color by source location

Examples - Waveforms (map details)



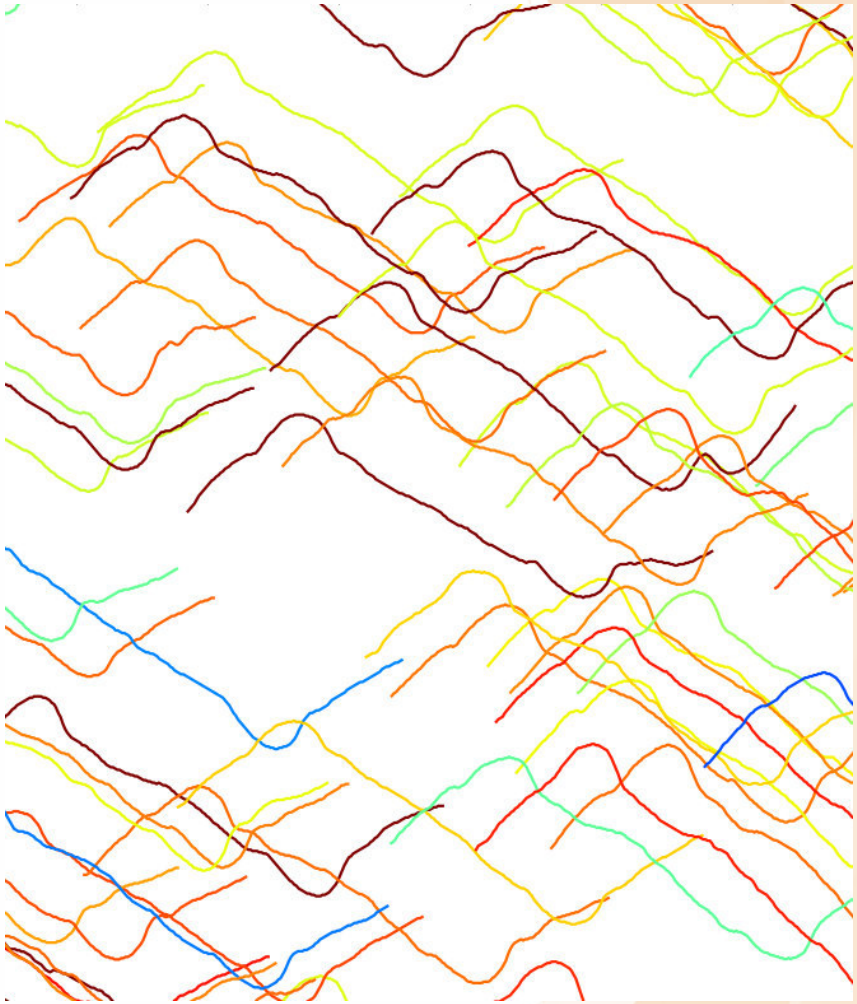


Figure : color by source location

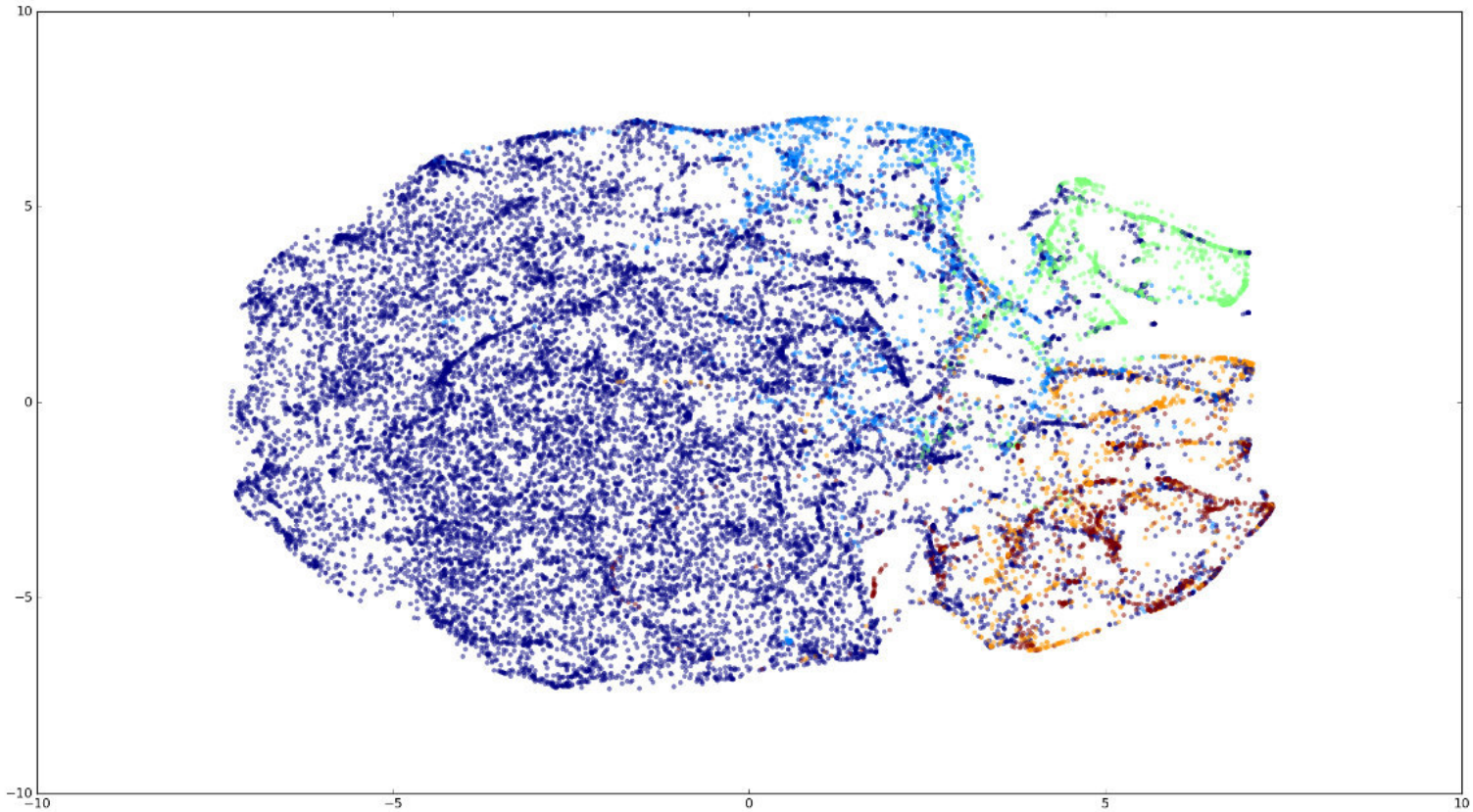
CAMLIN

Other ways of using t-SNE

- asses how extracted features describe your data
- asses how NNs transform your data: will the classifier at the top of the net be able to separate data?
- asses whether classifiers are doing well or not...

Other ways of using t-SNE

Project onto a 2D place the output of 15 different classifiers trying to distinguish 5 different classes of input data



Conclusions

Good Things

- tSNE performs very well in recognizing local structures
- maps produced with tSNE looks much better than those obtained with other techniques
- tSNE was used on many different problems proving very robust

Bad Things

- Does not scale well with high number of points: approximated version were proposed (Barnes-Hut tSNE)
- **It is not possible to directly project a new point onto an already computed map**
  (...but this is something Camlin Italy solved about one year ago!)

CAMLIN

L10: Linear discriminants analysis

Linear discriminant analysis, two classes

Linear discriminant analysis, C classes

LDA vs. PCA

Limitations of LDA

Variants of LDA

Other dimensionality reduction methods

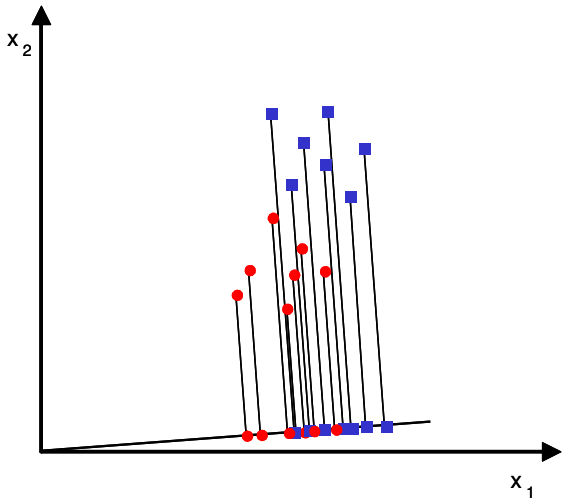
Linear discriminant analysis, two-classes

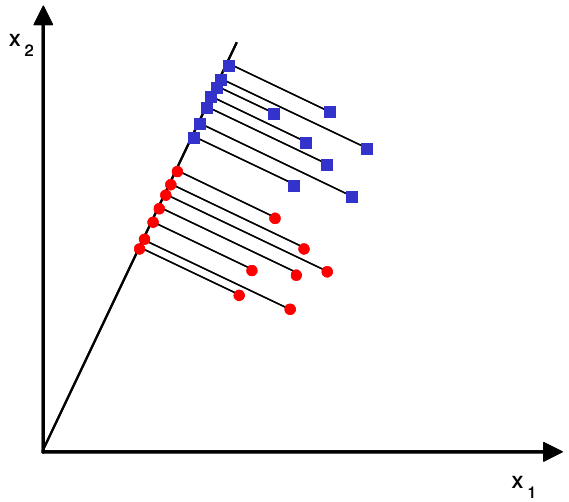
Objective

- LDA seeks to reduce dimensionality while preserving as much of the class discriminatory information as possible
- Assume we have a set of D-dimensional samples $\{x^{(1}, x^{(2}, \dots x^{(N}\}$, N_1 of which belong to class ω_1, and N_2 to class ω_2
- We seek to obtain a scalar y by projecting the samples x onto a line

$$y = w^T x$$

- Of all the possible lines we would like to select the one that maximizes the separability of the scalars





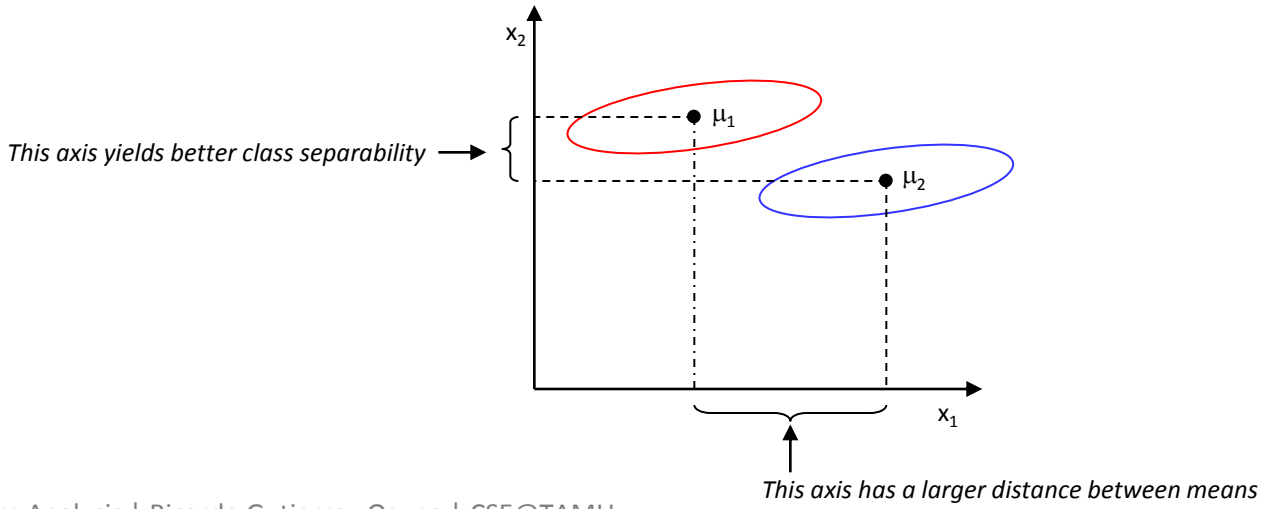
- In order to find a good projection vector, we need to define a measure of separation
- The mean vector of each class in x-space and y-space is

$$\mu_i = \frac{1}{N_i}\sum_{x\in\omega_i} x \text{ and } \tilde{\mu}_i = \frac{1}{N_i}\sum_{y\in\omega_i} y = \frac{1}{N_i}\sum_{x\in\omega_i} w^T x = w^T\mu_i$$

- We could then choose the distance between the projected means as our objective function

$$J(w) = |\tilde{\mu}_1 - \tilde{\mu}_2| = |w^T(\mu_1 - \mu_2)|$$

  - However, the distance between projected means is not a good measure since it does not account for the standard deviation within classes



Fisher's solution

- Fisher suggested maximizing the difference between the means, normalized by a measure of the within-class scatter
- For each class we define the scatter, an equivalent of the variance, as

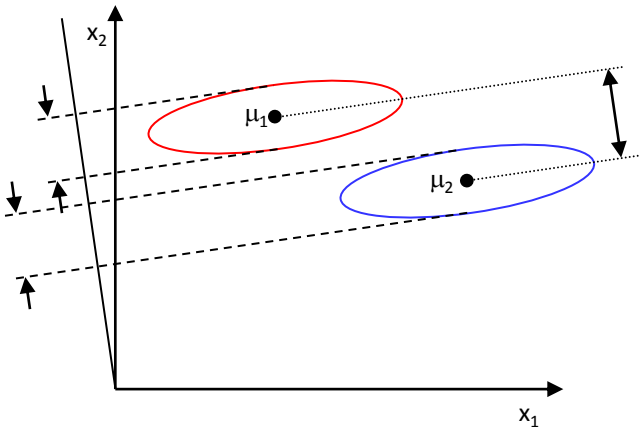
$$\tilde{s}_i^2 = \sum_{y \in \omega_i} (y - \tilde{\mu}_i)^2$$

  - where the quantity $(\tilde{s}_1^2 + \tilde{s}_2^2)$ is called the within-class scatter of the projected examples

- The Fisher linear discriminant is defined as the linear function $w^T x$ that maximizes the criterion function

$$J(w) = \frac{|\tilde{\mu}_1 - \tilde{\mu}_2|^2}{\tilde{s}_1^2 + \tilde{s}_2^2}$$

- Therefore, we are looking for a projection where examples from the same class are projected very close to each other and, at the same time, the projected means are as farther apart as possible



To find the optimum w^*, we must express $J(w)$ as a function of w

- First, we define a measure of the scatter in feature space x

$$S_i = \sum_{x \in \omega_i} (x - \mu_i)(x - \mu_i)^T$$

$$S_1 + S_2 = S_W$$

  - where S_W is called the within-class scatter matrix

- The scatter of the projection y can then be expressed as a function of the scatter matrix in feature space x

$$\tilde{s}_i^2 = \sum_{y \in \omega_i} (y - \tilde{\mu}_i)^2 = \sum_{x \in \omega_i} (w^T x - w^T \mu_i)^2 =$$
$$= \sum_{x \in \omega_i} w^T (x - \mu_i)(x - \mu_i)^T w = w^T S_i w$$

$$\tilde{s}_1^2 + \tilde{s}_2^2 = w^T S_W w$$

- Similarly, the difference between the projected means can be expressed in terms of the means in the original feature space

$$(\tilde{\mu}_1 - \tilde{\mu}_2)^2 = (w^T \mu_1 - w^T \mu_2)^2 = w^T \underbrace{(\mu_1 - \mu_2)(\mu_1 - \mu_2)^T}_{S_B} w = w^T S_B w$$

  - The matrix S_B is called the between-class scatter. Note that, since S_B is the outer product of two vectors, its rank is at most one

- We can finally express the Fisher criterion in terms of S_W and S_B as

$$J(w) = \frac{w^T S_B w}{w^T S_W w}$$

- To find the maximum of $J(w)$ we derive and equate to zero

$$\frac{d}{dw}[J(w)] = \frac{d}{dw}\left[\frac{w^T S_B w}{w^T S_W w}\right] = 0 \Rightarrow$$

$$[w^T S_W w]\frac{d[w^T S_B w]}{dw} - [w^T S_B w]\frac{d[w^T S_W w]}{dw} = 0 \Rightarrow$$

$$[w^T S_W w]2S_B w - [w^T S_B w]2S_W w = 0$$

- Dividing by $w^T S_W w$

$$\left[\frac{w^T S_W w}{w^T S_W w}\right] S_B w - \left[\frac{w^T S_B w}{w^T S_W w}\right] S_W w = 0 \Rightarrow$$

$$S_B w - J S_W w = 0 \Rightarrow$$

$$S_W^{-1} S_B w - J w = 0$$

- Solving the generalized eigenvalue problem ($S_W^{-1} S_B w = J w$) yields

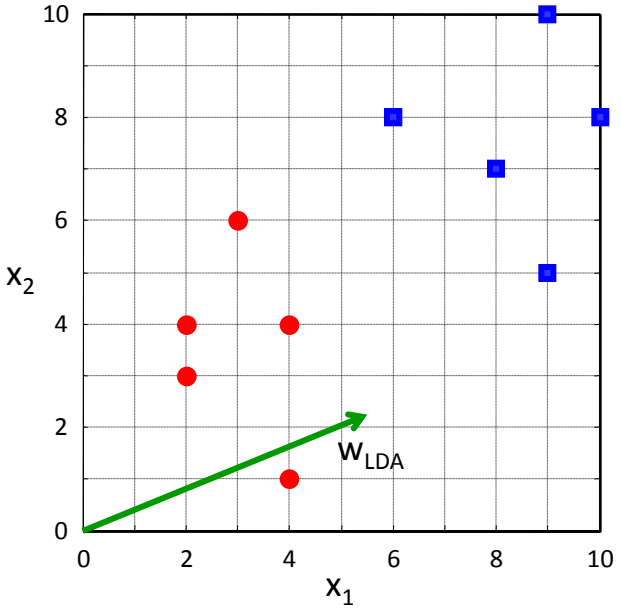
$$w^* = \arg\max\left[\frac{w^T S_B w}{w^T S_W w}\right] = S_W^{-1}(\mu_1 - \mu_2)$$

- This is know as Fisher's linear discriminant (1936), although it is not a discriminant but rather a specific choice of direction for the projection of the data down to one dimension

Example

Compute the LDA projection for the following 2D dataset

$$X1 = \{(4,1), (2,4), (2,3), (3,6), (4,4)\}$$
$$X2 = \{(9,10), (6,8), (9,5), (8,7), (10,8)\}$$



SOLUTION (by hand)

- The class statistics are

$$S_1 = \begin{bmatrix} .8 & -.4 \\ & 2.64 \end{bmatrix} \quad S_2 = \begin{bmatrix} 1.84 & -.04 \\ & 2.64 \end{bmatrix}$$

$$\mu_1 = [3.0\ 3.6]^T; \quad \mu_2 = [8.4\ 7.6]^T$$

- The within- and between-class scatter are

$$S_B = \begin{bmatrix} 29.16 & 21.6 \\ & 16.0 \end{bmatrix} \quad S_W = \begin{bmatrix} 2.64 & -.44 \\ & 5.28 \end{bmatrix}$$

- The LDA projection is then obtained as the solution of the generalized eigenvalue problem

$$S_W^{-1} S_B v = \lambda v \Rightarrow \left| S_W^{-1} S_B - \lambda I \right| = 0 \Rightarrow \begin{vmatrix} 11.89 - \lambda & 8.81 \\ 5.08 & 3.76 - \lambda \end{vmatrix} = 0 \Rightarrow \lambda = 15.65$$

$$\begin{bmatrix} 11.89 & 8.81 \\ 5.08 & 3.76 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = 15.65 \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} \Rightarrow \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = \begin{bmatrix} .91 \\ .39 \end{bmatrix}$$

- Or directly by

$$w^* = S_W^{-1}(\mu_1 - \mu_2) = [-.91\ -.39]^T$$

LDA, C classes

Fisher's LDA generalizes gracefully for C-class problems

- Instead of one projection y, we will now seek $(C-1)$ projections $[y_1, y_2, \dots y_{C-1}]$ by means of $(C-1)$ projection vectors w_i arranged by columns into a projection matrix $W = [w_1|w_2|\dots|w_{C-1}]$:

$$y_i = w_i^T x \Rightarrow y = W^T x$$

Derivation

- The within-class scatter generalizes as

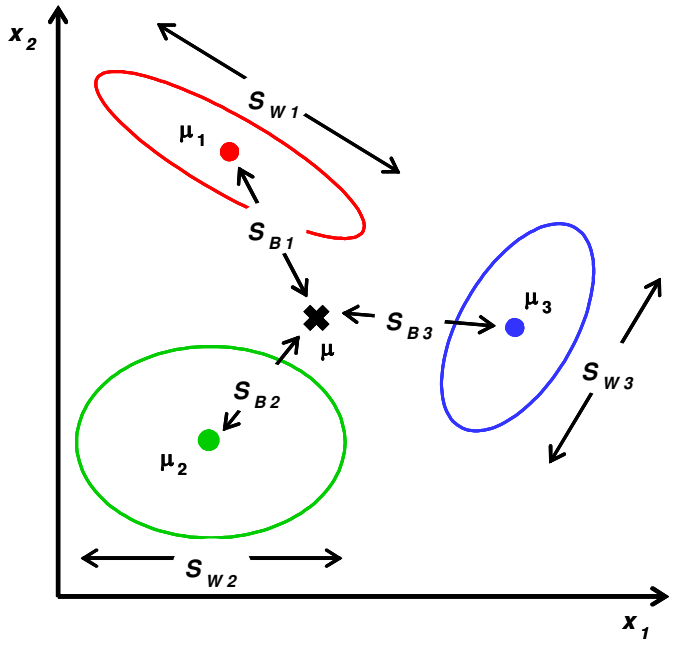
$$S_W = \sum_{i=1}^{C} S_i$$

  - where $S_i = \sum_{x \in \omega_i} (x - \mu_i)(x - \mu_i)^T$ and $\mu_i = \frac{1}{N_i} \sum_{x \in \omega_i} x$

- And the between-class scatter becomes

$$S_B = \sum_{i=1}^{C} N_i (\mu_i - \mu)(\mu_i - \mu)^T$$

  - where $\mu = \frac{1}{N} \sum_{\forall x} x = \frac{1}{N} \sum_{i=1}^{C} N_i \mu_i$



- Matrix $S_T = S_B + S_W$ is called the <u>total scatter</u>

- Similarly, we define the mean vector and scatter matrices for the projected samples as

$$\tilde{\mu}_i = \frac{1}{\mathrm{N}_i}\sum_{y\in\omega_i} y \qquad \tilde{S}_W = \sum_{i=1}^{C}\sum_{y\in\omega_i}(y-\tilde{\mu}_i)(y-\tilde{\mu}_i)^T$$

$$\tilde{\mu} = \frac{1}{N}\sum_{\forall y} y \qquad \tilde{S}_B = \sum_{i=1}^{C} N_i(\tilde{\mu}_i-\tilde{\mu})(\tilde{\mu}_i-\tilde{\mu})^T$$

- From our derivation for the two-class problem, we can write

$$\tilde{S}_W = W^T S_W W$$
$$\tilde{S}_B = W^T S_B W$$

- Recall that we are looking for a projection that maximizes the ratio of between-class to within-class scatter. Since the projection is no longer a scalar (it has $C-1$ dimensions), we use the determinant of the scatter matrices to obtain a scalar objective function

$$J(W) = \frac{\left|\tilde{S}_B\right|}{\left|\tilde{S}_W\right|} = \frac{|W^T S_B W|}{|W^T S_W W|}$$

- And we will seek the projection matrix W^* that maximizes this ratio

– It can be shown that the optimal projection matrix W^* is the one whose columns are the eigenvectors corresponding to the largest eigenvalues of the following generalized eigenvalue problem

$$W^* = [w_1^* | w_2^* | \dots w_{C-1}^*] = \arg\max \frac{|W^T S_B W|}{|W^T S_W W|} \Rightarrow (S_B - \lambda_i S_W) w_i^* = 0$$

NOTES

– S_B is the sum of C matrices of rank ≤ 1 and the mean vectors are constrained by $\frac{1}{C}\sum_{i=1}^{C} \mu_i = \mu$
  - Therefore, S_B will be of rank $(C-1)$ or less
  - This means that only $(C-1)$ of the eigenvalues λ_i will be non-zero
– The projections with maximum class separability information are the eigenvectors corresponding to the largest eigenvalues of $S_W^{-1} S_B$
– LDA can be derived as the Maximum Likelihood method for the case of normal class-conditional densities with equal covariance matrices

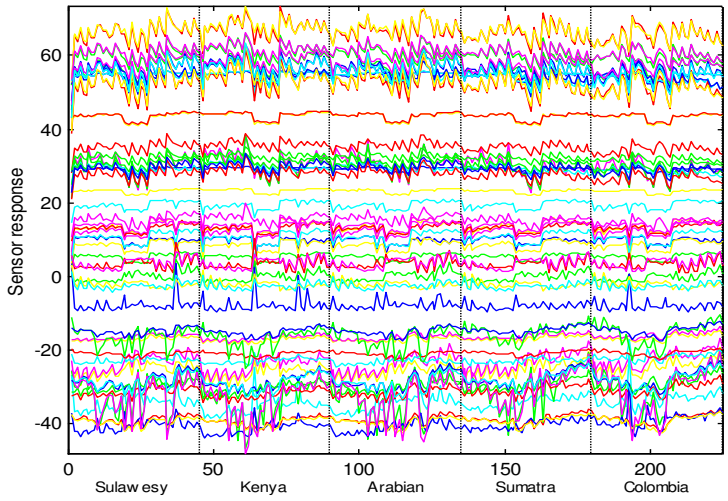
LDA vs. PCA

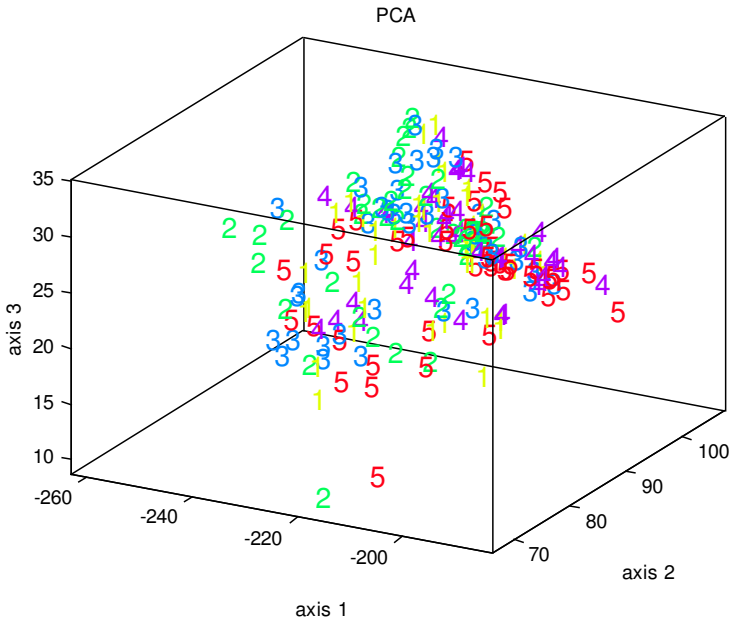
This example illustrates the performance of PCA and LDA on an odor recognition problem

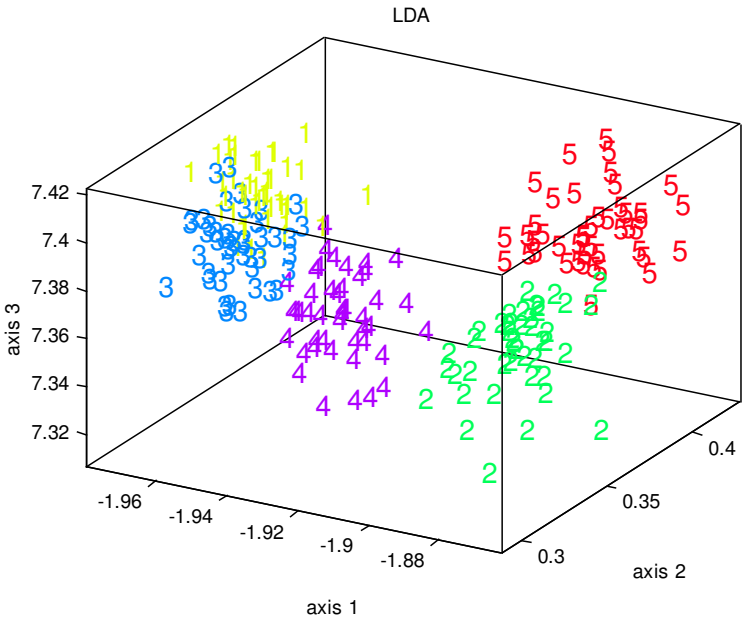
- Five types of coffee beans were presented to an array of gas sensors
- For each coffee type, 45 "sniffs" were performed and the response of the gas sensor array was processed ir order to obtain a 60-dimensional feature vector

Results

- From the 3D scatter plots it is clear that LDA outperforms PCA in terms of class discrimination
- This is one example where the discriminatory information is not aligned with the direction of maximum variance







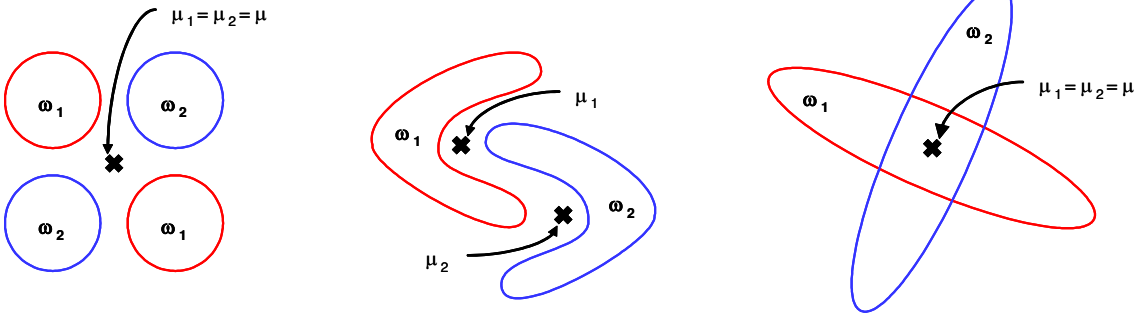
Limitations of LDA

LDA produces at most $C - 1$ feature projections

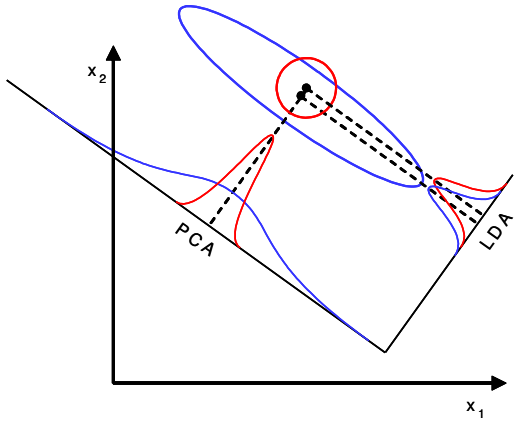
- If the classification error estimates establish that more features are needed, some other method must be employed to provide those additional features

LDA is a parametric method (it assumes unimodal Gaussian likelihoods)

- If the distributions are significantly non-Gaussian, the LDA projections may not preserve complex structure in the data needed for classification



LDA will also fail if discriminatory information is not in the mean but in the variance of the data



Evaluating Hypotheses – Lecture Overview

- Measures of classification performance
  - Classification Error Rate
  - UAR
  - Recall, Precision, Confusion Matrix
  - Imbalanced Datasets
  - Overfitting
  - Cross-validation

- Estimating hypothesis accuracy
  - Sample Error vs. True Error
  - Confidence Intervals
  - Binomial and Normal Distributions

- Comparing Learning Algorithms
  - t-test

Classification Measures – Confusion Matrix

	Class 1 Predicted	Class 2 Predicted
Class 1 Actual	TP	FN
Class 2 Actual	FP	TN

- Class 1: Positive
- Class 2: Negative
- TP: True Positive
- FN: False Negative
- FP: False Positive
- TN: True Negative

- Visualisation of the performance of an algorithm
- Allows easy identification of confusion between between classes
  e.g. one class is commonly mislabelled as the other
- Most performance measures are computed from the confusion matrix

Classification Measures – Classification Rate

	Class 1 Predicted	Class 2 Predicted
Class 1 Actual	TP	FN
Class 2 Actual	FP	TN

- Class 1: Positive
- Class 2: Negative
- TP: True Positive
- FN: False Negative
- FP: False Positive
- TN: True Negative

- Classification Rate / Accuracy: $$\frac{TP + TN}{TP + TN + FP + FN}$$

- Number of correctly classified examples divided by the total number of examples
- Classification Error = 1 – Classification Rate
- Classification Rate = Pr(correct classification)

Classification Measures – Recall

	Class 1 Predicted	Class 2 Predicted
Class 1 Actual	TP	FN
Class 2 Actual	FP	TN

- Class 1: Positive
- Class 2: Negative
- TP: True Positive
- FN: False Negative
- FP: False Positive
- TN: True Negative

- Recall: $\frac{TP}{TP + FN}$

- Number of correctly classified positive examples divided by the total number of positive examples
- High recall: The class is correctly recognised (small number of FN)
- Recall = Pr(correctly classified | positive example)

Classification Measures – Precision

	Class 1 Predicted	Class 2 Predicted
Class 1 Actual	TP	FN
Class 2 Actual	FP	TN

- Class 1: Positive
- Class 2: Negative
- TP: True Positive
- FN: False Negative
- FP: False Positive
- TN: True Negative

- Precision: $\frac{TP}{TP + FP}$
- Number of correctly classified positive examples divided by the total number of predicted positive examples
- High precision: An example labeled as positive is indeed positive (small number of FP)
- Precision = Pr(positive example | example is classified as positive)

Classification Measures – Recall/Precision

	Class 1 Predicted	Class 2 Predicted
Class 1 Actual	TP	FN
Class 2 Actual	FP	TN

- Class 1: Positive
- Class 2: Negative
- TP: True Positive
- FN: False Negative
- FP: False Positive
- TN: True Negative

- High recall, low precision: Most of the positive examples are correctly recognised (low FN) but there are a lot of false positives.

- Low recall, high precision: We miss a lot of positive examples (high FN) but those we predict as positive are indeed positive (low FP).

Classification Measures – F1 Measure/Score

- It is useful to have one number to measure the performance of the classifier
- $F_\alpha = (1 + \alpha^2) \frac{Precision * Recall}{\alpha^2 * Precision + recall}$
- When α=1 → $F_1 = 2 \frac{Precision * Recall}{Precision + recall}$

Classification Measures – UAR

	Class 1 Predicted	Class 2 Predicted
Class 1 Actual	TP	FN
Class 2 Actual	FP	TN

- Class 1: Positive
- Class 2: Negative
- TP: True Positive
- FN: False Negative
- FP: False Positive
- TN: True Negative

- We compute recall for class1 (R1) and for class2 (R2).
- Unweighted Average Recall (UAR) = mean(R1, R2)

Classification Measures – Extension to Multiple Classes

	Class 1 Predicted	Class 2 Predicted	Class 3 Predicted
Class 1 Actual	TP	FN	FN
Class 2 Actual	FP	TN	?
Class 3 Actual	FP	?	TN

- In the multiclass case it is still very useful to compute the confusion matrix.
- We can define one class as positive and the other as negative.
- We can compute the performance measures in exactly the same way.

- CR = number of correctly classified examples (trace) divided by the total number of examples.
- Recall and precision and F1 are still computed for each class.
- UAR = mean(R1, R2, R3,…, RN)

Classification Measures – Balanced Dataset

	Class 1 Predicted	Class 2 Predicted
Class 1 Actual	70	30
Class 2 Actual	10	90

- CR: 80%
- Recall (cl.1): 70%
- Precision (cl.1): 87.5%
- F1 (cl.1): 77.8%
- UAR: 80%
- Recall (cl.2): 90%
- Precision (cl.2): 75%
- F1 (cl.2): 81.8%

- Balanced Dataset: The number of examples in each class are similar
- All measures result in similar performance

Classification Measures – Imbalanced Dataset
Case 1: Both classifiers are good

	Class 1 Predicted	Class 2 Predicted
Class 1 Actual	700	300
Class 2 Actual	10	90

- CR: 71.8%
- Recall (cl.1): 70%
- Precision (cl.1): 98.6%
- F1 (cl.1): 81.9%
- UAR: 80%
- Recall (cl.2): 90%
- Precision (cl.2): 23.1%
- F1 (cl.2): 36.8%

- Imbalanced Dataset: Classes are not equally represented
- CR goes down, is affected a lot by the majority class
- Precision (and F1) for Class 2 is significantly affected –
  - 30% of class1 examples are misclassified→ leads to a higher number of FP than TN due to imbalance

Classification Measures – Imbalanced Dataset
Case 2: One classifier is useless

	Class 1 Predicted	Class 2 Predicted
Class 1 Actual	700	300
Class 2 Actual	100	0

- CR: 70%
- Recall (cl.1): 70%
- Precision (cl.1): 87.5%
- F1 (cl.1): 77.8%
- UAR: 35%
- Recall (cl.2): 0%
- Precision (cl.2): 0%
- F1 (cl.2): Not defined

- CR is misleading, one classifier is useless.
- F1 for class2 and UAR tell us that something is wrong.
- UAR also detects that there is a problem.

Classification Measures – Imbalanced Dataset Conclusions

- CR can be misleading, simply follows the performance of the majority class
- UAR is useful and can help to detect that one or more classifiers are not good but it does not give us any information about FP
- F1 is useful as well but is also affected by the class imbalance problem

  - We are not sure if the low score is due to one/more classifiers being useless or class imbalance

- That's why we should always have a look at the confusion matrix

Classification Measures – Imbalanced Dataset Some solutions

	Class 1 Predicted	Class 2 Predicted
Class 1 Actual	700	300
Class 2 Actual	10	90

Divide by the total number of examples per class →

	Class 1 Predicted	Class 2 Predicted
Class 1 Actual	0.7	0.3
Class 2 Actual	0.1	0.9

- Report performance ALSO on the "normalised matrix"

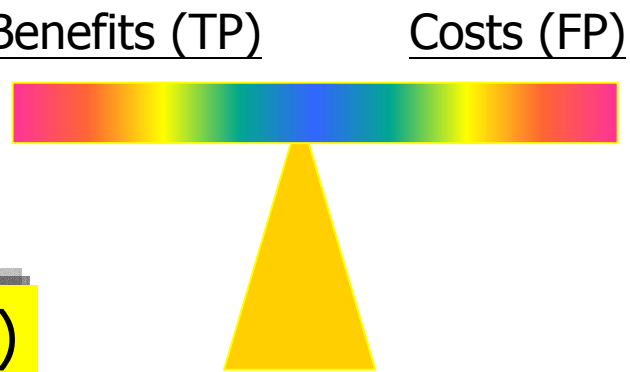
- CR: 71.8%
- Recall (cl.1): 70%
- Precision (cl.1): 98.6%
- F1 (cl.1): 81.9%
- UAR: 80%
- Recall (cl.2): 90%
- Precision (cl.2): 23.1%
- F1 (cl.2): 36.8%

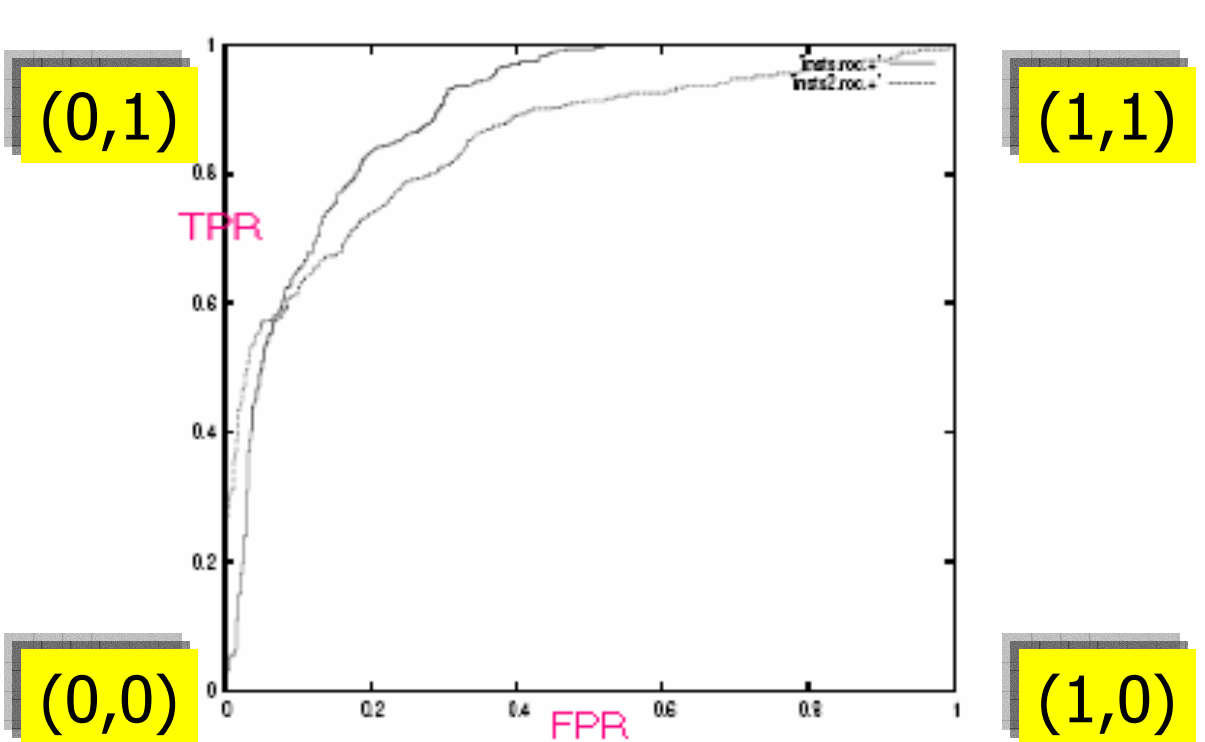
→

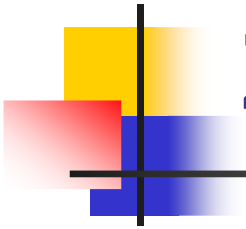
- CR: 80%
- Recall (cl.1): 70%
- Precision (cl.1): 87.5%
- F1 (cl.1): 77.8%
- UAR: 80%
- Recall (cl.2): 90%
- Precision (cl.2): 75%
- F1 (cl.2): 81.8%

ROC curve

- Y axis: TPR
  X axis: FPR





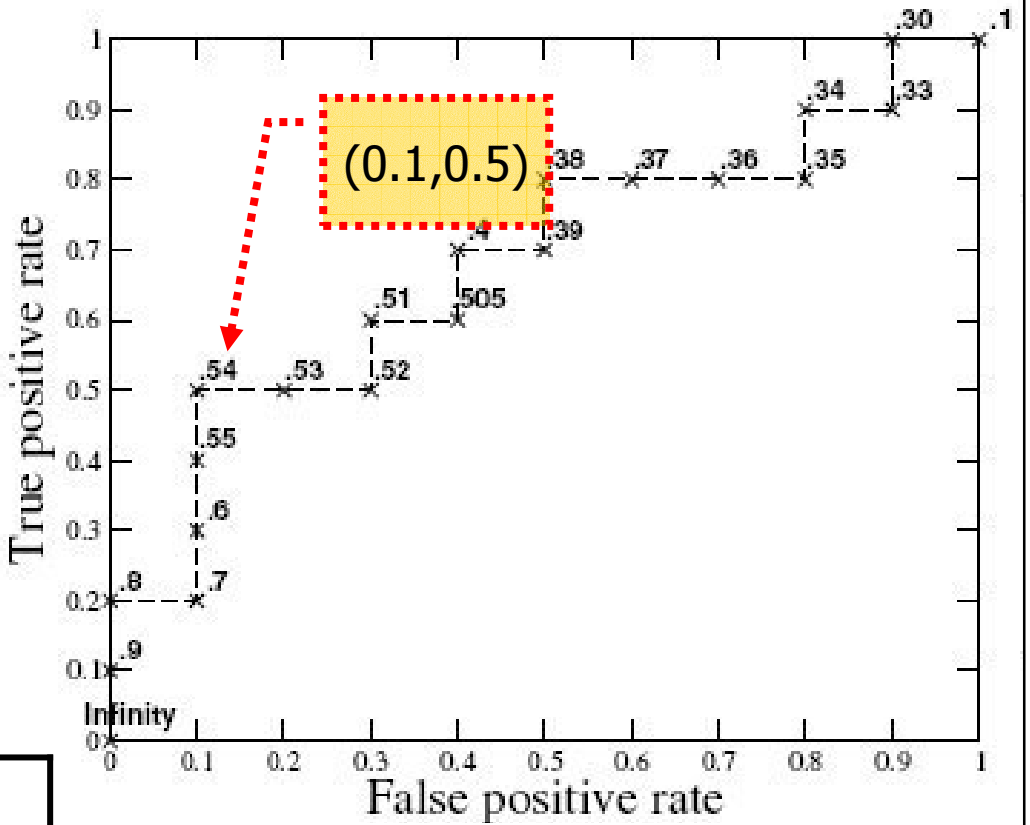


2. Create an ROC curve

- A ranking or scoring classifier can be used with a threshold to produce a binary classifier.
- If the classifier output is above the threshold, the classifier produces a Y, else a N.

Use thresholds to create ROC curve

Inst#	Class	Score	Inst#	Class	Score
1	p	.9	11	p	.4
2	p	.8	12	n	.39
3	n	.7	13	p	.38
4	p	.6	14	n	.37
5	p	.55	15	n	.36
6	p	.54	16	n	.35
7	n	.53	17	p	.34
8	n	.52	18	n	.33
9	p	.51	19	p	.30
10	n	.505	20	n	.1



If threshold =0.54 →
Numbers of Score ≧0.54 → 6

5	1	6
5	9	14
10	10	20

$$x: \frac{1}{10} = 0.1$$

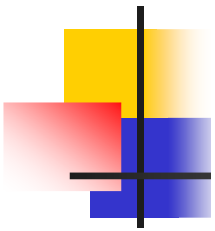
$$y: \frac{5}{10} = 0.5$$

f(i) : the probabilistic classifier's estimate that instance i is positive;
min and max, the smallest and largest values returned by f;
increment : the smallest difference between any two f values.

```
1:  for t = min to max by increment do
2:      FP ⇐ 0
3:      TP ⇐ 0
4:      for i ∈ L do
5:          if f(i) ≥ t then
6:              if i is a positive example then
7:                  TP ⇐ TP + 1
8:              else
9:                  FP ⇐ FP + 1
10:     Add point (FP/N, TP/P) to ROC curve
11: end
```

L Inputs: the set of test instances;

Conceptual Algorithm

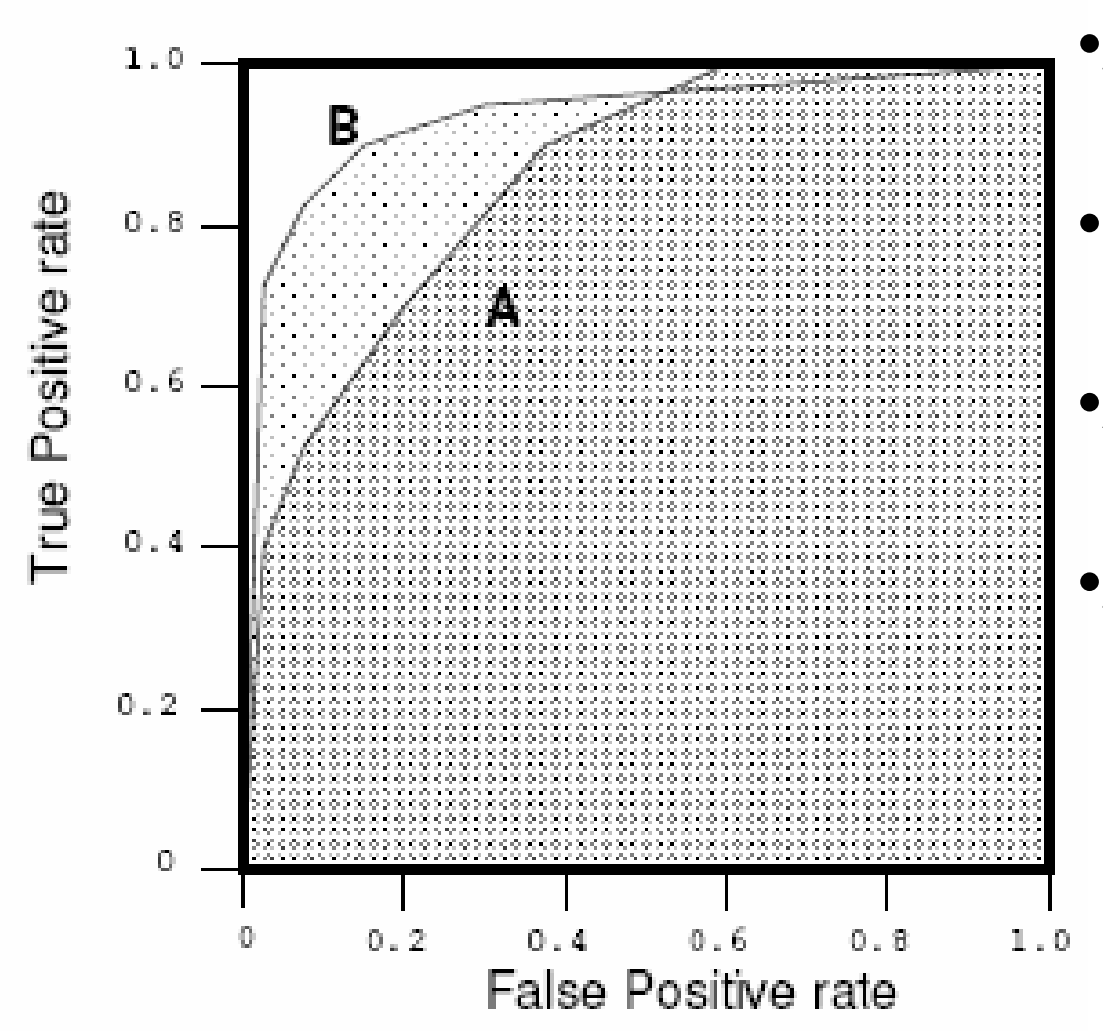


1: $L_{sorted} \Leftarrow L$ sorted decreasing by f scores
2: $FP \Leftarrow 0$
3: $TP \Leftarrow 0$
4: $R \Leftarrow \langle\rangle$
5: $f_{prev} \Leftarrow -\infty$
6: **for** $i \in L_{sorted}$ **do**
7: **if** $f(i) \neq f_{prev}$ **then**
8: ADD_POINT$\left(\left(\frac{FP}{N}, \frac{TP}{P}\right), R\right)$
9: $f_{prev} \Leftarrow f(i)$
10: **if** i is a positive example **then**
11: $TP \Leftarrow TP + 1$
12: **else**
13: $FP \Leftarrow FP + 1$
14: ADD_POINT$\left(\left(\frac{FP}{N}, \frac{TP}{P}\right), R\right)$
15: **end**

1: **subroutine** ADD_POINT(P, R)
2: push P onto R
3: **end subroutine**

Practical Algorithm

3. Area Under an ROC Curve (AUC)



- AUC (Bradley, 1997)
- Wilcoxon test of ranks
- Area : Classifier B > A
- Average performance → B > A

Moving from unranked to ranked evaluation

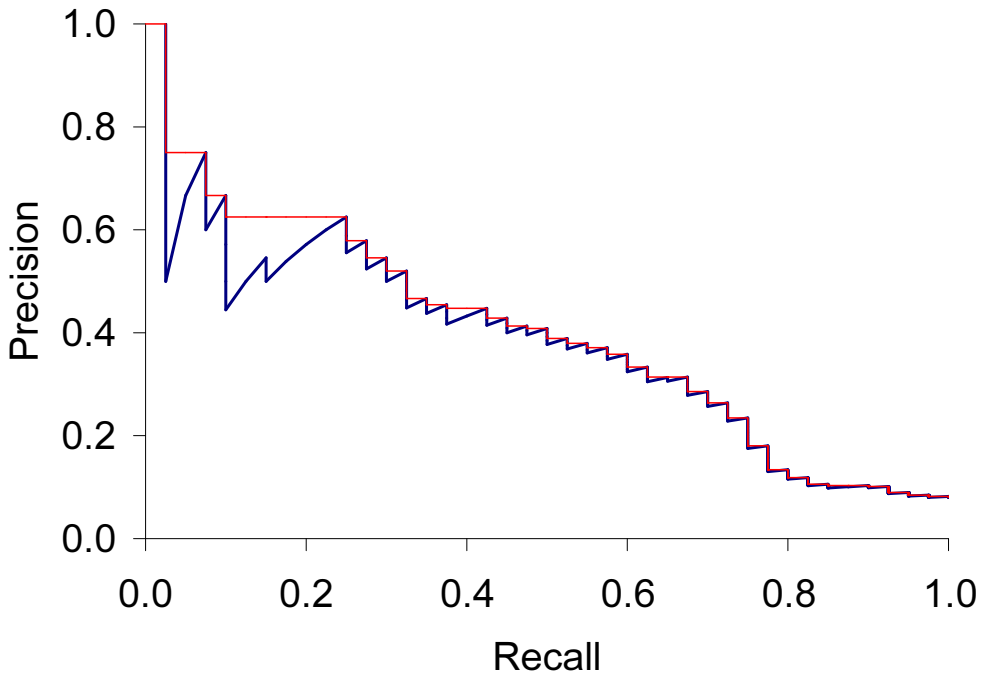
- Precision/recall/F are measures for unranked sets.
- We can easily turn set measures into measures of ranked lists.
- Just compute the set measure for each "prefix": the top 1, top 2, top 3, top 4 etc results
- This is called Precision/Recall at Rank
- Rank statistics give some indication of how quickly user will find relevant documents from ranked list

Precision/Recall @ Rank

Rank	Doc
1	d_{12}
2	d_{123}
3	d_4
4	d_{57}
5	d_{157}
6	d_{222}
7	d_{24}
8	d_{26}
9	d_{77}
10	d_{90}

- Blue documents are relevant
- P@n: P@3=0.33, P@5=0.2, P@8=0.25
- R@n: R@3=0.33, R@5=0.33, R@8=0.66

A precision-recall curve



- Each point corresponds to a result for the top k ranked hits ($k = 1, 2, 3, 4, \ldots$)
- Interpolation (in red): Take maximum of all future points
- Rationale for interpolation: The user is willing to look at more stuff if both precision and recall get better.

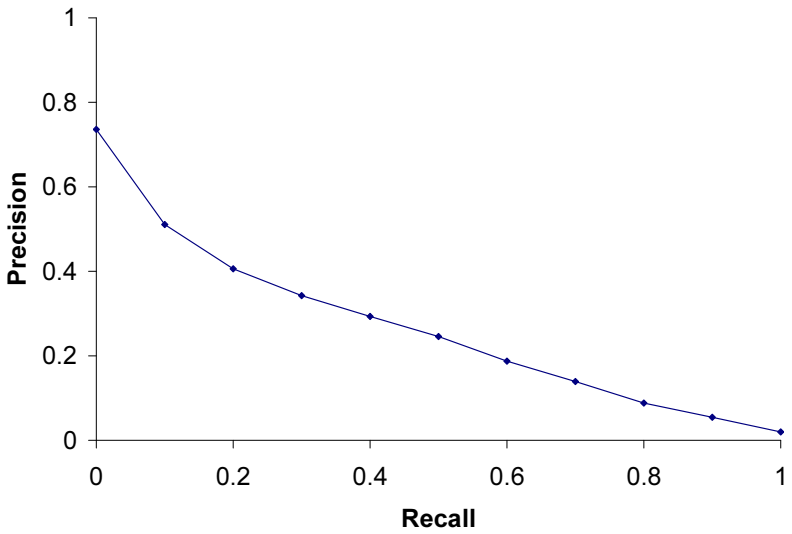
Another idea: Precision at Recall r

Rank	S1	S2
1	X	
2		X
3	X	
4		
5		X
6	X	X
7		X
8		X
9	X	
10	X	

$\rightarrow$

	S1	S2
p @ r 0.2	1.0	0.5
p @ r 0.4	0.67	0.4
p @ r 0.6	0.5	0.5
p @ r 0.8	0.44	0.57
p @ r 1.0	0.5	0.63

Averaged 11-point precision/recall graph



- Compute interpolated precision at recall levels 0.0, 0.1, 0.2, ...
- Do this for each of the queries in the evaluation benchmark
- Average over queries
- The curve is typical of performance levels at TREC (more later).

Mean Average Precision (MAP)

- Also called "average precision at seen relevant documents"
- Determine precision at each point when a new relevant document gets retrieved
- Use P=0 for each relevant document that was not retrieved
- Determine average for each query, then average over queries

$$MAP = \frac{1}{N} \sum_{j=1}^{N} \frac{1}{Q_j} \sum_{i=1}^{Q_j} P(doc_i)$$

with:

Q_j	number of relevant documents for query j
N	number of queries
$P(doc_i)$	precision at ith relevant document

Mean Average Precision: example ($MAP = \frac{0.564+0.623}{2} = 0.594$)

Query 1		
Rank		$P(doc_i)$
1	X	1.00
2		
3	X	0.67
4		
5		
6	X	0.50
7		
8		
9		
10	X	0.40
11		
12		
13		
14		
15		
16		
17		
18		
19		
20	X	0.25
AVG:		0.564

Query 2		
Rank		$P(doc_i)$
1	X	1.00
2		
3	X	0.67
4		
5		
6		
7		
8		
9		
10		
11		
12		
13		
14		
15	X	0.2
AVG:		0.623