Dimensionality Reduction

- Typical PR problems involve dozens, hundreds, or even thousands of features
- For example, in bioinformatics:
  - Tumor classification involves 10,000+ features: 30,000+ genes or 60,000+ transcripts
  - Email or text classification: 1,000+ features; 2-grams, 3-grams, 4-grams, etc.
- Now, we need:
  - *high* classification accuracy, while
  - *reducing* computational complexity (in training and classification phases)
- While it is not easy to achieve **both**,
  - we may *substantially* improve one
  - while *slightly* diminishing the other
- How is this achieved?
  - Dimensionality reduction

- Feature generation:
  - Using combinations of original features generate new (fewer) ones
  - Linear combinations: Linear Dimensionality Reduction (LDR)
  - Nonlinear combinations (e.g., kernels): Nonlinear Dimensionality Reduction (NLDR)
- Feature selection:
  - A subset of features from original ones that delivers reasonable (or even better) classification performance

Linear Dimensionality Reduction

- **Aim:** Use linear combinations of original features to generate new (fewer) features in a lower dimensional space
- Main Schemes:
  - Unsupervised:
    - Principal Component Analysis (PCA)
    - Singular Value Decomposition (SVD)
    - Independent Component Analysis (ICA)
    - Nonnegative Matrix Factorization (NMF)
    - Linear Autoencoder
  - Supervised:
    - Linear Discriminant Analysis (LDA):
      - Fisher's Discriminant Analysis (FDA) or homoscedastic
      - Heteroscedastic Discriminant Analysis (HDA)
      - Chernoff Discriminant Analysis (CDA)

Principal Component Analysis

- Seeks a projection that best represents the data… in a sum of squared-errors sense.
- It better applies to *unsupervised* classification
- Unsuitable for *supervised* classification (though it can be applied, if no other method is suitable)

- Given $D = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$
- What is the vector $\mathbf{x}_0$ that best represents D in the sum of squared-errors sense?
- That is, we want to *minimize*

$$J_0(\mathbf{x}_0) = \sum_{k=1}^{n} \left\| \mathbf{x}_0 - \mathbf{x}_k \right\|^2$$

- The solution to this problem is given by $\mathbf{m} = \mathbf{x}_0$, where

$$\mathbf{m} = \frac{1}{n}\sum_{k=1}^{n}\mathbf{x}_k$$

  is the sample mean
- The mean is a *zero-dimensional* representation of *D*
- But… it does not reveal info about the dispersion of the samples in *D*
- Let's seek a *one-dimensional* representation
- say, the line $\mathbf{x} = \mathbf{m} + a\mathbf{e}$

  where:
  - the line passes through **m**
  - **e** is a unit vector in the direction of the line, and
  - scalar *a* is the distance from **x** to **m**

- If we represent $\mathbf{x}_k$ by $\mathbf{m} + a_k \mathbf{e}$, then minimizing the sum of squared errors implies minimizing:

$$J_1(a_1,\ldots,a_n,\mathbf{e}) = \sum_{k=1}^{n} \left\|(\mathbf{m} + a_k\mathbf{e}) - \mathbf{x}_k\right\|^2$$

$$= \sum_{k=1}^{n} a_k^2 \left\|\mathbf{e}\right\|^2 - 2\sum_{k=1}^{n} a_k \mathbf{e}^t(\mathbf{x}_k - \mathbf{m}) + \sum_{k=1}^{n} \left\|\mathbf{x}_k - \mathbf{m}\right\|^2$$

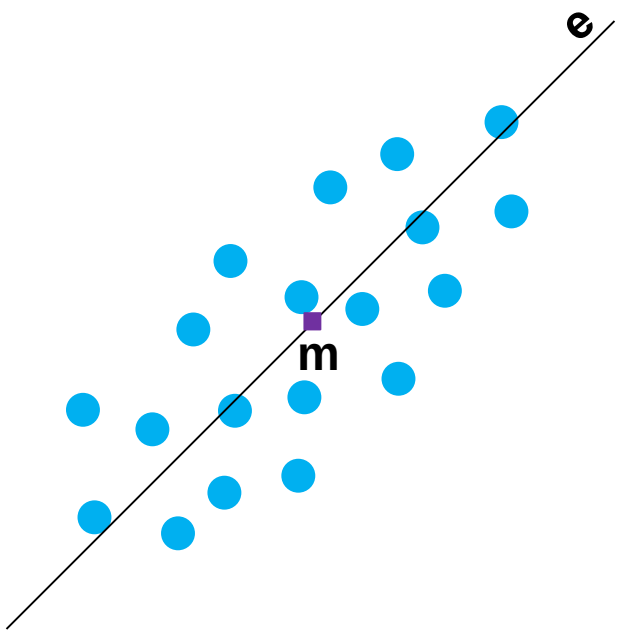
- whose solution is (where $\|\mathbf{e}\|^2 = 1$):

$$a_k = \mathbf{e}^t(\mathbf{x}_k - \mathbf{m})$$

Geometrically

- We project vector $\mathbf{x}_k$ onto the line, in the direction of **e** that passes through **m**
- What is the *best* direction of **e**?
- Lets introduce the *scatter matrix* **S**

$$\mathbf{S} = \sum_{k=1}^{n} (\mathbf{x}_k - \mathbf{m})(\mathbf{x}_k - \mathbf{m})^t$$



- Substituting a_k for the solution found, we have:

$$J_1(\mathbf{e}) = \sum_{k=1}^{n} a_k^2 - 2\sum_{k=1}^{n} a_k \mathbf{e}^t(\mathbf{x}_k - \mathbf{m}) + \sum_{k=1}^{n} \|\mathbf{x}_k - \mathbf{m}\|^2$$

$$= -\sum_{k=1}^{n} \left[\mathbf{e}^t(\mathbf{x}_k - \mathbf{m})\right]^2 + \sum_{k=1}^{n} \|\mathbf{x}_k - \mathbf{m}\|^2$$

$$= -\mathbf{e}^t\mathbf{S}\mathbf{e} + \sum_{k=1}^{n} \|\mathbf{x}_k - \mathbf{m}\|^2$$

- Then, the vector **e** that *minimizes* J_1 also *maximizes* $\mathbf{e}^t\mathbf{S}\mathbf{e}$
- Using the method of Lagrange multipliers, we differentiate:

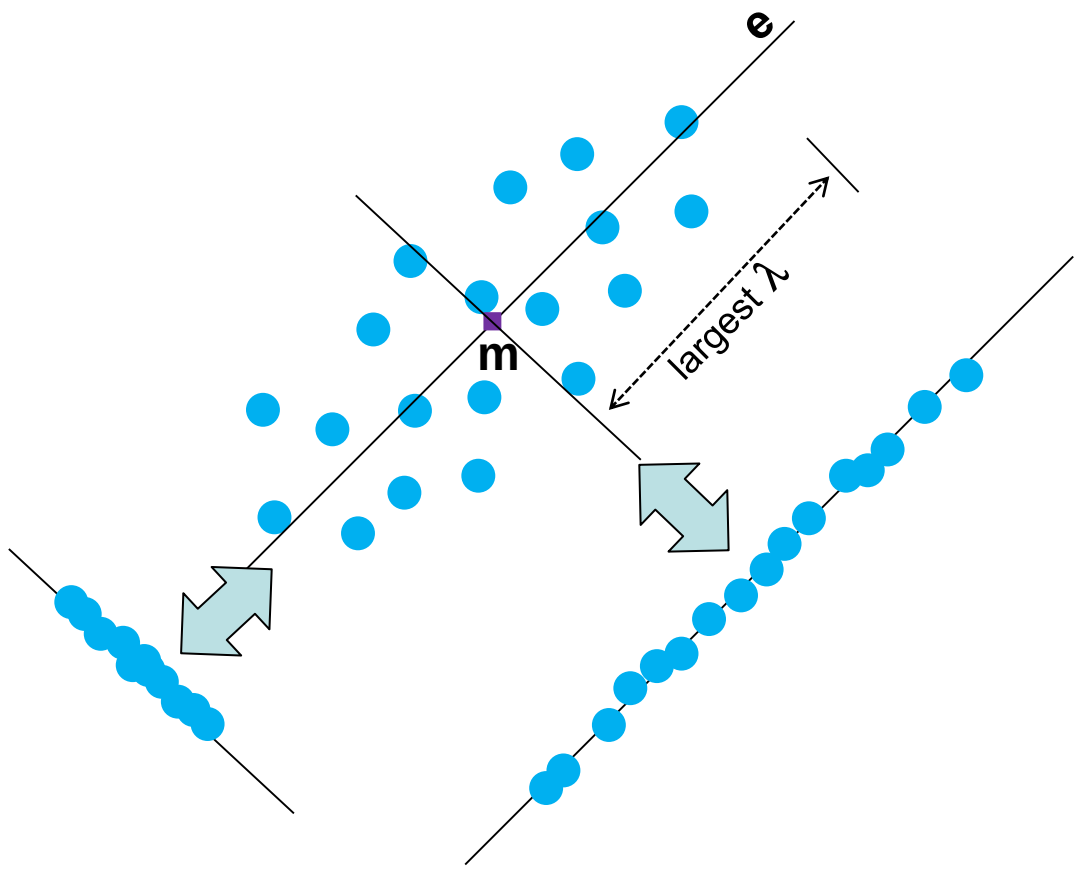
$$u = \mathbf{e}^t\mathbf{S}\mathbf{e} - \lambda(\mathbf{e}^t\mathbf{e} - 1) \qquad [\ \mathbf{e}^t\mathbf{e} - 1 = 0\ ]$$

- with respect to **e** obtaining

$$2\mathbf{S}\mathbf{e} - 2\lambda\mathbf{e} = \mathbf{0} \quad \text{or equivalently} \quad \mathbf{S}\mathbf{e} = \lambda\mathbf{e}$$

- Clearly, λ is an eigenvalue of **S**
- Since $\mathbf{e}^t\mathbf{S}\mathbf{e} = \lambda\mathbf{e}^t\mathbf{e} = \lambda$, then …
- maximizing $\mathbf{e}^t\mathbf{S}\mathbf{e}$ is equivalent to finding …

  the largest eigenvalue of **S**
- Thus, the **best** one-dimensional projection of the data in the sum-of-squared-error sense
  - is in the *direction* of the eigenvector **e**
  - having the *largest* eigenvalue λ
- Can be generalized to consider higher dimensions



- Then, the projection is onto a hyperplane:

$$\mathbf{x} = \mathbf{m} + \sum_{i=1}^{m} a_i \mathbf{e}_i$$

  and the aim is to find *m* vectors
- The solution is given by the *m* eigenvectors $\mathbf{e}_i$ of **S**
  having the *largest* eigenvalues λ_i
- Since **S** is *real* and *symmetric*,
  the eigenvectors are orthogonal
- They then form a basis in the *m*-dimensional space, and
  coefficients a_i are called the “principal components”

How to apply PCA?

Given $D = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$

- Find the scatter matrix $\mathbf{S}$
- Find the eigenvalue decomposition of $\mathbf{S}$:
  - $\mathbf{\Phi} = [\boldsymbol{\phi}_1, \boldsymbol{\phi}_2, \ldots, \boldsymbol{\phi}_d]$ are the eigenvectors
  - $\mathbf{\Lambda} = \text{diag}(\lambda_1, \lambda_2, \ldots, \lambda_d)$ are the eigenvalues
- Arrange eigenvectors and eigenvalues such that $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_d$
- Put the first *m* eigenvectors in a matrix

$$\mathbf{A} = \left[\boldsymbol{\phi}_1^t, \boldsymbol{\phi}_2^t, \ldots, \boldsymbol{\phi}_m^t\right]^t$$

- For each $\mathbf{x}_i$, perform $\mathbf{y}_i = \mathbf{A}\,\mathbf{x}_i$

Example

4 classes:

D_1={[1,10],[1,9],[1,7],[1,6],[1,5],[2,8],[2,9],[2,10],[3,9],[3,11],[4,9],[5,9],[6,9],[7,9],[5, 10],[5,11]}^{t}

D_2={[5,3],[6,1],[6,2],[7,1],[7,2],[7,3],[7,5],[8,2],[8,4]}^{t}

D_3={[8,6],[9,3],[9,4],[9,5],[10,2],[10,3],[10,4],[10,5],[10,6],[9,7],[11,3]}^{t}

D_4={[3,3],[3,4],[3,2],[2,2],[2,4],[3,5],[4,3]}^{t}

where $\mathbf{x} \in D \quad \text{and} \quad n = |D|$

- Calculate the sample mean: $\mathbf{m} = \frac{1}{n}\sum_{k=1}^{n}\mathbf{x}_k = \begin{bmatrix} 5.46 \\ 5.46 \end{bmatrix}$

- Calculate the *scatter matrix* **S**:

$$\mathbf{S} = \sum_{k=1}^{n} (\mathbf{x}_k - \mathbf{m})(\mathbf{x}_k - \mathbf{m})^t = \begin{bmatrix} 9.92 & -3.67 \\ -3.67 & 9.25 \end{bmatrix}$$

- The *eigenvectors* and *eigenvalues* of **S** are:

$$\mathbf{e}_1 = \begin{bmatrix} -0.67 & -0.74 \end{bmatrix}^t \qquad \lambda_1 = 5.90$$

$$\mathbf{e}_2 = \begin{bmatrix} -0.74 & 0.67 \end{bmatrix}^t \qquad \lambda_2 = 13.28$$

- Then, the largest eigenvalue is $\lambda_2 = 13.28$,

which corresponds to eigenvector $\mathbf{e}_2$

- Project the data D_1, D_2, D_3 and D_4 in the direction of $\mathbf{e}_2$

$$\mathbf{y}_i = \mathbf{e}_2 \mathbf{x}_i \quad \text{with } i = 1,\ldots,n$$

$$\mathbf{y}_1 = \begin{bmatrix} -0.74 & 0.67 \end{bmatrix} \begin{bmatrix} 1 \\ 10 \end{bmatrix} = 5.96$$

$$\mathbf{y}_2 = \begin{bmatrix} -0.74 & 0.67 \end{bmatrix} \begin{bmatrix} 1 \\ 9 \end{bmatrix} = 5.29$$

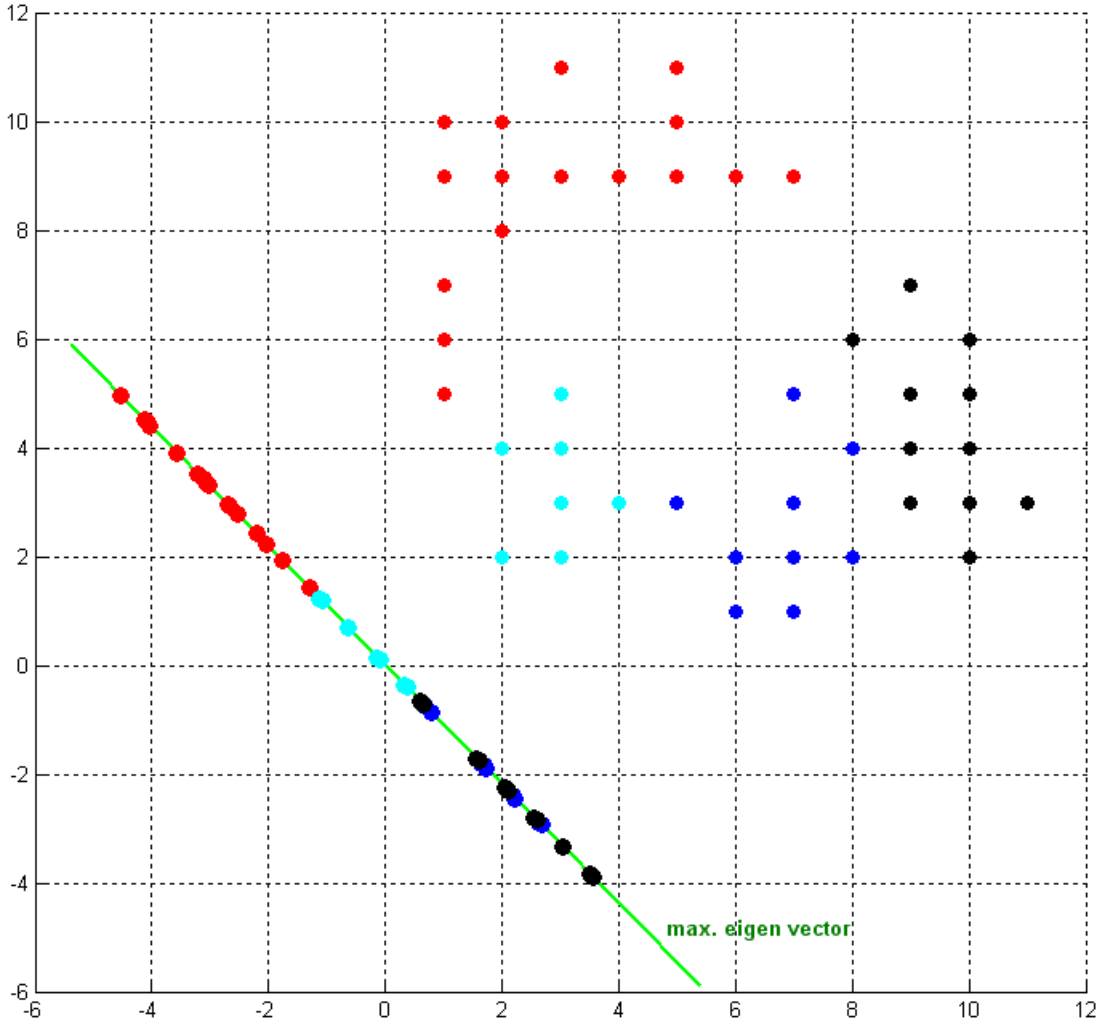
$$\mathbf{y}_3 = \begin{bmatrix} -0.74 & 0.67 \end{bmatrix} \begin{bmatrix} 1 \\ 7 \end{bmatrix} = 3.95$$

$$\vdots$$

$$\mathbf{y}_n = \begin{bmatrix} -0.74 & 0.67 \end{bmatrix} \mathbf{x}_n$$

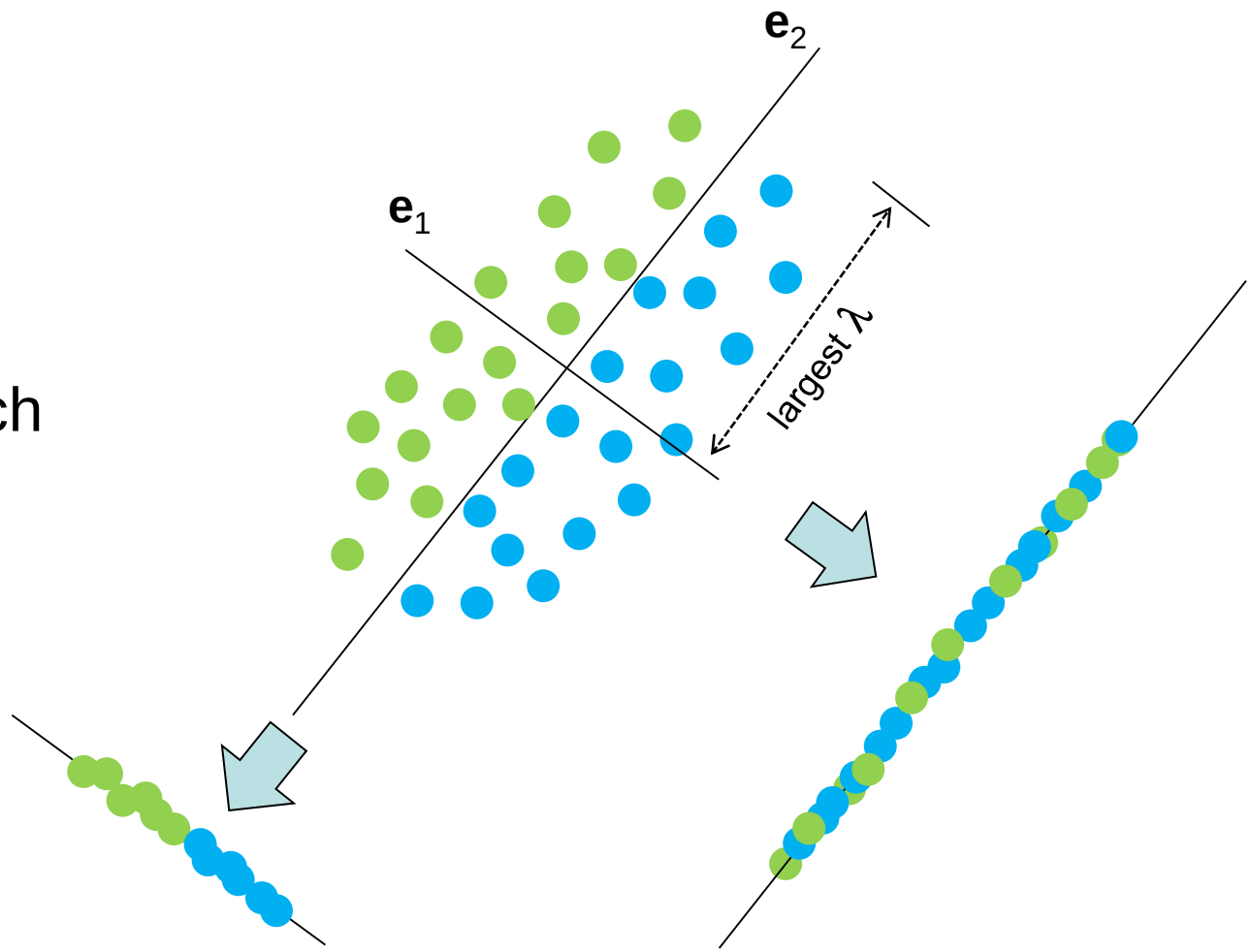
where $\mathbf{y}_i$ is the projection of $\mathbf{x}_i$ on the 1D space

Graphically



Example 2: Unsupervised vs Supervised LDR

- 2 dimensions
- 2 classes
- ≠ means
- Similar covariances
- PCA may not be the best approach for a supervised classification problem!

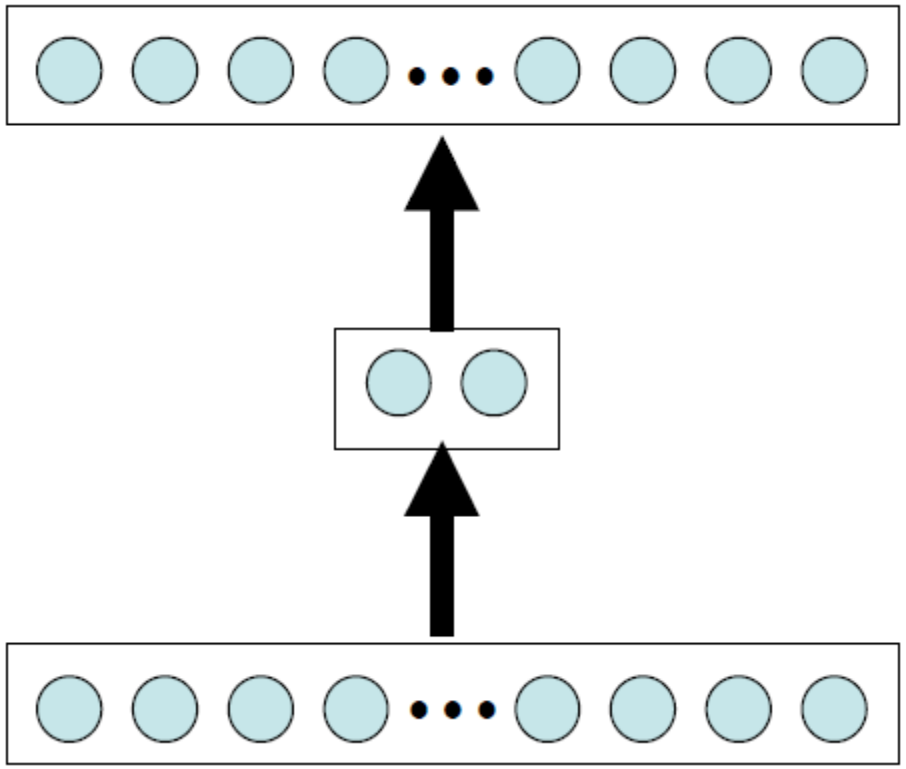


PCA vs Linear Autoencoder

- Autoencoder:
- Each layer implements a linear transformation
- Cost function:
- Minimize the reconstruction error

$$\mathcal{L} = \sum_{\mathbf{x}_i \in D} \|\mathbf{x}_i - \mathbf{A}^t \mathbf{A} \mathbf{x}_i\|_2^2$$

where $\mathbf{A}$ is the transformation matrix obtained by PCA



Singular Value Decomposition (SVD)

- Given $D = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$
- We can write:

$$\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_n]$$

- $\mathbf{X}$ is a matrix of rank $r \leq \min\{n, d\}$
- There exist unitary matrices $\mathbf{U}, \mathbf{V}$ such that

$$\mathbf{X} = \mathbf{U} \begin{bmatrix} \mathbf{\Lambda}^{1/2} & 0 \\ 0 & 0 \end{bmatrix} \mathbf{V}^t$$

where $\mathbf{0}$ is a matrix of zeros and

$\mathbf{\Lambda} = \text{diag}(\lambda_1, \ldots, \lambda_r)$ contains the r nonzero eigenvalues of $\mathbf{X}^t\mathbf{X}$

SVD

- Removing zero sub-matrices, we can write the SVD of $\mathbf{X}$ as:

$$\mathbf{X} = [\mathbf{u}_1, \ldots, \mathbf{u}_r] \begin{bmatrix} \sqrt{\lambda_1} & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & \sqrt{\lambda_r} \end{bmatrix} \begin{bmatrix} \mathbf{v}_1{}^t \\ \vdots \\ \mathbf{v}_r{}^t \end{bmatrix}$$

where $\mathbf{u}_i$ and $\mathbf{v}_i$ are the corresponding eigenvectors of the nonzero eigenvalues of $\mathbf{X}\mathbf{X}^t$ and $\mathbf{X}^t\mathbf{X}$ resp.

SVD

- We can also write:

$$\mathbf{X} = \sum_{i=1}^{r} \sqrt{\lambda_i}\mathbf{u}_i{\mathbf{v}_i}^t$$

- This is an exact representation of $\mathbf{X}$
- But we can do, instead, dimensionality reduction as follows:
  - Sort the eigenvalues: $\lambda_1 \geq \cdots \geq \lambda_r$
  - and pick the largest m eigenvalues $(m \leq r)$
- Then, an approximation of $\mathbf{X}$ is:

$$\mathbf{X} \cong \hat{\mathbf{X}} = \sum_{i=1}^{m} \sqrt{\lambda_i}\mathbf{u}_i{\mathbf{v}_i}^t$$

SVD

- In matrix form, $\widehat{\mathbf{X}}$ is expressed as:

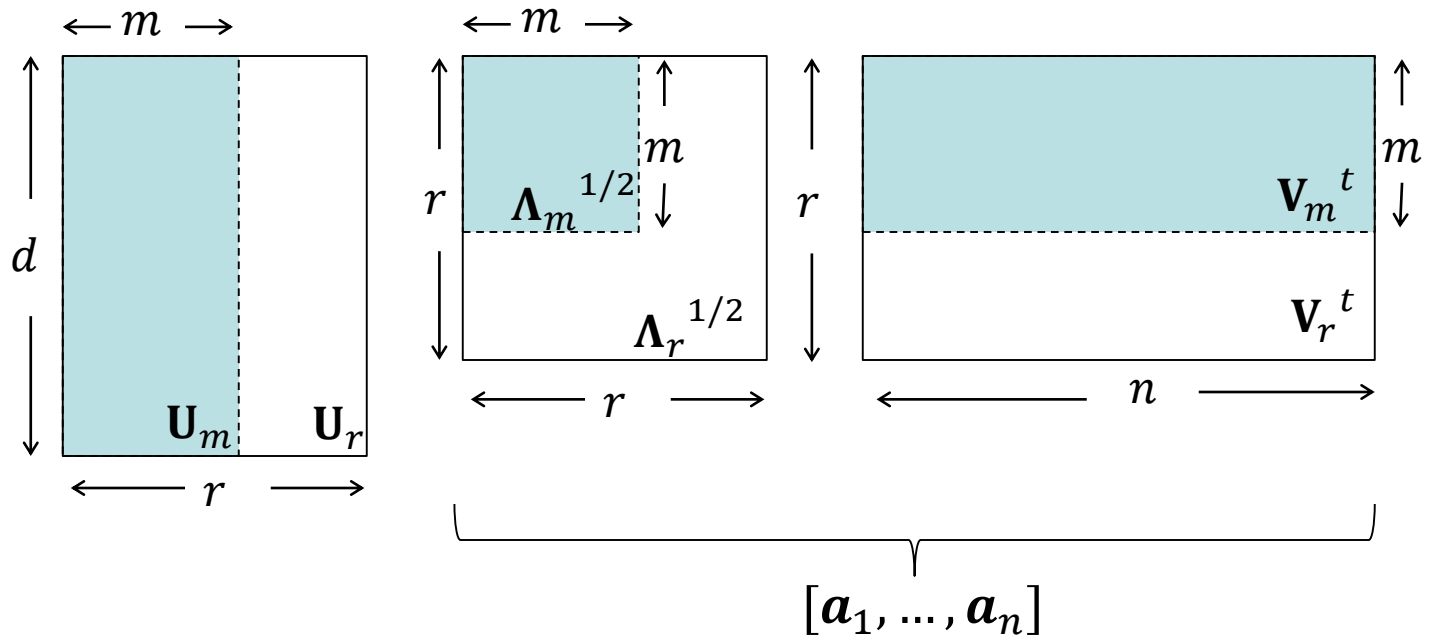
$$\widehat{\mathbf{X}} = \mathbf{U}_m \boldsymbol{\Lambda}_m{}^{1/2} \mathbf{V}_m{}^t = \mathbf{U}_m \, [\boldsymbol{a}_1, \ldots, \boldsymbol{a}_n]$$

  where $\mathbf{U}_m = [\mathbf{u}_1, \ldots, \mathbf{u}_m]$ and $\mathbf{V}_m = [\mathbf{v}_1, \ldots, \mathbf{v}_m]$

- Each $\mathbf{x}_i$ is projected onto the m-dimensional subspace spanned by $\mathbf{U}_m$
- $\boldsymbol{a}_1, \ldots, \boldsymbol{a}_n$ are the corresponding vectors in the new space
- SVD vs PCA:
  - $\widehat{\mathbf{X}}$ is the best rank-m approximation of $\mathbf{X}$ in the Frobenius space
  - whereas the optimality of PCA is in terms of the mean square error

SVD

- SVD graphically:



- PCA vs SVD:
  - PCA is statistics-oriented
    - Uses the parameters of the distributions
  - SVD is data-oriented
    - Decomposes the data matrix itself

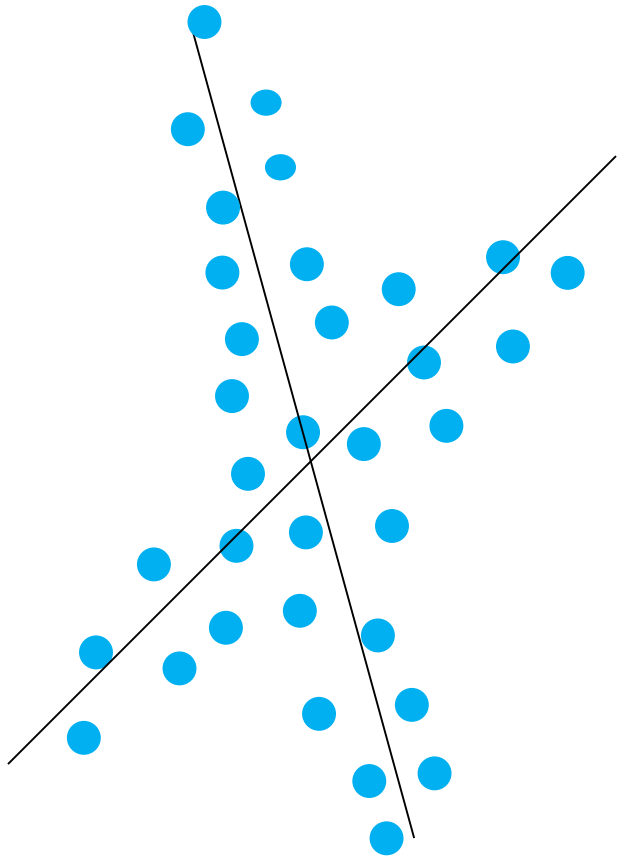
Independent Component Analysis (ICA)

- PCA focuses more on correlation of random variables (second order moments)
- Independent component analysis considers higher order moments

Given $\mathbf{Z} = \{\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_n\}$

For each $\mathbf{z}_t$, perform $\hat{\mathbf{x}}_t = \mathbf{W}^{-1}\mathbf{z}_t$

- Find $\mathbf{W}$ whose vectors are mutually independent
- Methods based on:
  - MLE + gradient
  - FastICA: based on Hessian matrix
    - Approximate Newton method
- PCA finds orthogonal vectors for projection, whereas
- ICA finds linearly independent vectors (not necessarily orthogonal)



The Cocktail Party Problem

- You’re in a cocktail party
- Many people talking at the same time
- We can we understand anything at all?
- Problem: Blind signal separation or blind source separation
- Applicable to EEG, financial or other types of data

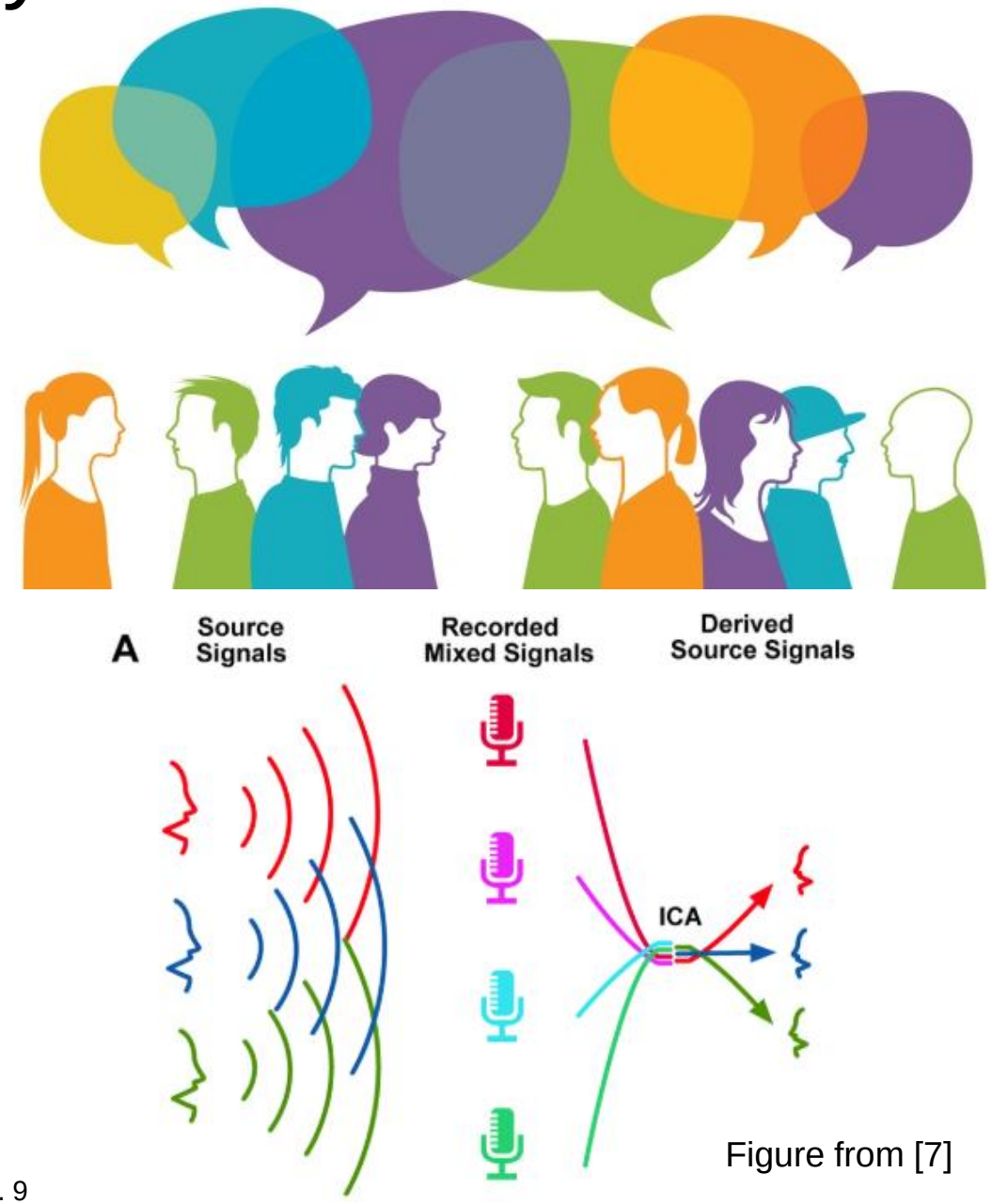
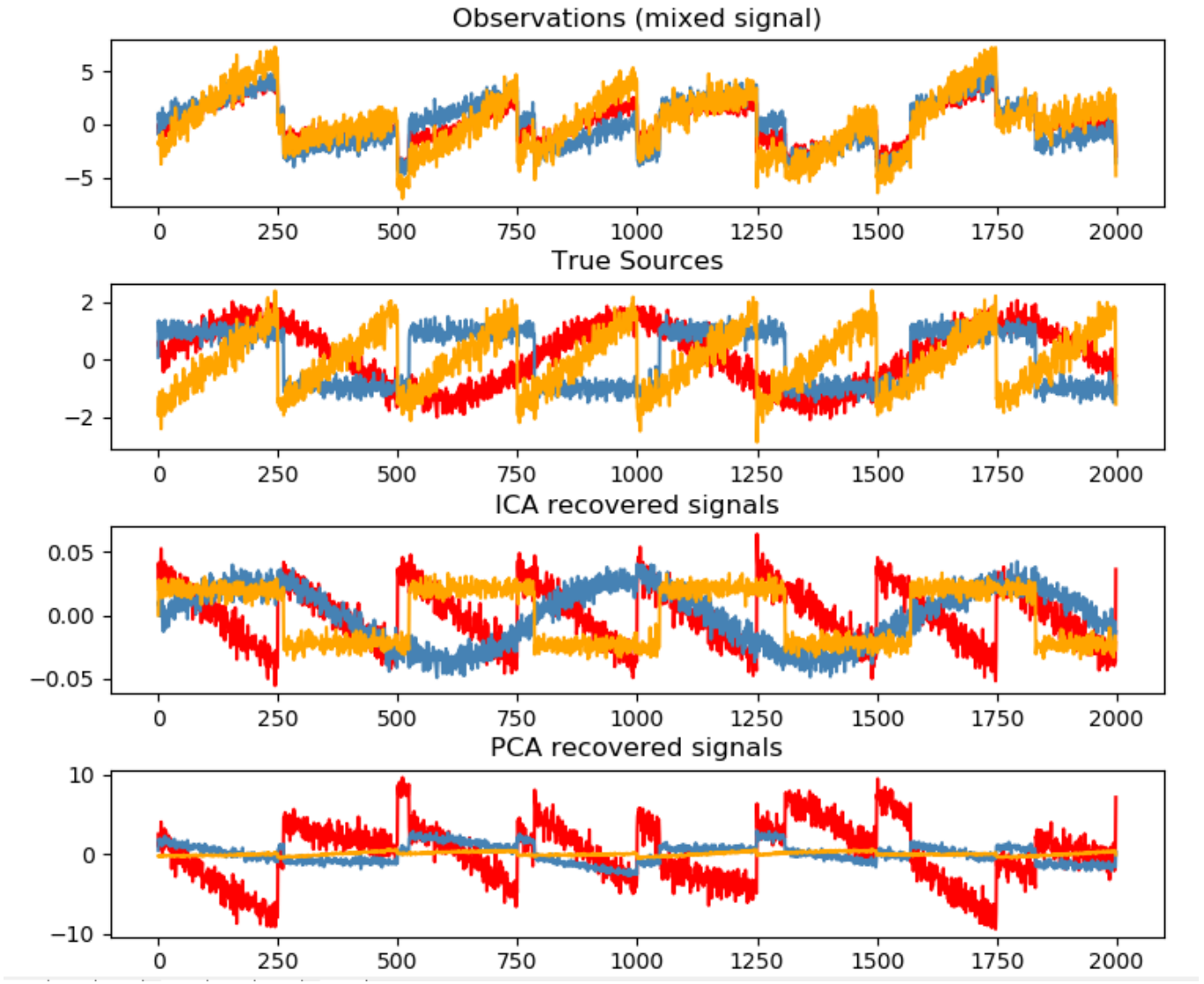


Figure from [7]

Musical Instruments

- Problem similar to the cocktail party
- 3 musical instruments played at the same time
- $d = l = 3$
- Data is sampled over time
- May not be the case in other applications



ICA – Problem Formulation

More formally:

- $\mathbf{x}_t \in \mathbb{R}^d$ are the observed signals
- t = time
- $\mathbf{z}_t \in \mathbb{R}^l$ is the source of signals
- Assume $d = l$
- i.e., we know how many people are talking at the same time
- Assume that

$$\mathbf{x}_t = \mathbf{W}\mathbf{z}_t + \boldsymbol{\epsilon}_t$$

where $\mathbf{W}$ is a $d \times l$ matrix

and $\boldsymbol{\epsilon}_t \sim N(\mathbf{0}, \boldsymbol{\Psi})$;

$\boldsymbol{\Psi}$ is aka "white noise"

In general, we assume $\boldsymbol{\Psi}$ is zero

- Given $\mathbf{Z}$, the aim is to find $\mathbf{W}$ whose rows are not necessarily orthogonal

where $\mathbf{z}_t$ follows a distribution:

$$p(\mathbf{z}_t) = \prod_{j=1}^{l} p_j(z_{tj})$$

- This distribution is like "mixture" of distributions.
- In case of PCA, we would be restricted to Gaussian distributions

Algorithms:

- MLE
- FastICA (uses 2nd order derivatives)
- Expectation maximization

Example – PCA vs ICA

- 2D data $\mathbf{Z}$ generated randomly
- 2 t-student data generated (true independent sources)
- Data mixed with matrix

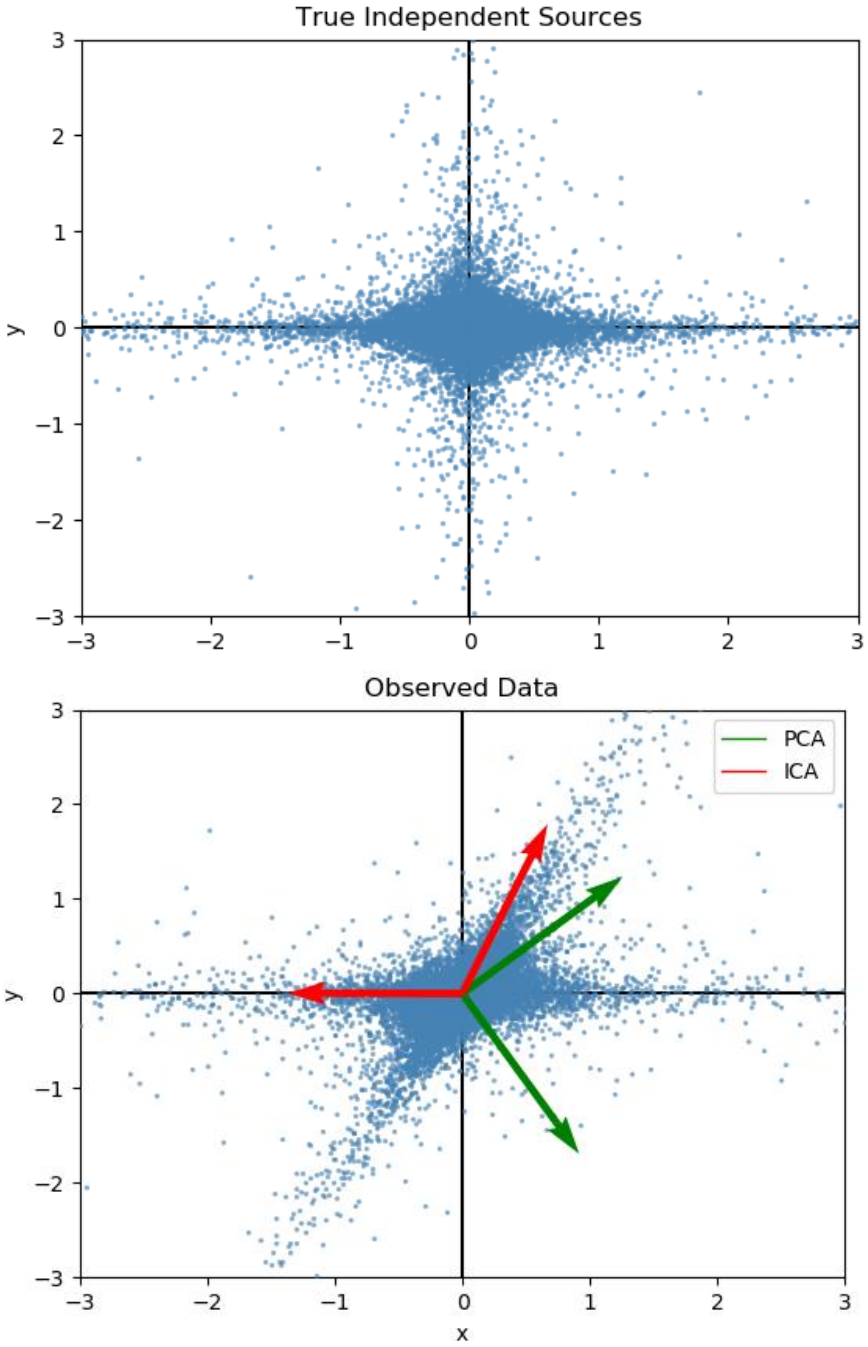
$$\mathbf{W} = \begin{bmatrix} 1 & 1 \\ 0 & 2 \end{bmatrix}$$

- $\mathbf{W}$ is not orthogonal
- PCA and FastICA applied to Observed Data $\mathbf{X} = \mathbf{WZ}$
- Components of PCA are orthogonal but not capture the 2 signals
- Components of ICA are not orthogonal but represent the 2 signals better
- Apply inverse transformation

$$\hat{\mathbf{Z}} = \mathbf{W}^{-1}\mathbf{Z}$$

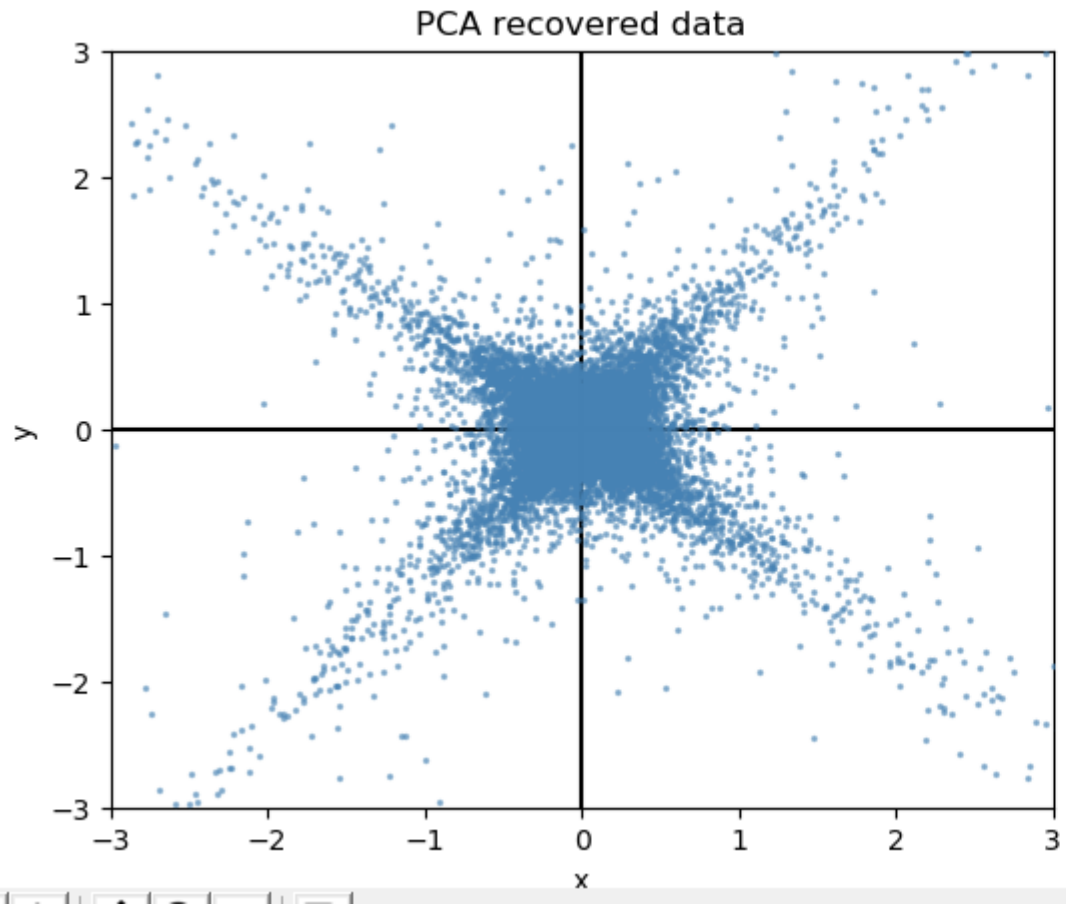
$$\text{where } \mathbf{W} = \begin{bmatrix} 1 & -0.5 \\ 0 & 0.5 \end{bmatrix}$$

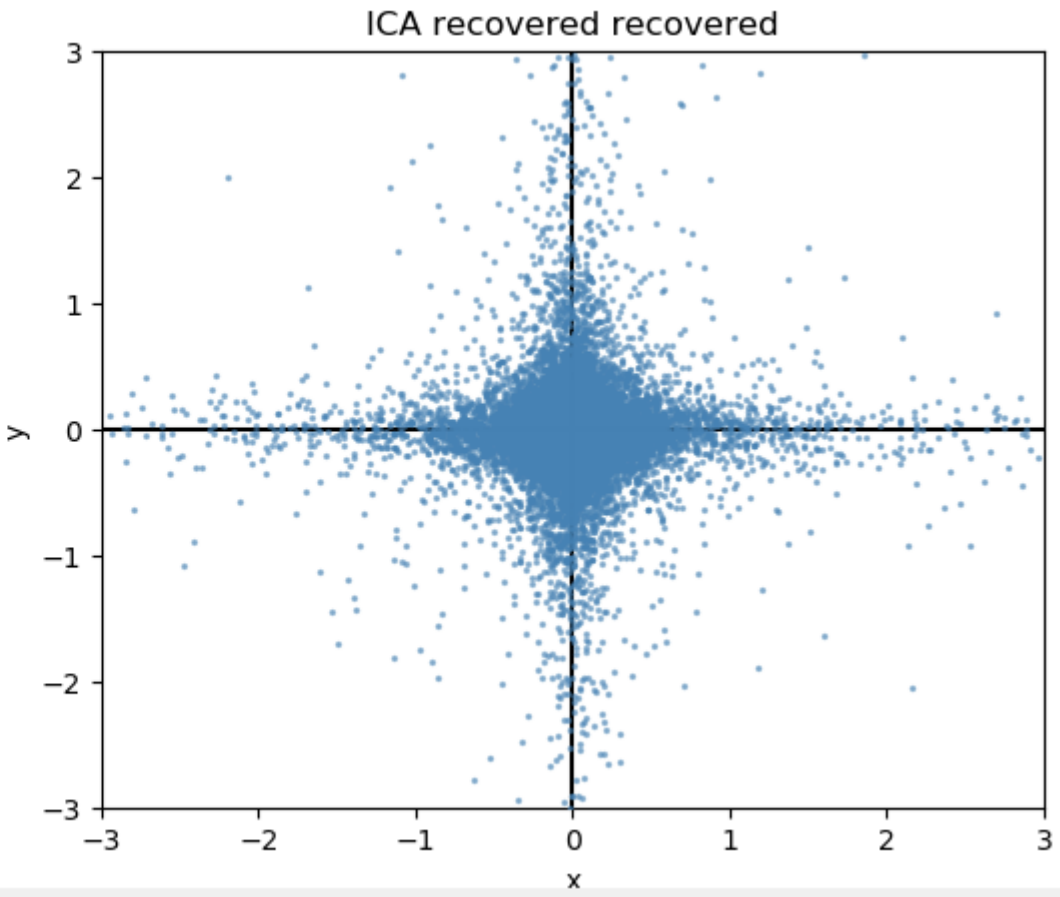
- The actual matrix obtained from FastICA $\widehat{\mathbf{W}}$ is slightly different from $\mathbf{W}$



Recovered data from PCA and ICA

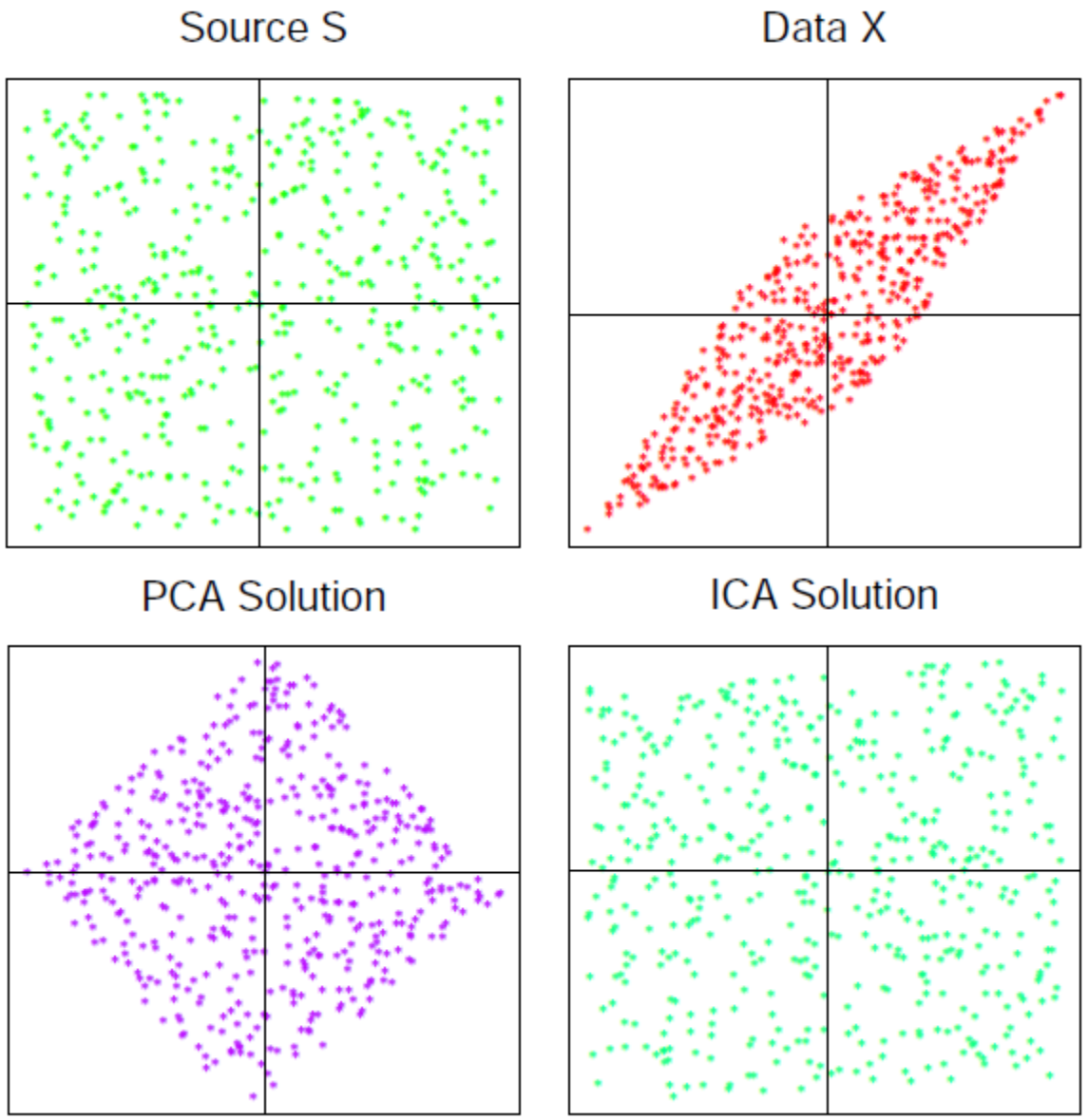
$$\hat{\mathbf{X}} = \mathbf{W}^{-1}\mathbf{Y}$$





Example - ICA

- Source, S: 500 random points uniformly distributed on the plane
- Data, X: points after applying an arbitrary non-orthogonal linear transformation
- PCA captures the main diagonals but not the shape
- ICA recovers the original data with high accuracy



Nonnegative Matrix Factorization (NMF)

- Given $D = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$
- We can write:

$$\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_n]$$

- $\mathbf{X}$ is a matrix of rank $r \leq \min\{n, d\}$
- Recall SVD, which aims to find an approximation of $\mathbf{X}$:

$$\mathbf{X} \cong \widehat{\mathbf{X}} = \sum_{i=1}^{m} \sqrt{\lambda_i}\mathbf{u}_i{\mathbf{v}_i}^t$$

where $\widehat{\mathbf{X}}$ is the best rank-m approximation of $\mathbf{X}$

- Problems with PCA/SVD
  - Original matrix $\mathbf{X}$ is typically dense
  - Interpretation of basis vectors is difficult because it contains negative values
  - It is very difficult to obtain a good value of *m*
  - Can't deal with sparse matrices

$$\begin{bmatrix} 0 & 3.8 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 5.2 & 0 & 0 \\ 0 & 0 & -3.2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 10 & 0 \end{bmatrix}$$

NMF

- Given a sparse *n* x *d* matrix:

$$\mathbf{X} = [\mathbf{x}_1^t, \ldots, \mathbf{x}_n^t]^t$$

- Find two smaller matrices $\mathbf{P}$ and $\mathbf{Q}$ of dimensions *n* x *m* and *d* x *m* such that

$$\mathbf{X} \approx \widehat{\mathbf{X}} = \mathbf{P}\mathbf{Q}^t$$

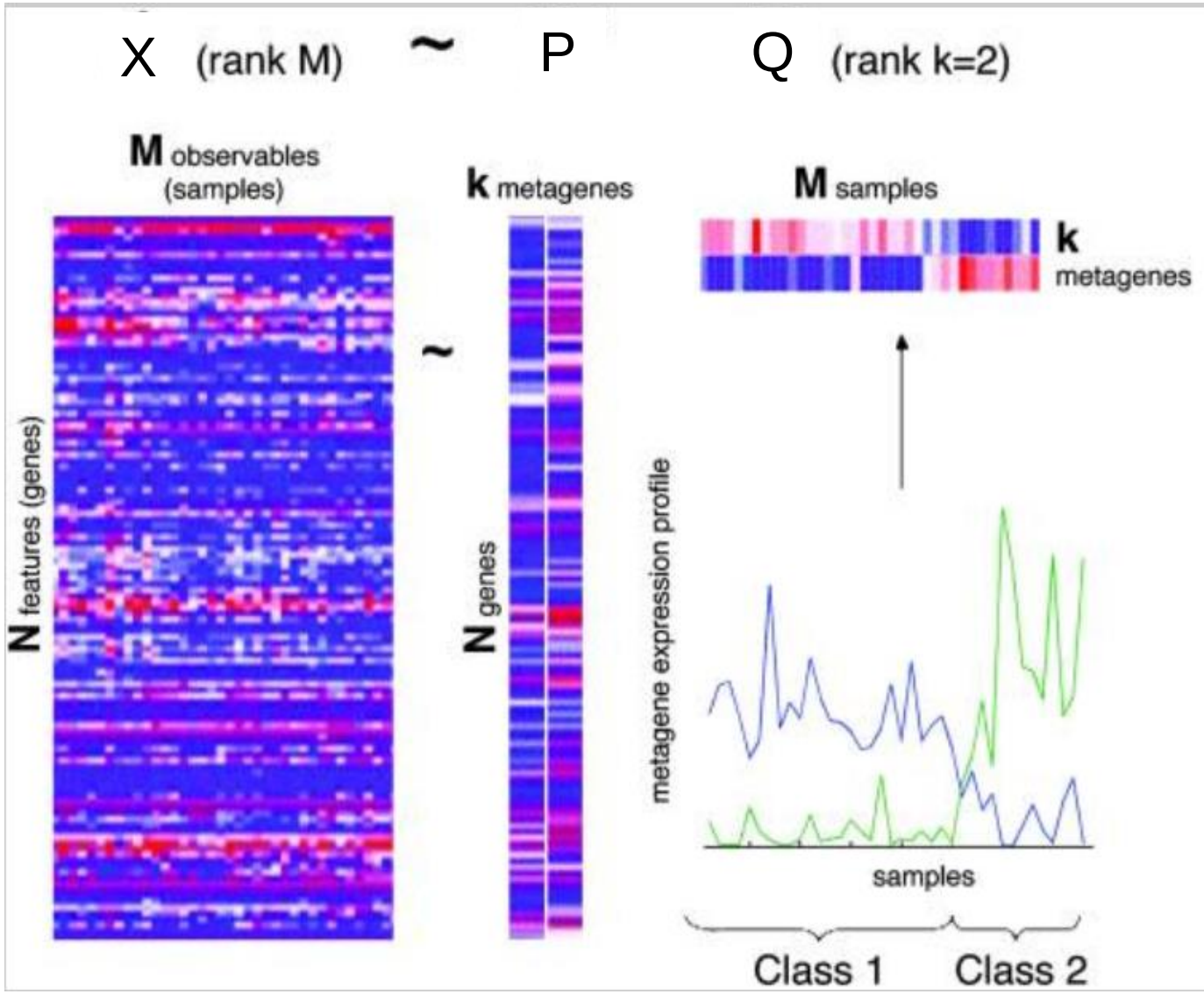
- i.e., $\widehat{\mathbf{X}}$ is an approximation of $\mathbf{X}$
- where $\widehat{\mathbf{X}}$, $\mathbf{P}$ and $\mathbf{Q}$ are nonnegative, i.e., they contain only positive values
- Ideally, $m \ll d$
- Example:

$$\underset{\mathbf{X}}{\begin{bmatrix} 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \end{bmatrix}} \approx \underset{\mathbf{P}}{\begin{bmatrix} 1.58 & 0 \\ 1.57 & 0.01 \\ 0 & 1.42 \end{bmatrix}} \underset{\mathbf{Q}}{\begin{bmatrix} 0.32 & 0.32 & 0.63 & 0.63 & 0 \\ 0 & 0 & 0.7 & 0 & 0.7 \end{bmatrix}}$$

- The columns of $\mathbf{P}$ are the underlying basis vectors
- each of the d columns of $\widehat{\mathbf{X}}$ can be obtained from the columns of $\mathbf{P}$
- The columns of $\mathbf{Q}$ are the weights associated with each basis vector
- The columns of $\mathbf{P}$ are the new features, and the rows are the samples

NMF - Example

- Microarray data from [5]
- $\mathbf{X}$ contains *n* genes and *d* observables (M in the example)
- Aim is to find *m* metagenes (*k* in example)
- A smaller number of genes meaningful with biological data
  - Just 2 metagenes
  - Can divide data into 2 classes



NMF - Properties

- Basis vectors of $\mathbf{P}$ are not orthogonal
- can overlap samples/features
- Can restrict $\mathbf{P}$, $\mathbf{Q}$ to be sparse
- NMF is algorithm dependent
  - $\mathbf{P}$ and $\mathbf{Q}$ are not unique
- Solved as an optimization problem
- Objective function based on mean squared error

$$\min \left\| \widehat{\mathbf{X}} - \mathbf{PQ} \right\|_F^2$$

- such that $\mathbf{P}, \mathbf{Q} \geq 0$

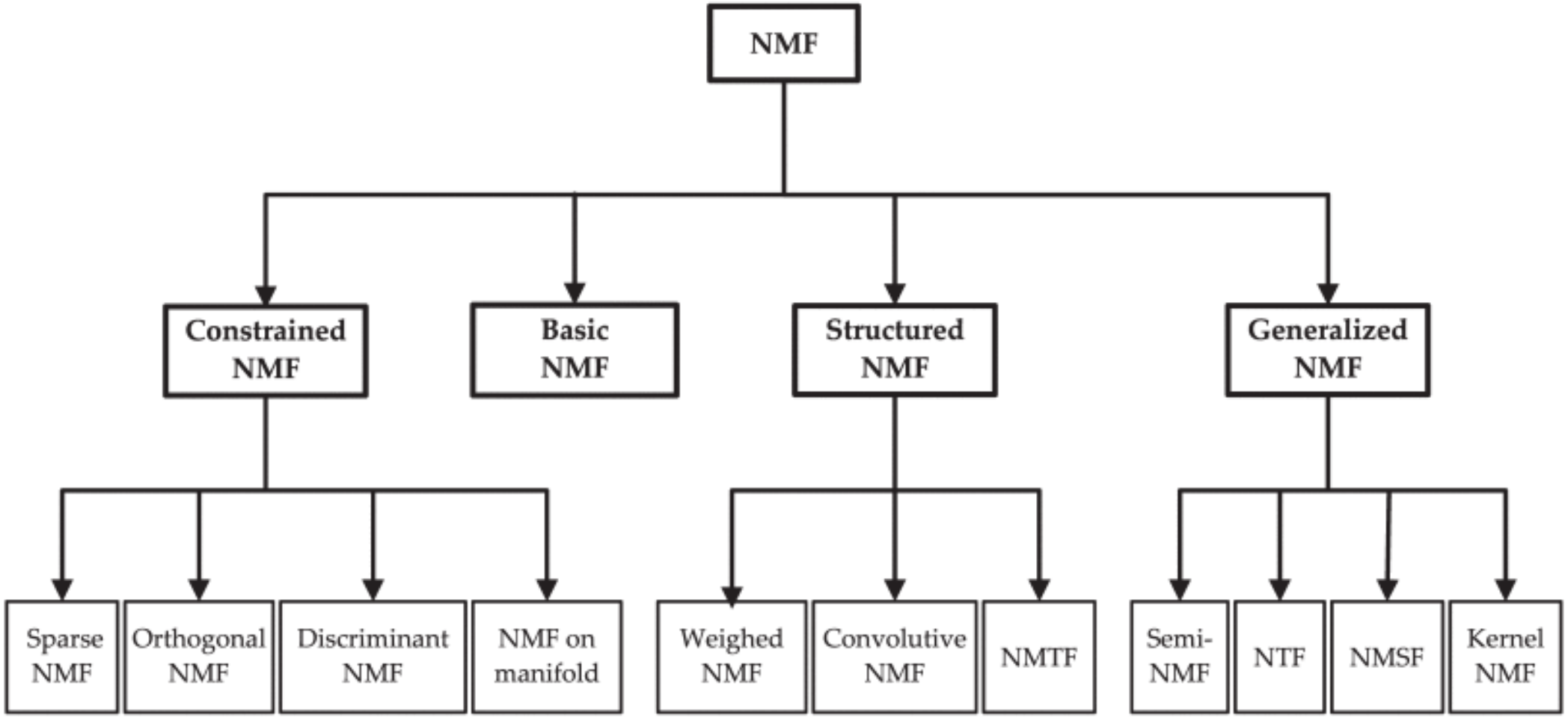
- Nonlinear optimization problem
- Convex in $\mathbf{P}$ or $\mathbf{Q}$ but not on both
- Difficult to obtain a global minimum
- Large number of unknowns, i.e., the values of $\mathbf{P}$ and $\mathbf{Q}$
- Optimization problem not easily solved
- Many algorithms exist

NMF - Algorithms

Four main categories:

- Basic NMF (BNMF), which only imposes the nonnegativity constraint.
- Constrained NMF (CNMF), which imposes some additional constraints as regularization.
- Structured NMF (SNMF), which modifies the standard factorization formulations.
- Generalized NMF (GNMF), which breaks through the conventional data types or factorization modes in a broad sense.

NMF – Algorithms – Schematic view



NMF - Algorithms

Constrained NMF – four subclasses:

- Sparse NMF (SPNMF), which imposes the sparseness constraint.
- Orthogonal NMF (ONMF), which imposes the orthogonality constraint.
- Discriminant NMF (DNMF), which involves the information for classification and discrimination.
- NMF on manifold (MNMF), which preserves the local topological properties.

NMF - Algorithms

Structured NMF – three subclasses:

- Weighed NMF (WNMF), which attaches different weights to different elements regarding their relative importance.
- Convolutive NMF (CVNMF), which considers the time-frequency domain factorization.
- Nonnegative Matrix Trifactorization (NMTF), which decomposes the data matrix into three factor matrices.

NMF - Algorithms

Generalized NMF – four subclasses:

- Semi-NMF, which relaxes the nonnegativity constraint only on the specific factor matrix
- Nonnegative Tensor Factorization (NTF), which generalizes the matrix-form data to higher dimensional tensors.
- Nonnegative Matrix-Set Factorization (NMSF), which extends the data sets from matrices to matrix-sets.
- Kernel NMF (KNMF), which is the nonlinear model of NMF.

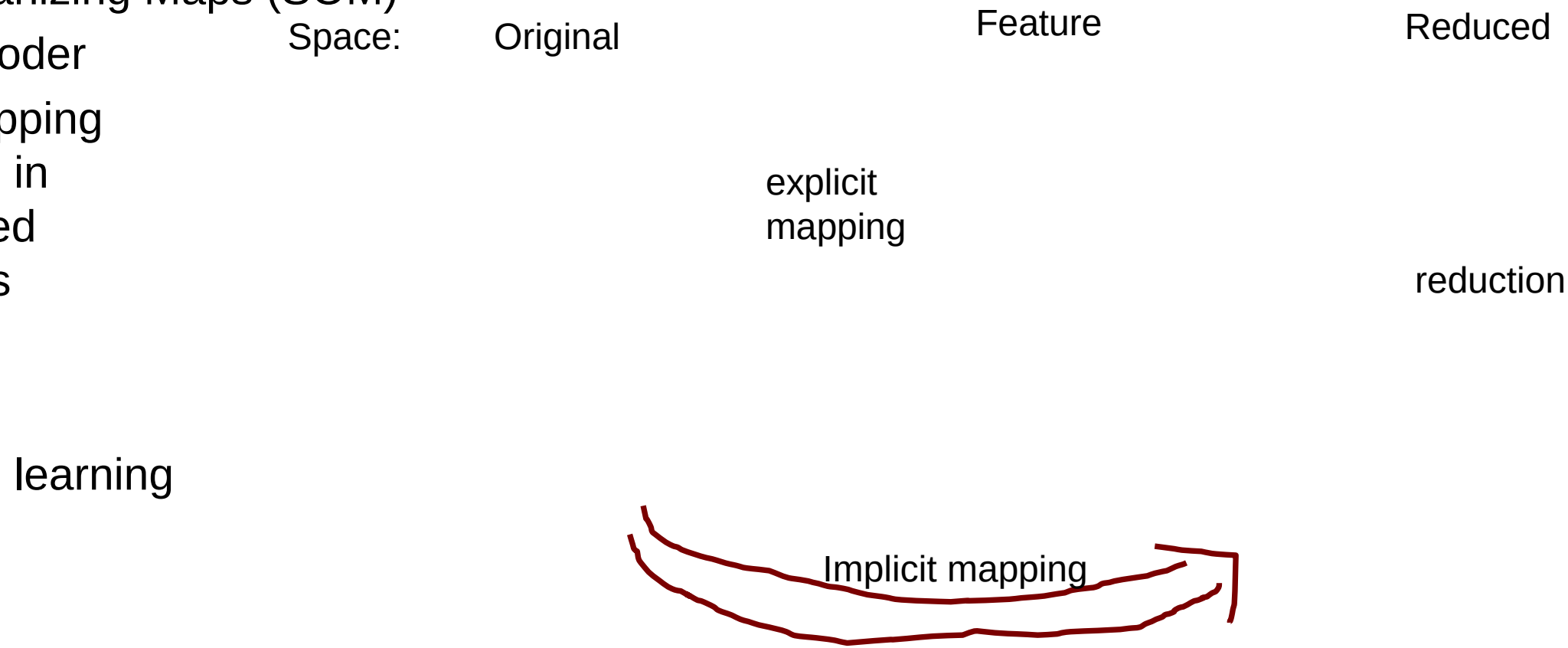
Nonlinear Dimensionality Reduction (NDR)

Methods:

- Kernel-based:
  - PCA, NMF, FDA, HDA, etc… (even SVM)
- Multi-dimensional Scaling (MDS)
- Self-organizing Maps (SOM)
- Autoencoder

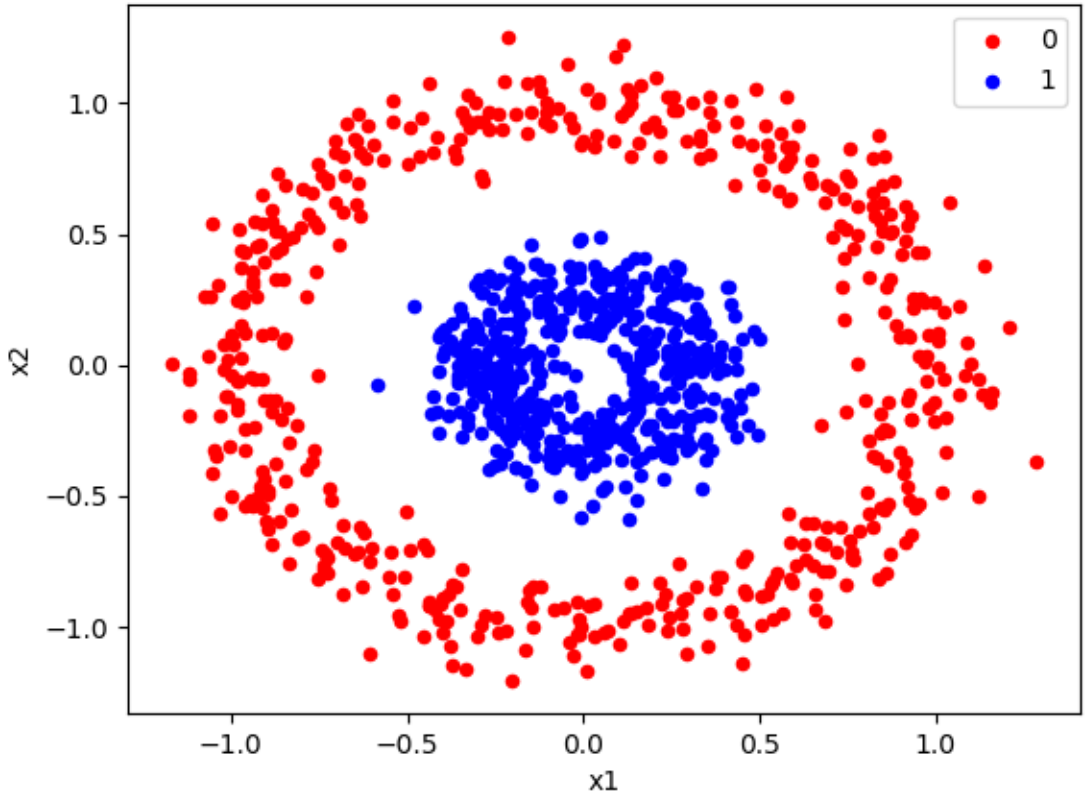
Explicit mapping not needed in kernel-based approaches

- KPCA
- KFDA
- Manifold learning

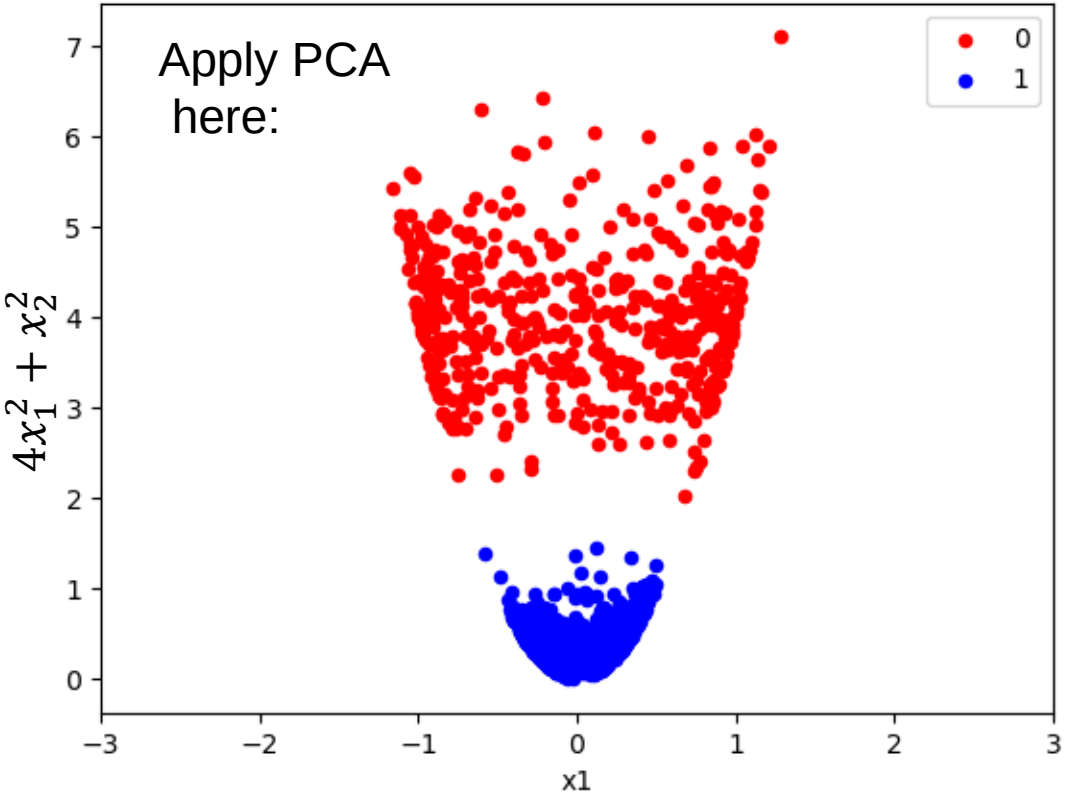


Kernel PCA – Motivation/Example

Original 2D space:



New 2D space:



Kernel PCA

- No known classes (unsupervised)
- Project vectors onto feature space:

$$\mathbf{x}_i \rightarrow \mathbf{\Phi}(\mathbf{x}_i)$$

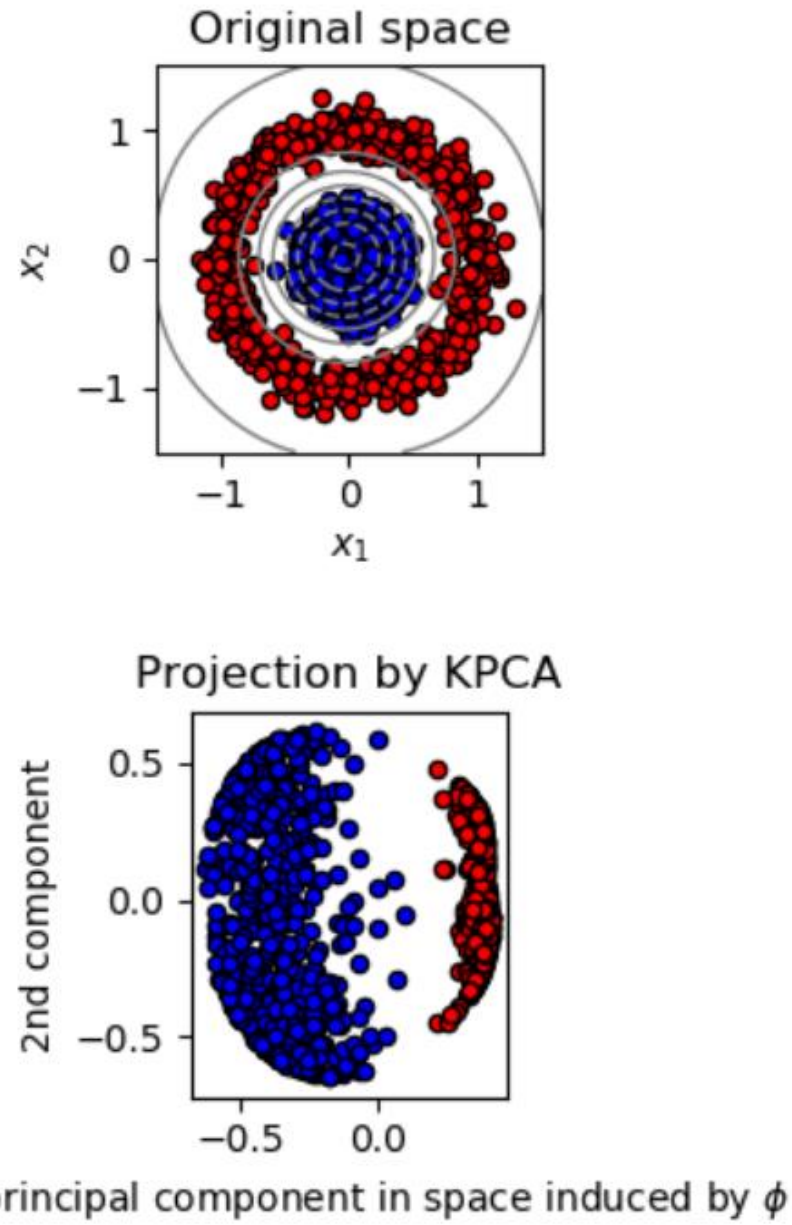
- Not needed, since the kernel trick allows us to use $k(\mathbf{x}, \mathbf{y})$ to replace $(\mathbf{\Phi}(\mathbf{x}) \cdot \mathbf{\Phi}(\mathbf{y}))$
- An eigenvector V of covariance in target space

$$\mathbf{V} = \sum_{i=1}^{n} \alpha_i \tilde{\mathbf{\Phi}}(\mathbf{x}_i)$$

- where

$$\tilde{\mathbf{\Phi}}(\mathbf{x}_i) = \mathbf{\Phi}(\mathbf{x}_i) - \frac{1}{n} \sum_{r=1}^{n} \mathbf{\Phi}(\mathbf{x}_r)$$

- α_i are the components of $\boldsymbol{\alpha}$



Kernel PCA

$\boldsymbol{\alpha}$ is an eigenvector of

$$\tilde{K}_{ij} = \left(\tilde{\boldsymbol{\Phi}}(\mathbf{x}_i) \cdot \tilde{\boldsymbol{\Phi}}(\mathbf{x}_j)\right)$$

choose length of $\boldsymbol{\alpha}$ such that:

$$||\mathbf{V}|| = 1 \Leftrightarrow ||\boldsymbol{\alpha}||^2 = 1/\lambda$$

for this, we need:

$$\tilde{K}_{ij} = K_{ij} - \frac{1}{n}\sum_{r=1}^{n} K_{ir} - \frac{1}{n}\sum_{r=1}^{n} K_{rj} + \frac{1}{n^2}\sum_{r,s=1}^{n} K_{rs}$$

where:

$$K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$$

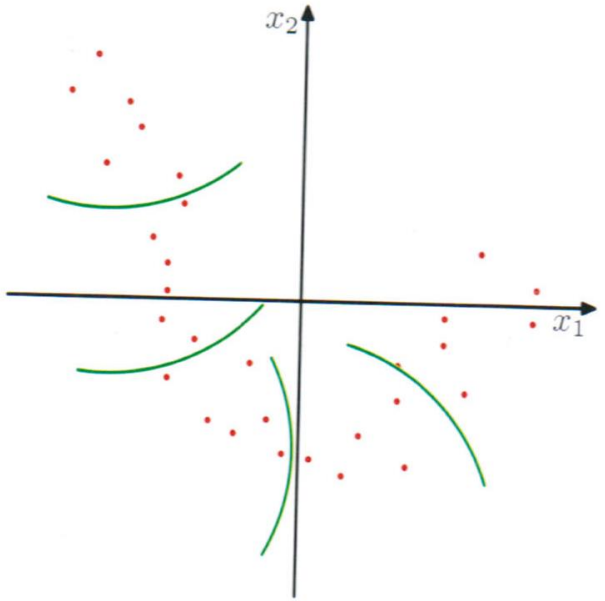
K is the Gram matrix and contains $k(\mathbf{x}_i, \mathbf{x}_j)$ for all pairs of vectors $\mathbf{x}_i$ and $\mathbf{x}_j$

Implementation of Kernel PCA

- Compute the Gram matrix K
- Compute the *m* dominant eigenvalues and eigenvectors of K
  - $\boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2, \ldots, \boldsymbol{\alpha}_m$
- Compute the *m* projections of each dimension of $\mathbf{x}$, y_k, onto each of the dominant eigenvectors
  - $y_k = \langle \mathbf{V}_k, \boldsymbol{\phi}(\mathbf{x}) \rangle = \sum_{i=1}^{d} \alpha_i^k \mathbf{K}(\mathbf{x}_i, \mathbf{x}), \ k = 1, \ldots, m$
- $\mathbf{y} = [y_1, \ldots, y_m]^t$ is the new vector

Kernel PCA: Example 1

original space



feature space

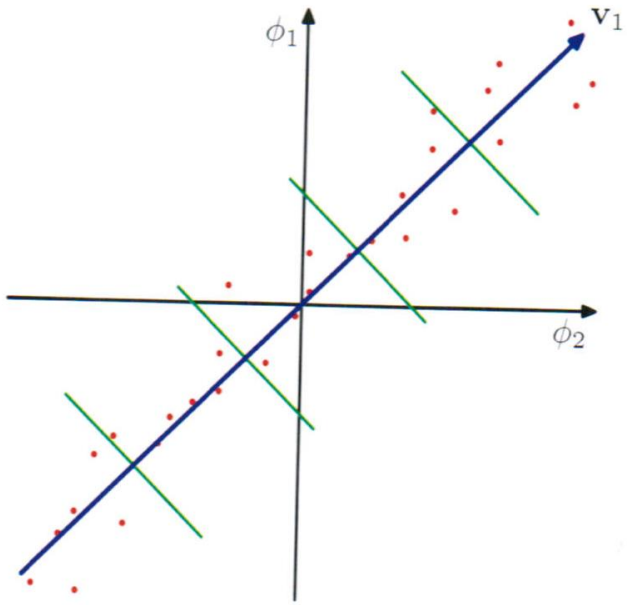
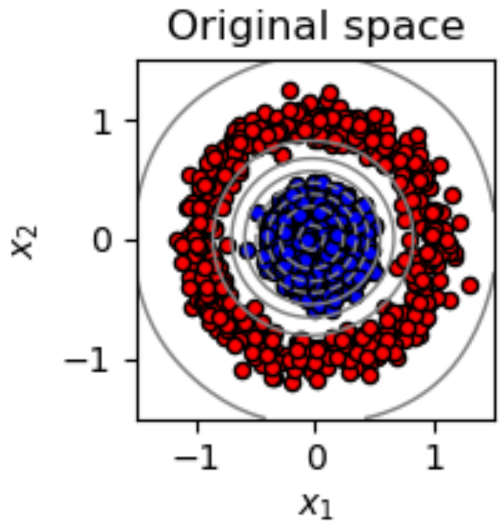
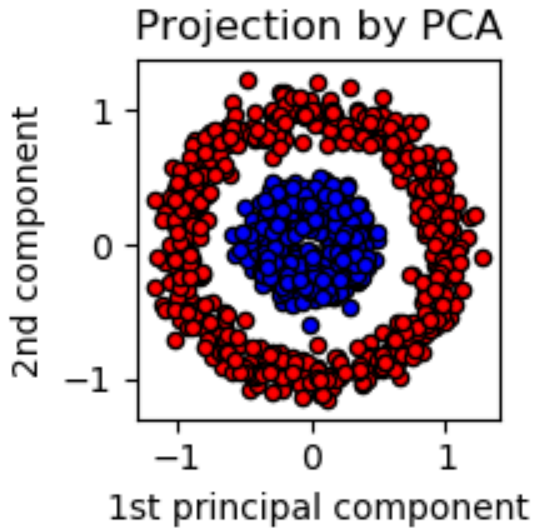


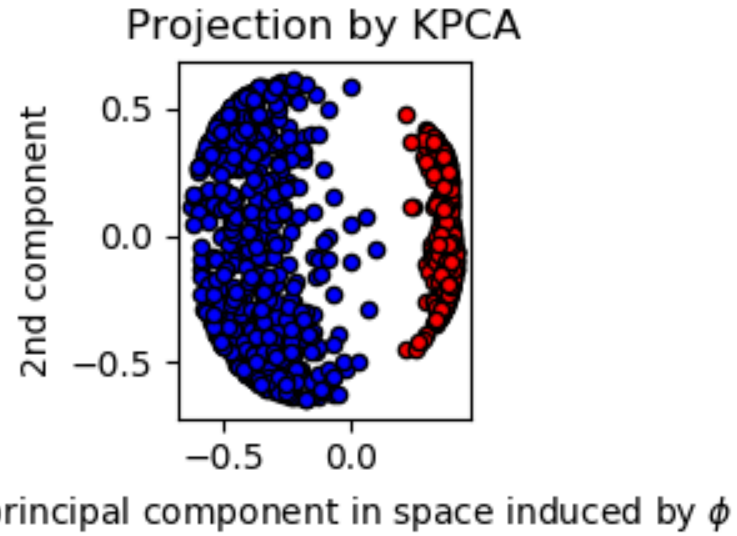
Figure from (Bishop, 2006)

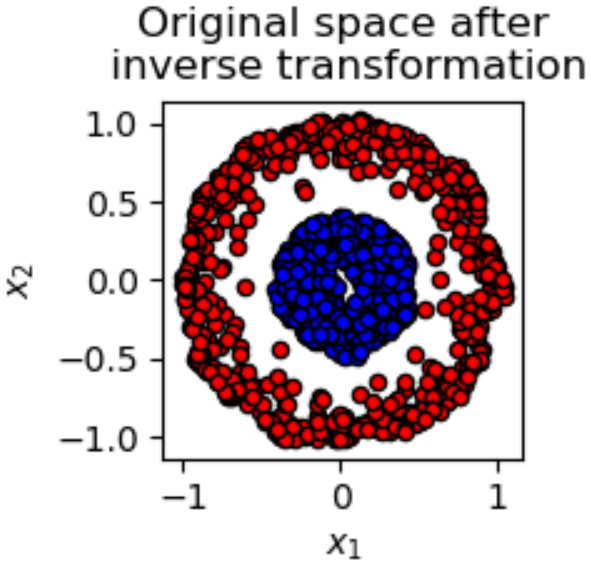
KPCA – Example 2

- 2D data points
- Cluster in cluster
- or “double doughnut”
- RBF kernel:
- $k(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{\sigma^2}\right)$
- Gamma $= \sigma = 4$



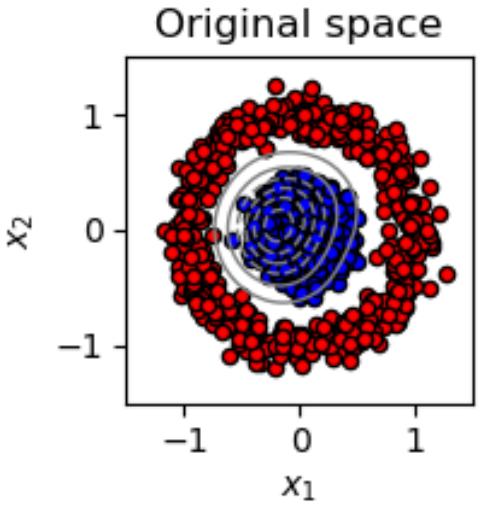


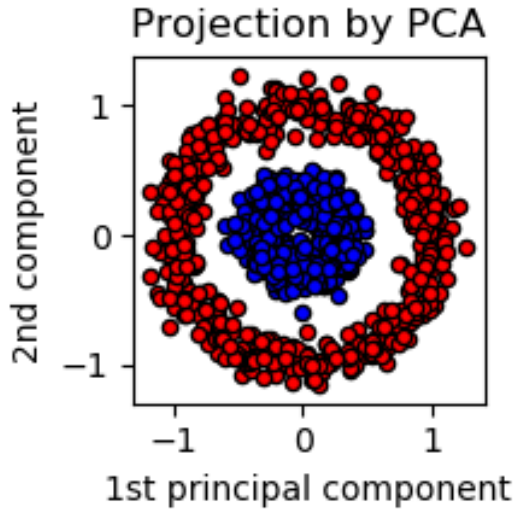


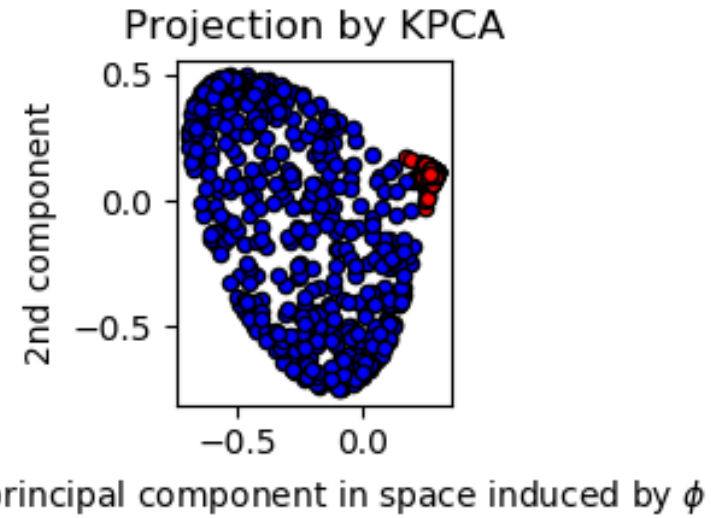


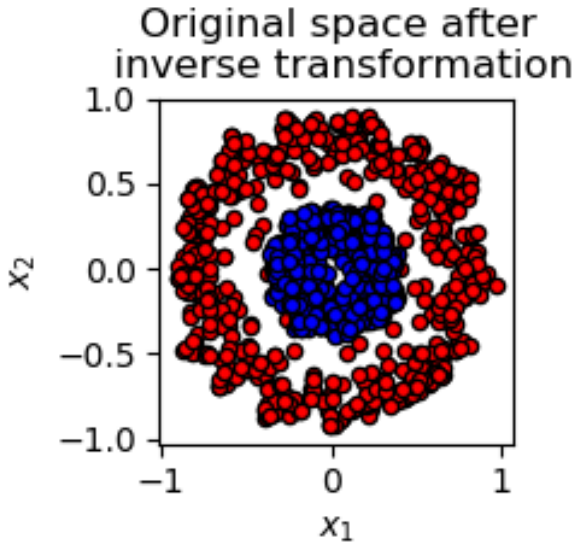
KPCA – Example 2

- 2D data points
- Cluster in cluster
- or “double doughnut”
- RBF kernel
- $k(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{\sigma^2}\right)$
- Gamma $= \sigma = 10$



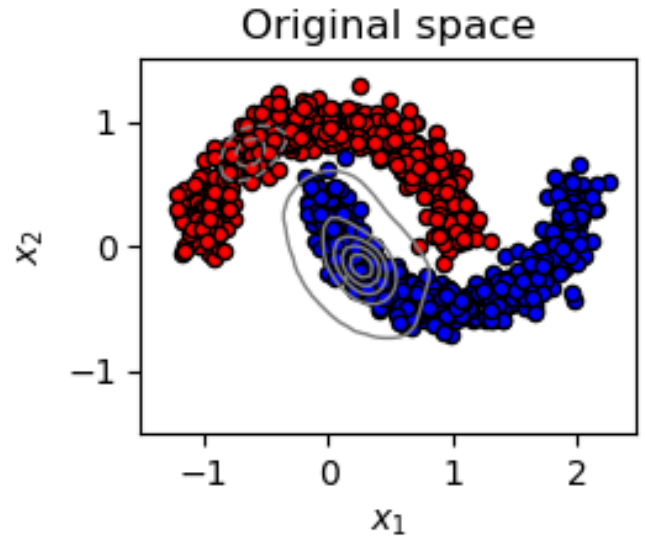


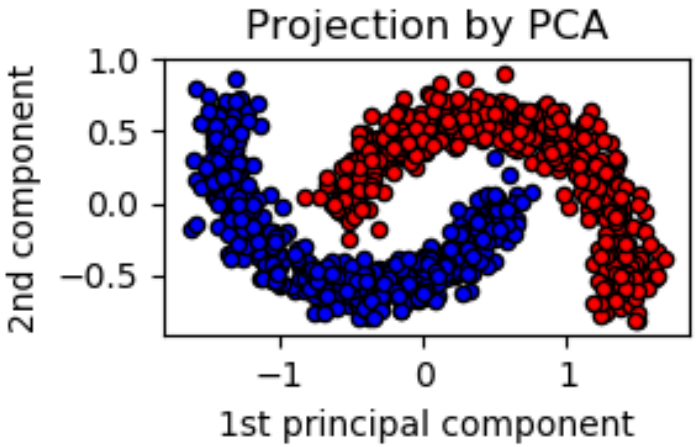


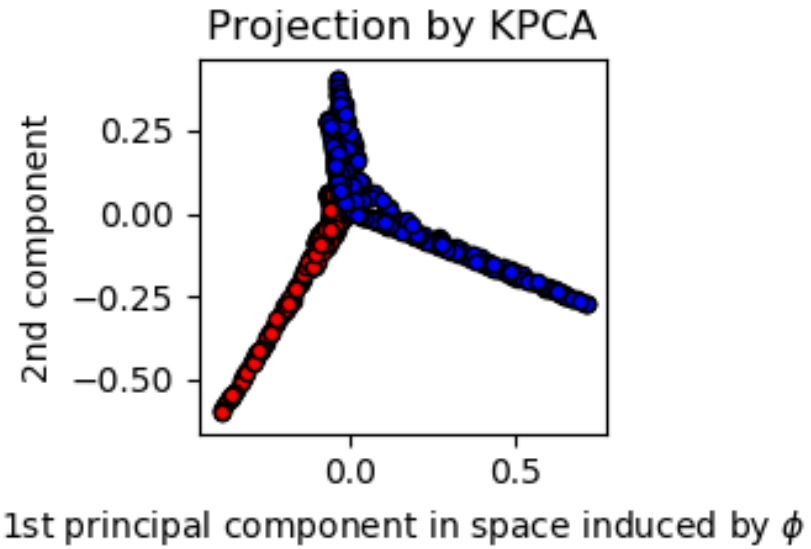


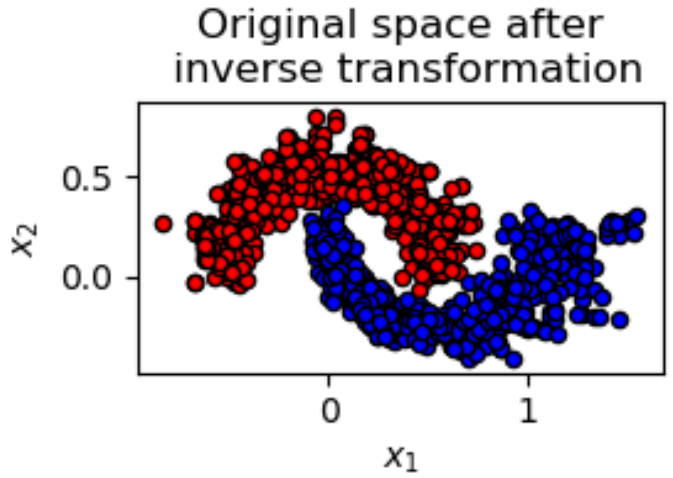
KPCA – Example 2

- 2D data points
- Moons dataset
- RBF kernel
- $k(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{\sigma^2}\right)$
- Gamma $= \sigma = 40$





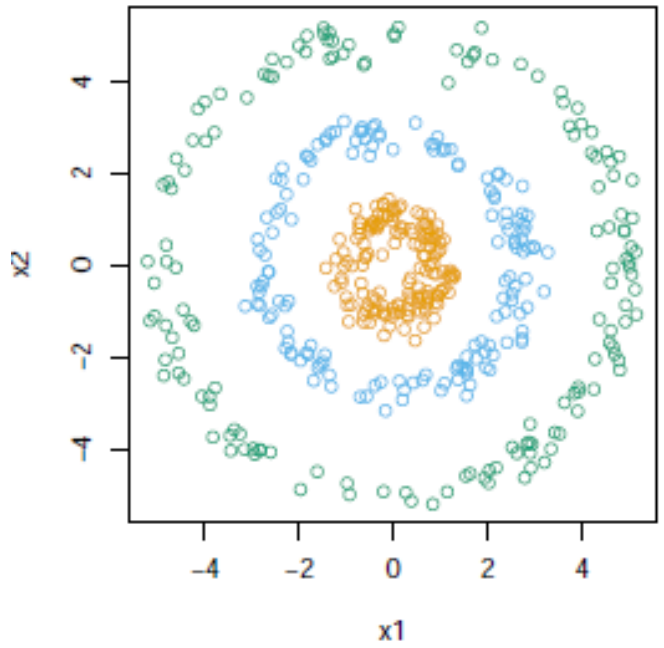




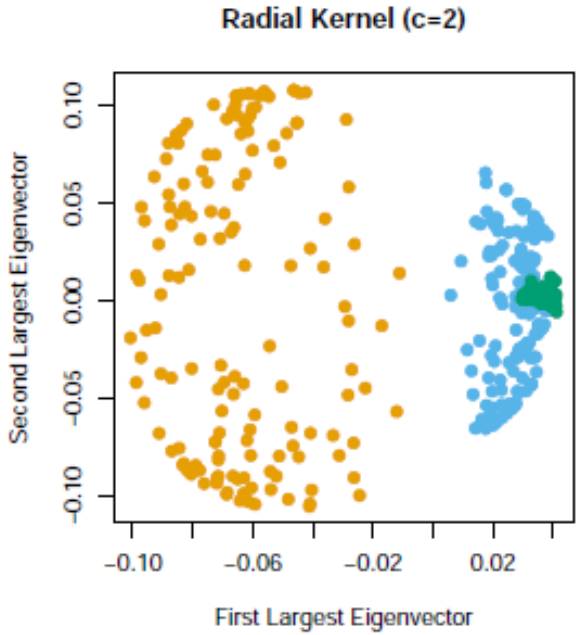
KPCA – Example 3

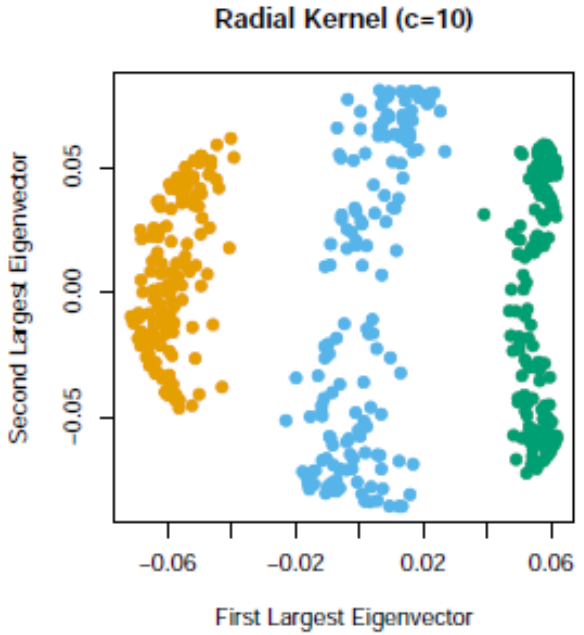
- 2D data points
- 3 circles
- RBF kernel:

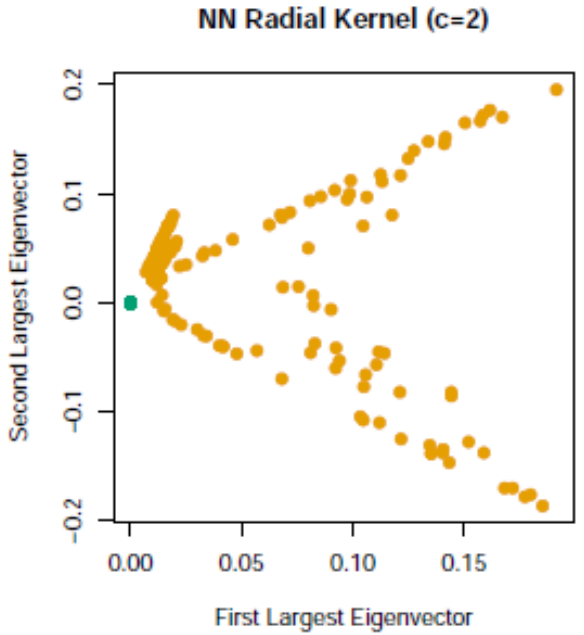
$$k(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{c}\right)$$

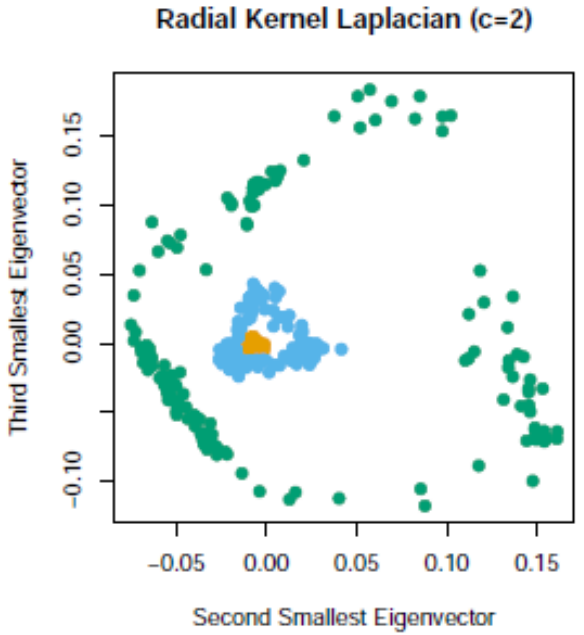


Example from [12]



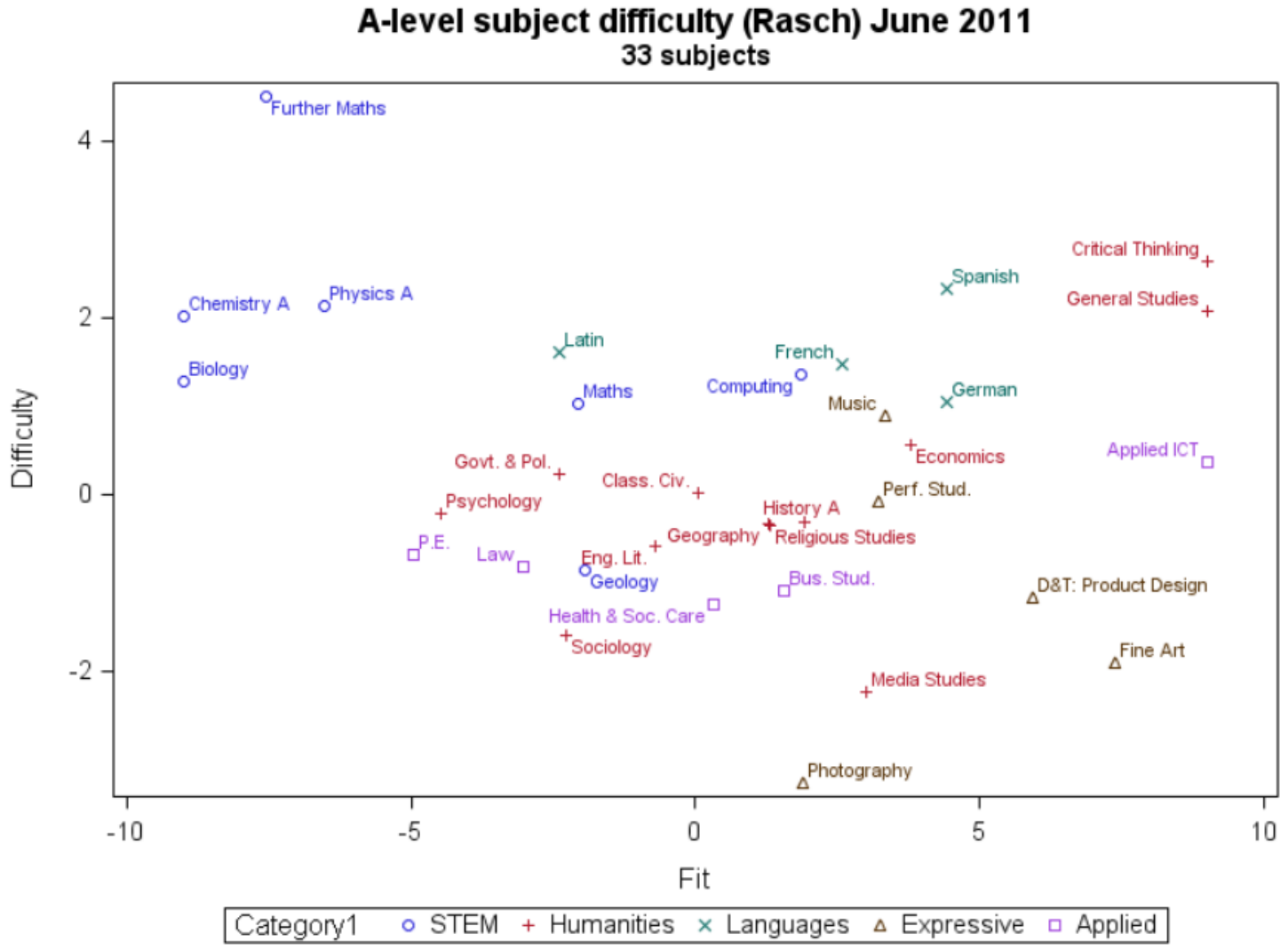






Multidimensional Scaling (MDS)

- MDS is a dimensionality reduction technique
- Aims at mapping high-dimensional data onto a lower-dimensional space
- Similarities between points in high dimensional (original) space are preserved in lower dimensional (new) space
- 2 cases of MDS
  - Metric MDS: Uses a metric to measure distance between points in original space
    - E.g., Euclidean, Manhattan, Mahalanobis, etc.
  - Non-metric MDS: Any kind of function is used to measure distance between points
    - E.g., Correlation, No. of occurrences, Edit distance, BLAST Alignment Score, etc.



Metric MDS

Input: $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$, where $\mathbf{x}_i \in \mathbb{R}^d$

- Let d_{ij} be the distance between $\mathbf{x}_i$ and $\mathbf{x}_j$
- d_{ij} is a metric
- Typically, Euclidean distance is used
- $d_{ij} = \|\mathbf{x}_i - \mathbf{x}_j\|$
- Other metrics can be used too

Aim:

- Find k-dimensional representation of $\mathbf{X}$:

$$\mathbf{Y} = \{\mathbf{y}_1, \mathbf{y}_2, \dots, \mathbf{y}_n\}$$

where $\mathbf{y}_i \in \mathbb{R}^k$

- by minimizing the stress function:

$$S_M(\mathbf{y}_1, \mathbf{y}_2, \dots, \mathbf{y}_n) = \sum_{i \neq i'} (d_{ii'} - \|\mathbf{y}_i - \mathbf{y}_{i'}\|)^2$$

Known as Kruskal-Shephard scaling

- S_M can be minimized via gradient descent algorithm

Variation of least squares:

- Sammon mapping, which minimizes:

$$S_{Sm}(\mathbf{y}_1, \mathbf{y}_2, \dots, \mathbf{y}_n) = \sum_{i \neq i'} \frac{(d_{ii'} - \|\mathbf{y}_i - \mathbf{y}_{i'}\|)^2}{d_{ii'}}$$

Better for preserving smaller pairwise distances

Why?

- Local similarities are important

Classical MDS

- Mean vector in original space:

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

Mean vector in new space:

$$\mathbf{m}_y = \sum_{i=1}^{n} \mathbf{y}_i$$

- Instead of samples, start with similarities using inner product:

$$s_{ii'} = (\mathbf{x}_i - \mathbf{m}_x)^t(\mathbf{x}_{i'} - \mathbf{m}_x)$$

- Aim is to minimize:

$$S_C(\mathbf{y}_1, \mathbf{y}_2, \dots, \mathbf{y}_n) = \sum_{i,i'} \left(s_{ii'} - (\mathbf{y}_i - \mathbf{m}_x)^t(\mathbf{y}_{i'} - \mathbf{m}_y)\right)^2$$

over $\mathbf{y}_i \in \mathbb{R}^k$

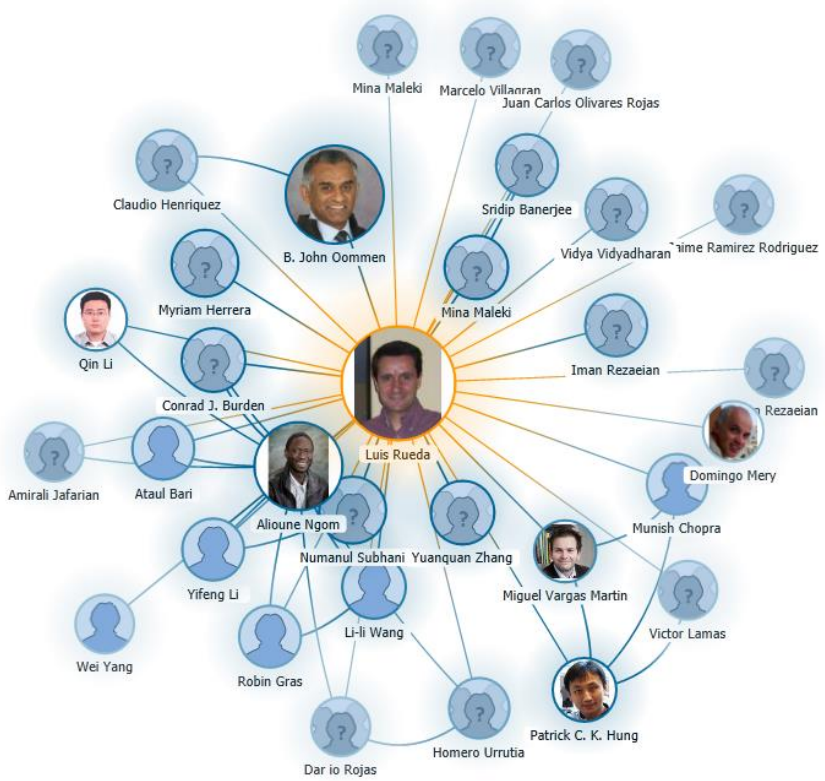
- Intereresting… because
- it can be shown that it's equivalent to find the largest eigenvalues of

$$\mathbf{S} = \{(\mathbf{x}_i - \mathbf{m}_x)^t(\mathbf{x}_{i'} - \mathbf{m}_x)\}_{ii'}$$

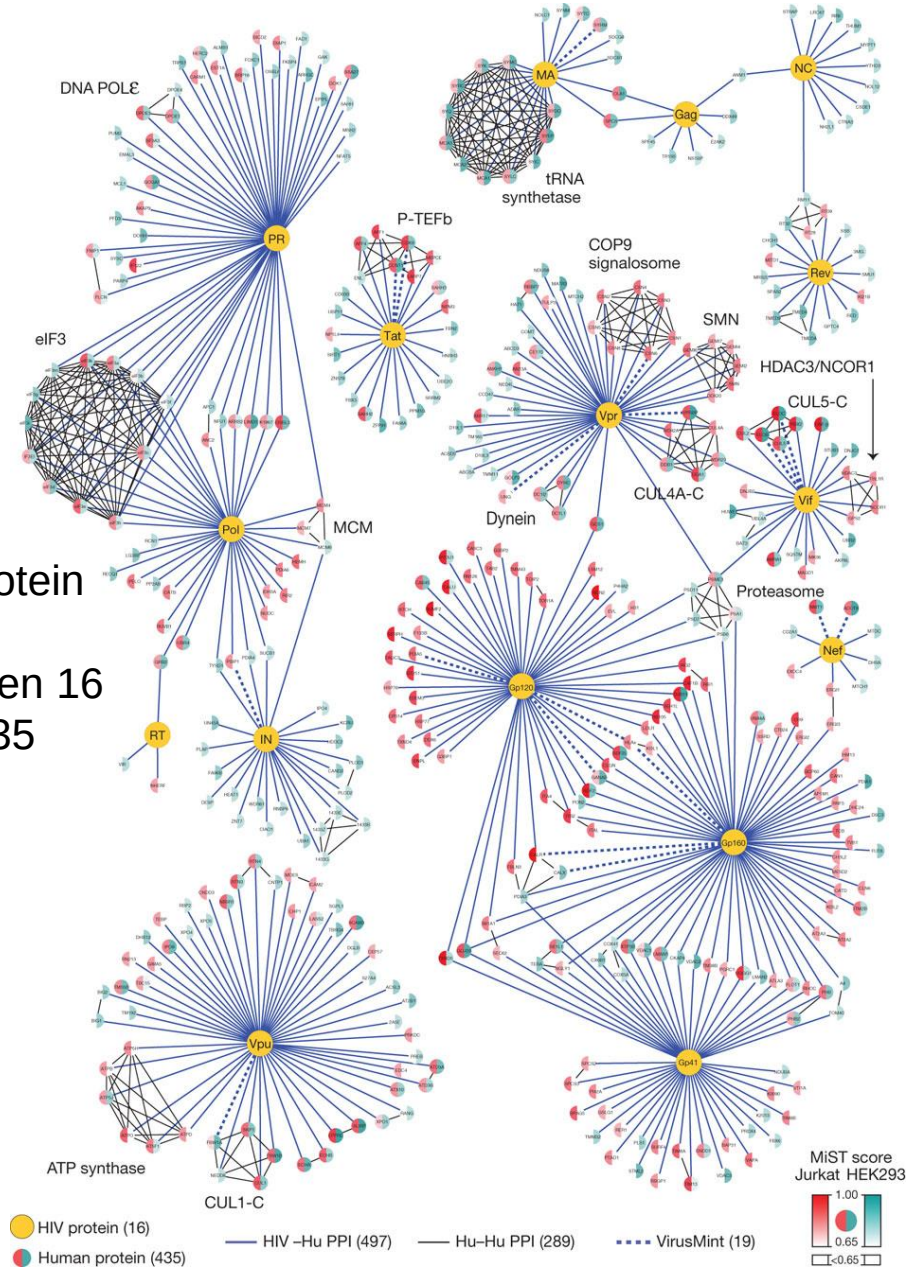
- and choose the corresponding eigenvectors
- Then, metric MDS is similar to PCA, and also to SVD
- However, life's not that easy in many real problems:
  - Network data: PPI, social networks, semantics of words, protein complexes
  - Graphs in general, in which data lies in very high-dimensional spaces
  - Relationships among text or strings: e.g., similarity between two DNA sequences via alignment or between strings via Edit distance

Real problems – Non-metric Data

- Research gate



497 HIV–human protein interactions (blue) representing between 16 HIV proteins and 435 human factors [6]



Real problems – Non-metric Data

- **Edit distance between two strings:**

  X = *excused* is transformed into

  Y = *exhausted*

- Steps:
  - First, substitute *h* for *c*, yielding
    X = *exhused*
  - Second, insert *a*, obtaining
    X = *exhaused*
  - Third, insert *t*, obtaining
    X = *exhausted*
  - Since the cost of each operation is 1, the distance between X and Y is 3

DNA sequence alignment

- Uses a scoring system that assigns:
  - score values for **each** pair of symbols
  - e.g. nucleic acid or amino acid pairs
  - penalty values to single gaps
- Score = sum of pair scores – sum of gap penalties

Example: *X* = **TGATAGCCAG** *Y* = **AGAGCA**

```
T - G A T A G C C A G
- A G A G - - C - A -
```

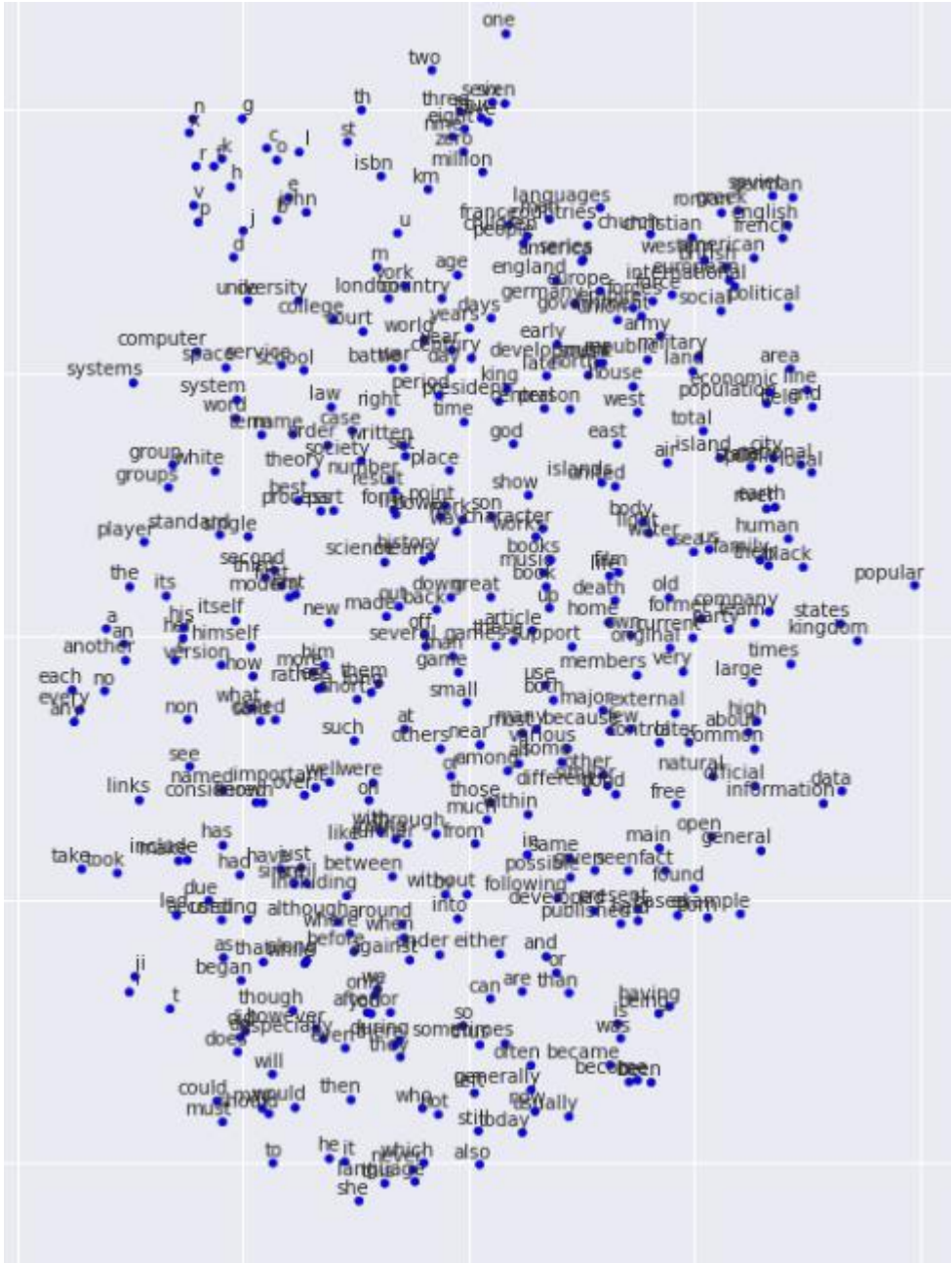
Score = -4-4+6+6+2-4-4+6-4+6-4 = 2

Non-metric Data

List of words:

…
can
college
common
computer
could
….

Embedding in 2D space

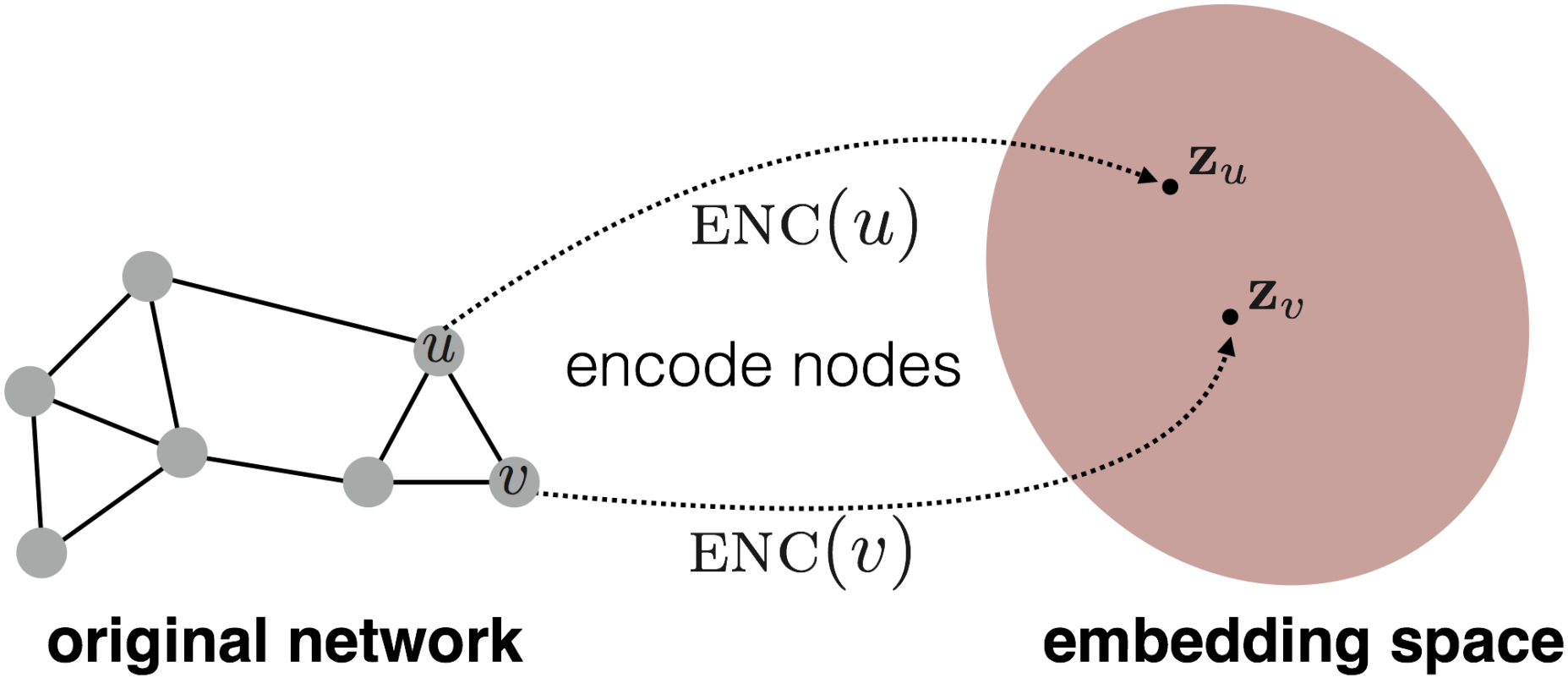


- Words in alphabetical (or some other) order
- Similarity based on meaning, rather than on spelling
  - Non-metric (e.g., non-Euclidean) in higher dimensional space

https://www.tensorflow.org/tutorials/representation/word2vec ref [9]

Real problems – Node Embedding in Graphs

- Aim is to encode nodes [10]
- Similarity in original network (graph) preserved in embedding space



Non-metric MDS

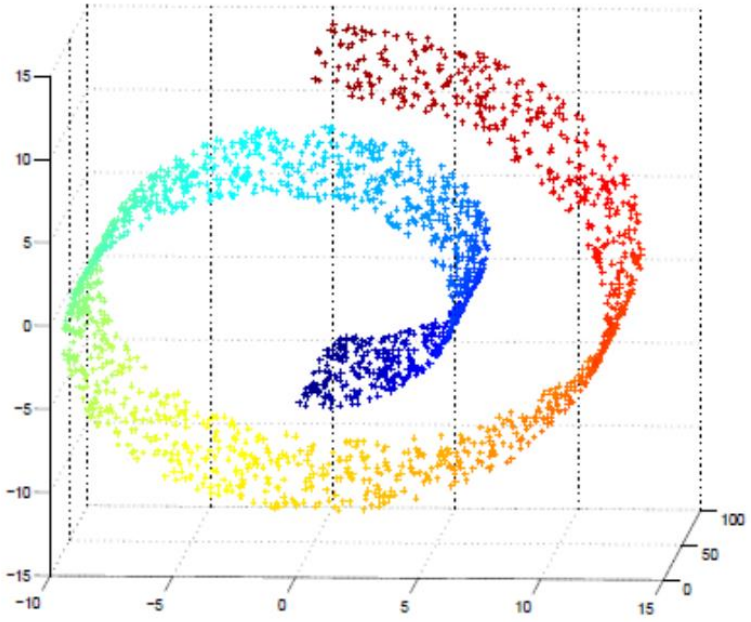
- Shephar-Kruskal non-metric MDS works with any distance measure
- Only requires a matrix distance $\{\Delta_{ij}\}$
- Aim: minimize the stress function:

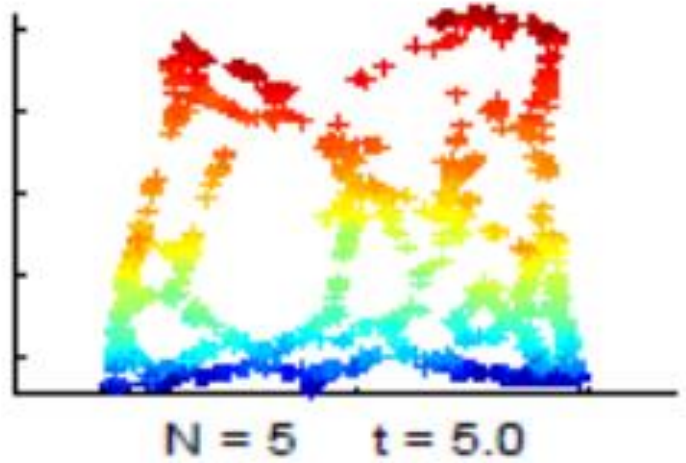
$$S_M(\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_n) = \frac{\sum_{i \neq i'} \left[\|\mathbf{y}_i - \mathbf{y}_{i'}\| - g(\Delta_{ij})\right]^2}{\sum_{i \neq i'} \|\mathbf{y}_i - \mathbf{y}_{i'}\|^2}$$

over $\mathbf{y}_i$, where g is an arbitrarily increasing function

- If g is fixed, S_M can be minimized using the gradient descent
- with $\mathbf{y}_i$ fixed, isotonic regression can be used to find best monotonic approximation of $g(\Delta_{ij})$ to $\|\mathbf{y}_i - \mathbf{y}_{i'}\|$

These two steps repeated until desired solution is found





Local MDS

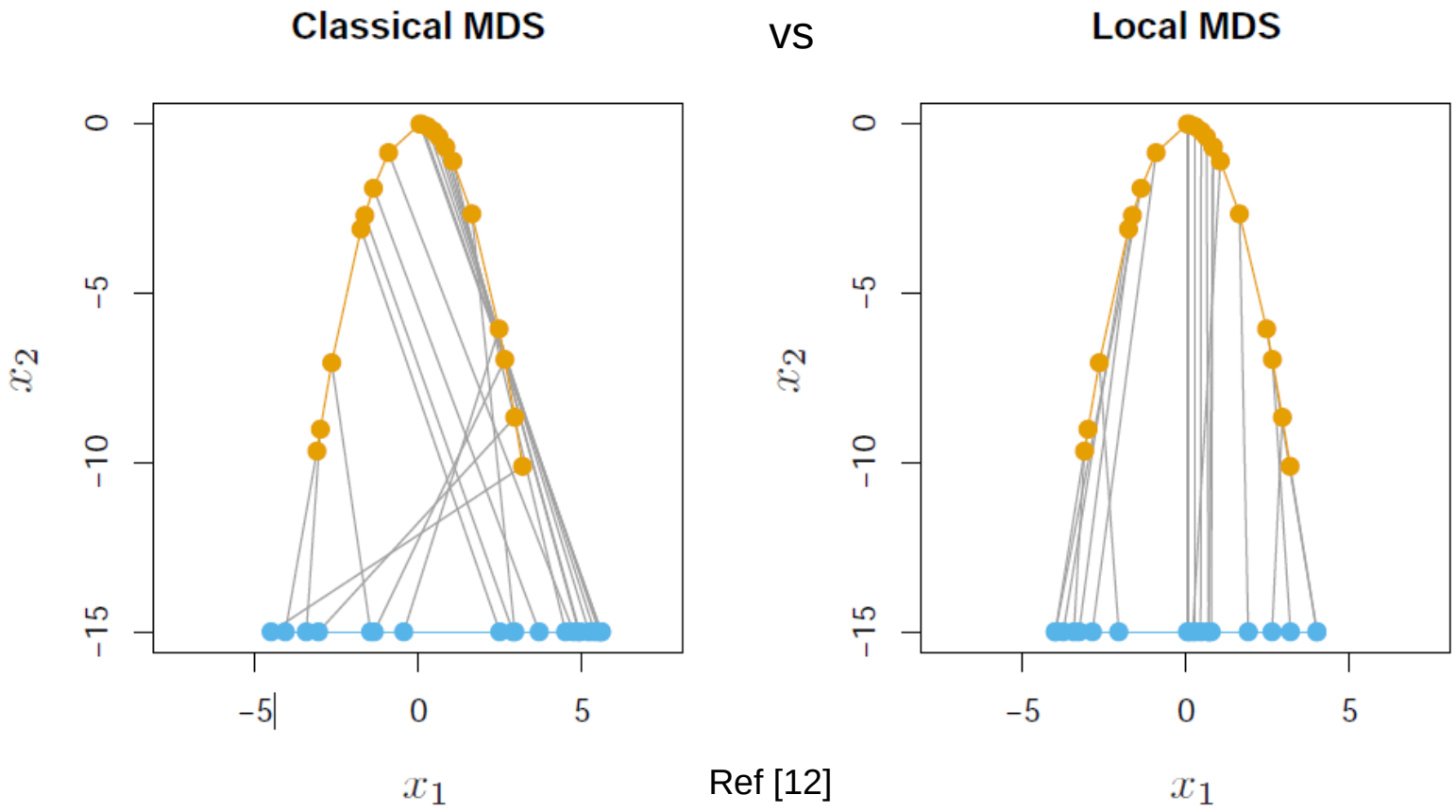
- Local MDS is good for manifold representation and graph embedding
- Aim for nonlinear dimensionality reduction and for capturing neighborhood relationships in high-dimensional and/or complex data

Main approaches:

- Isomap
- Laplacian eigenmaps
- Local linear embedding (LLE)

Example:

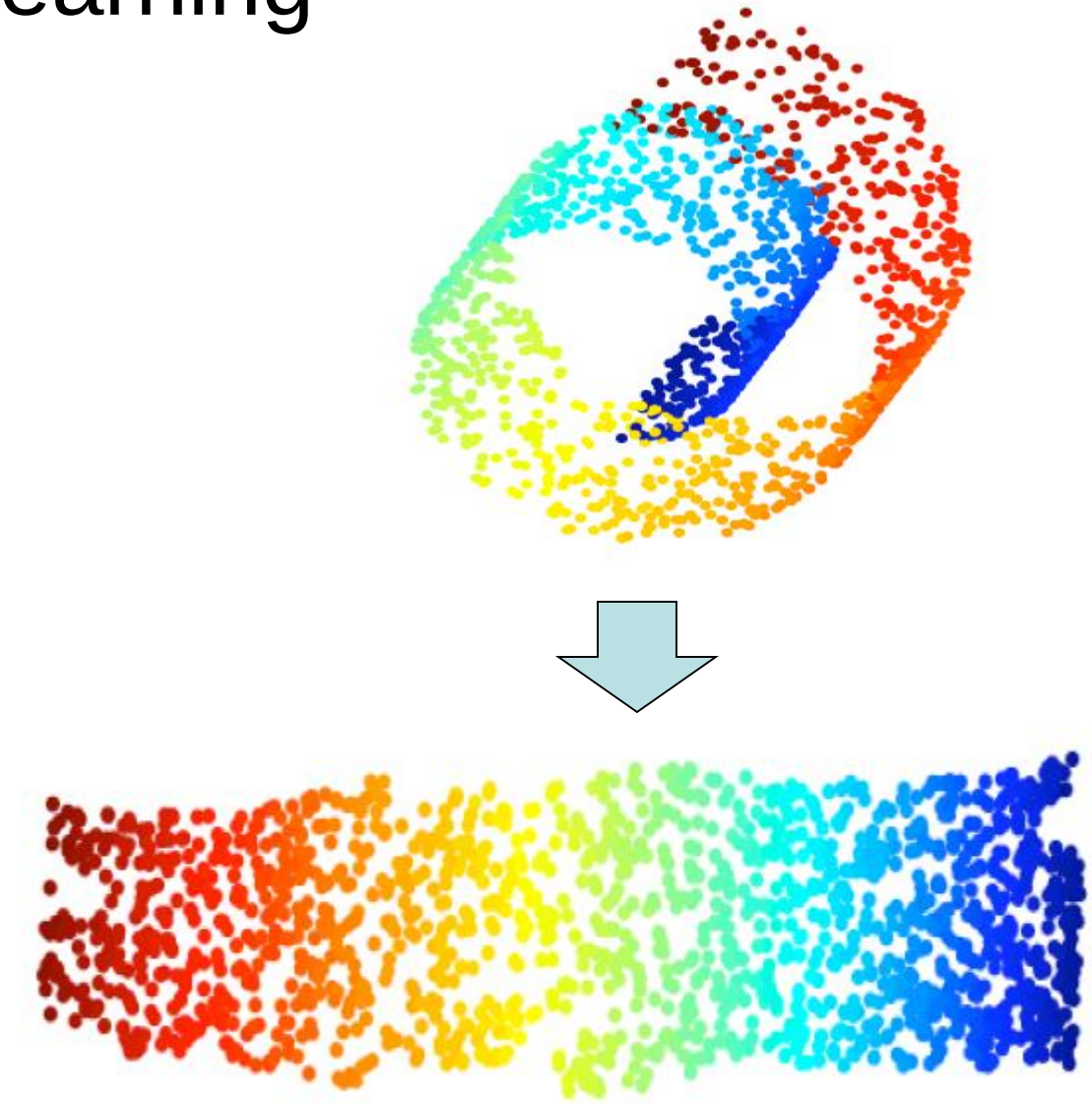
- Data lies in parabola in 2D
- Projected onto 1D space



Ref [12]

Manifold Learning

- Manifolds are techniques for NDR
- Assume higher dimensional data lie on a lower dimensional space via nonlinear mapping
- Transformation called “manifold embedding”
- Can be seen as special case of KPCA
- Most well-known techniques
  - Isomap
  - Laplacian eigenmaps
  - Locally linear embedding (LLE)

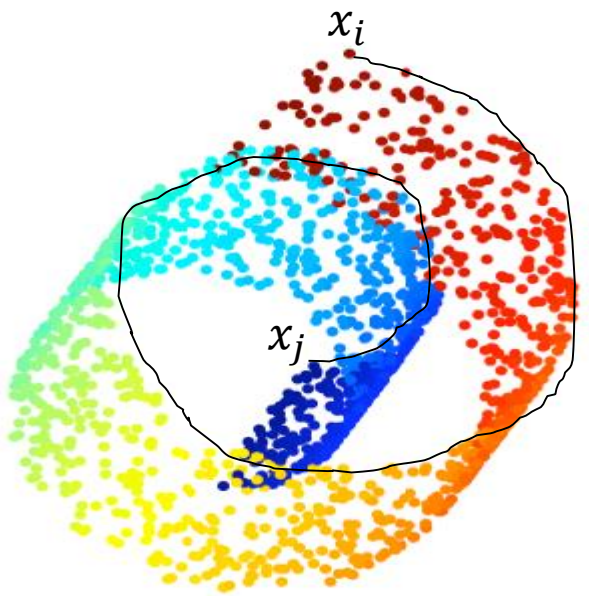


Isomap

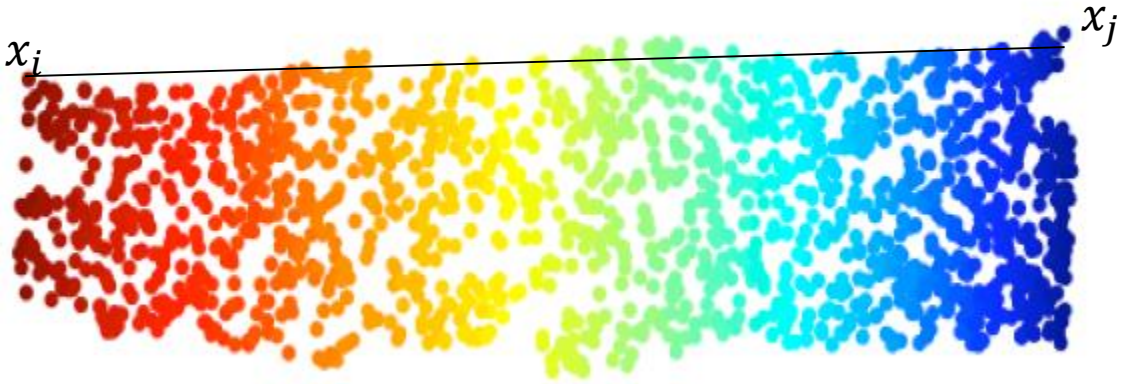
- Extract low dimensional representation of data
- Preserve pairwise distance between points measured by geodesic distances
- Use L_2-norm to measure distances
- Use nearest neighbors only (not all points)

Main steps:

- Find t nearest neighbors
- Construct adjacency graph
- Compute approximate geodesic distances
- Find singular values via eigen decomposition



Unroll the Swiss roll



Isomap - Algorithm

Input: $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$

Step 1:

- Find t nearest neighbors for each point $\mathbf{x}_i$
- Use L_2-norm to construct undirected neighborhood graph $G = (V, E)$
- V are the points as vertices
- E are the links between neighbors as edges

Step 2:

- Compute approximate geodesic distances Δ_{ij} for all pairs of vertices $(\mathbf{x}_i, \mathbf{x}_j)$
- Find all-pairs shortest distances in G using dynamic programming (e.g., Floyd-Warshall algorithm)

Step 3:

- Convert squared distance matrix into $m \times m$ similarity matrix:

$$\mathbf{K}_{Iso} = -\frac{1}{2}\mathbf{H}\Delta\mathbf{H}$$

where

- Δ is the squared distance matrix
- $\mathbf{H} = \mathbf{I}_m - \frac{1}{m}\mathbf{1}\mathbf{1}^t$ is the centering matrix
- $\mathbf{I}_m$ is the $m \times m$ identity matrix
- $\mathbf{1}$ is a column vector of all ones

Isomap

Step 4:

- Find optimal k-dimensional representation

$$\mathbf{Y} = \{\mathbf{y}_1, \mathbf{y}_2, \dots, \mathbf{y}_n\}$$

such that:

$$\mathbf{Y} = \underset{\mathbf{Y}'}{\operatorname{argmin}} \sum_{i,j} \left(\left\| \mathbf{y}'_i - \mathbf{y}'_j \right\|_2^2 - \Delta_{ij}^2 \right)$$

Solution given by:

- $\mathbf{Y} = \left(\mathbf{\Sigma}_{Iso,k}\right)^{1/2} \mathbf{U}_{Iso,k}^t$

where

- $\mathbf{\Sigma}_{Iso,k}$ is the diagonal matrix of the top k singular values of $\mathbf{K}_{Iso}$ and
- $\mathbf{U}_{Iso,k}^t$ are the associated singular vectors
- $\mathbf{K}_{Iso}$ can be viewed as a kernel matrix
- If positive semidefinte, Isomap resembles KPCA (typical of a smooth manifold)

Complexity

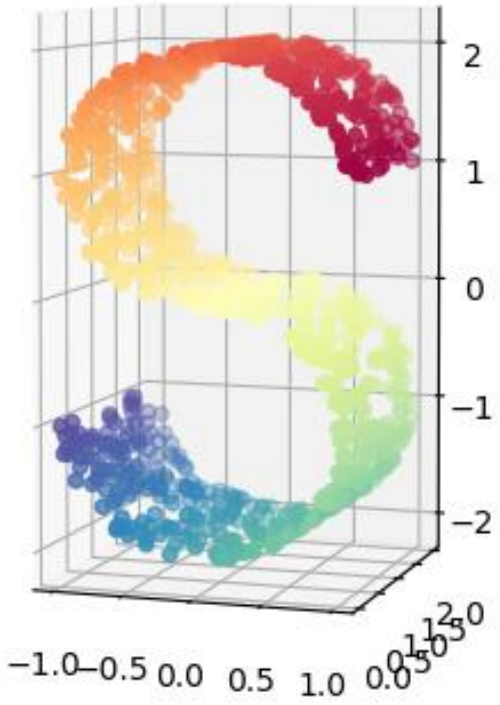
- t-NN is $O(n^2 d)$
- Djikstra's for shortest path: $O(n^2 \log n + n^2 k)$
- Embedding: top k eigenvectors found in $O(n^2 k)$
- Optimizations can be done via approximation

Big Data

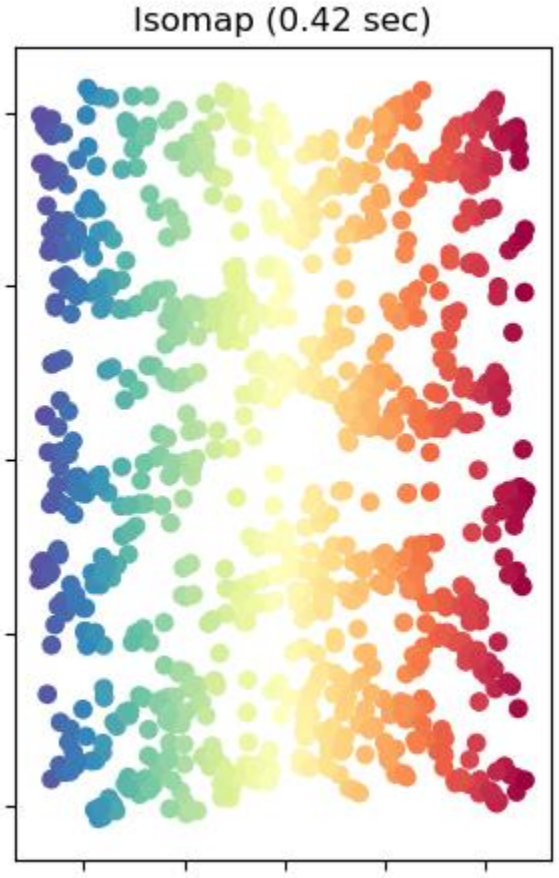
- Too expensive to find all shortest paths
- Can compute shortest paths in part of Gram matrix
- Use diagonalization of sub-matrix

Isomap – Example: S data

- Original data, S-shaped in 3D



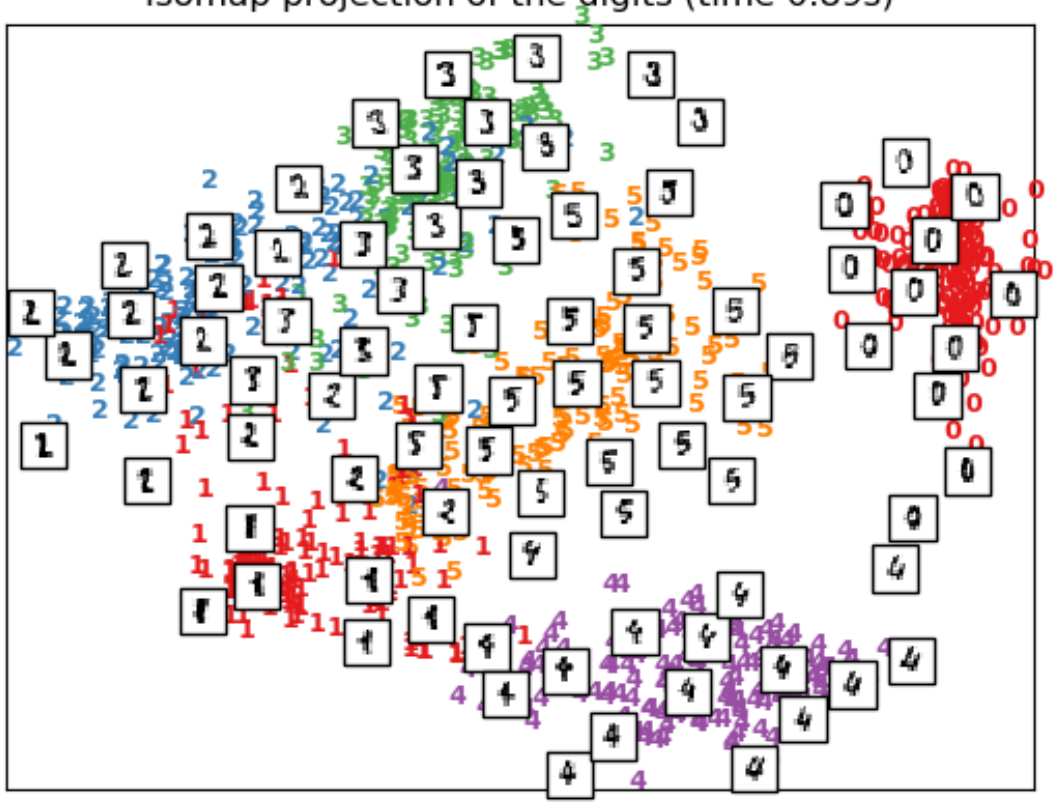
- Isomap
- $t = 10$



Isomap – Digits example

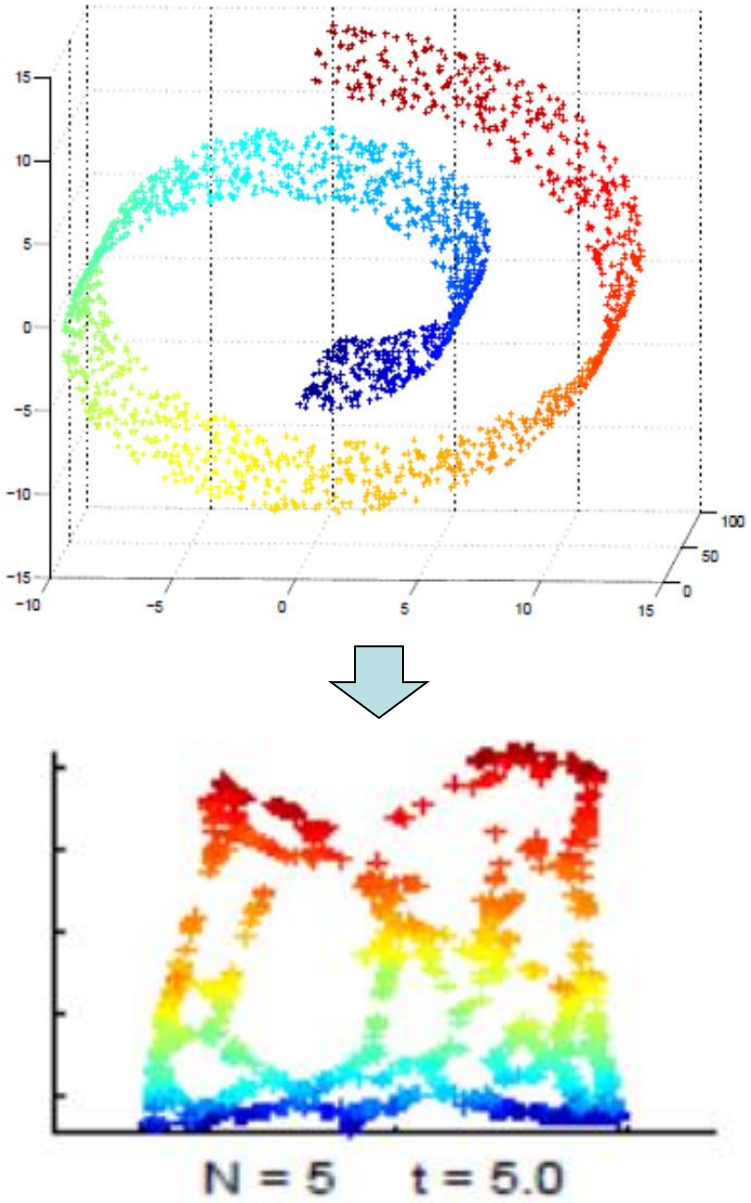
A selection from the 64-dimensional digits dataset

Isomap projection of the digits (time 0.89s)



Laplacian Eigenmaps

- Key idea: preserve local information only
- Find lower-dimensional representation
- Try to preserve neighborhood relations in lower dimension
- Use information of the spectral (frequency) domain
- Derive a weight matrix $\mathbf{W}$
- Matrix constructed via a kernel
  - Most commonly used kernel is RBF
- Perform eigen decomposition of the graph Laplacian matrix
- Can be applied to high-dimensional non-Euclidean weighted graphs



Laplacian Eigenmaps

Input: $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$

Step 1:

- Find t nearest neighbors for each point $\mathbf{x}_i$

Step 2:

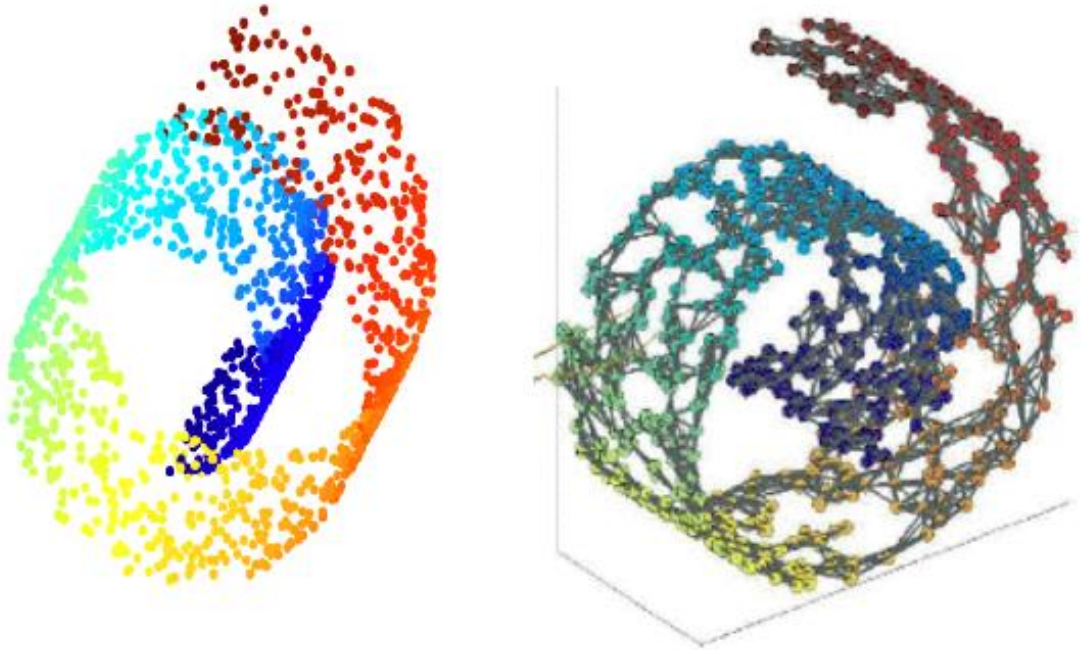
- Construct undirected neighborhood graph $G = (V, E, \mathbf{W})$
- V are the points as vertices
- E are the links between neighbors as edges
- $\mathbf{W}$ is a sparse, symmetric $\mathrm{n} \times n$ matrix where $w_{ij} = k(x_i, x_j)$ if $\mathbf{x}_i$ and $\mathbf{x}_j$ are neighbors
  = 0 otherwise

Commonly used kernel: RBF

$$k(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{\sigma^2}\right)$$

- Sparsity of G depends on t
  - Small $t \Rightarrow G$ is very sparse
  - Large $t \Rightarrow G$ is very dense

- Example: From Swiss roll to Sparse Graph



Laplacian Eigenmaps

Step 3:

- Construct the diagonal matrix $\mathbf{D}$

  where $d_{ij} = \sum_j w_{ij}$

- It's basically the degree of each node placed on the diagonal of $\mathbf{D}$

Step 4:

- Find the k-dimensional representation of the data
- Minimize the weighted distance between neighbors in the new space:

$$\mathbf{Y} = \underset{\mathbf{Y}'}{\operatorname{argmin}} \sum_{i,j} w_{ij} \left\| \mathbf{y}'_i - \mathbf{y}'_j \right\|_2^2$$

- Neighbor inputs in original space are mapped to points that are close to each other in new space, using info in $\mathbf{W}$

Solution:

$$\mathbf{Y} = \mathbf{U}_{L,k}^t$$

where

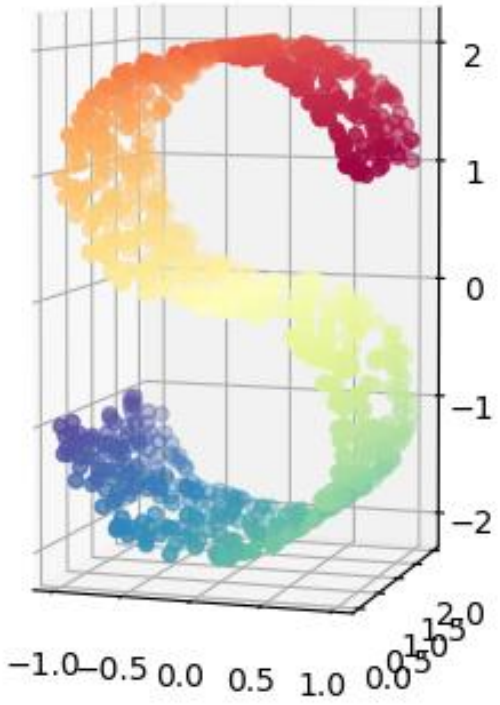
- $\mathbf{L} = \mathbf{D} - \mathbf{W}$ is the graph Laplacian
- $\mathbf{U}_{L,k}^t$ are the bottom k singular vectors (associated smallest eigenvalues)
  excluding the singular value 0

Interpretation:

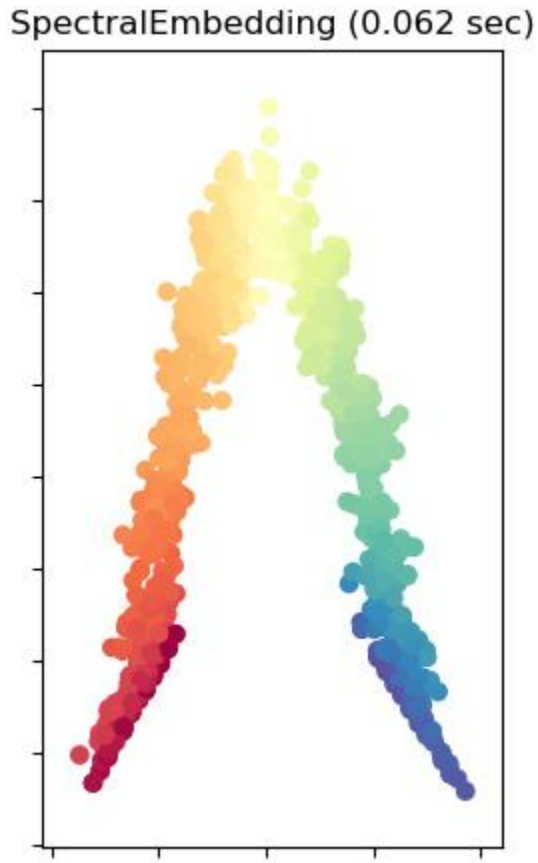
- Find the largest singular vectors of $\mathbf{L}^+$, the pseudoinverse of $\mathbf{L}$
- If we consider $\mathbf{K}_L = \mathbf{L}^+$, we can view Laplacian Eigenmaps as a particular case of KPCA

Laplacian Eigenmaps – Example: S data

- Original data, S-shaped in 3D

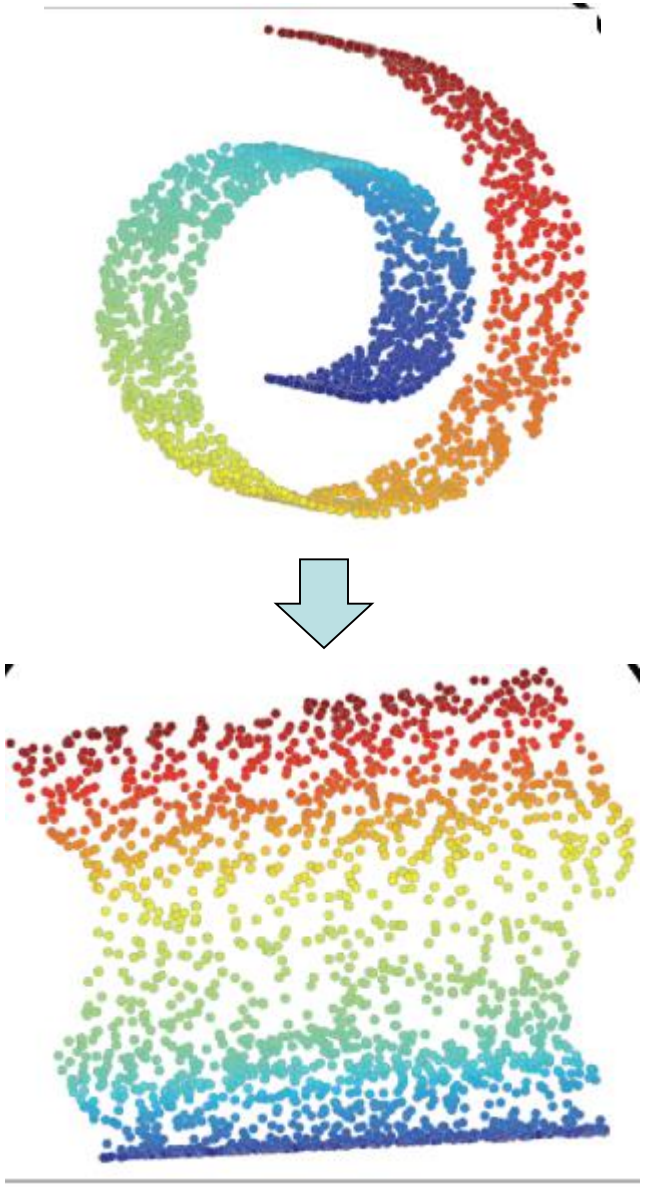


- Laplacian eigenmaps
- $t = 10$



Local Linear Embedding (LLE)

- Also aims at mapping high dimensional data to lower dimensional data
- Preserves the local linear structure of nearby input data points
- Unlike the Isomap, the outputs are derived from the bottom eigenvectors of a sparse matrix
- Also based on nearest neighbors
- Constructs the sparse matrix based on distances to those neighbors



Local Linear Embedding (LLE)

Input: $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$

Step 1:

- Find t nearest neighbors for each point $\mathbf{x}_i$

Step 2:

- Construct sparse, symmetric $n \times n$ matrix $\mathbf{W}$
- i^{th} row of $\mathbf{W}$ sums to one
- The reconstruction error is:

$$\left(\mathbf{x}_i - \sum_{j \in \mathcal{N}_i} w_{ij}\mathbf{x}_j\right)^2 = \left(\sum_{j \in \mathcal{N}_i} w_{ij}(\mathbf{x}_i - \mathbf{x}_j)\right)^2$$

$$= \sum_{j,k \in \mathcal{N}_i} w_{ij} w_{ik} \mathbf{C}'_{jk} \qquad (1)$$

- $\mathcal{N}_i$ is the set of indices of the of the neighbors of x_i
- $\mathbf{C}'_{jk} = (\mathbf{x}_i - \mathbf{x}_j)(\mathbf{x}_i - \mathbf{x}_k)^t$ is the local covariance matrix

- Minimizing (1) with constraint $\sum_j w_{ij} = 1$ gives solution:

$$w_{ij} = \frac{\sum_k (\mathbf{C}')^{-1}_{jk}}{st \sum (\mathbf{C}')^{-1}_{st}}$$

Step 4:

- Find the k-dimensional representation of the data
- Minimize the weighted distance between neighbors in the new space:

$$\mathbf{Y} = \underset{\mathbf{Y}'}{\operatorname{argmin}} \sum_i \left(\mathbf{y}'_i - \sum_j w_{ij} \boldsymbol{y}'_j\right)^2 \qquad (2)$$

Local Linear Embedding (LLE)

Solution:

$$\mathbf{Y} = \mathbf{U}_{M,k}^{t}$$

where

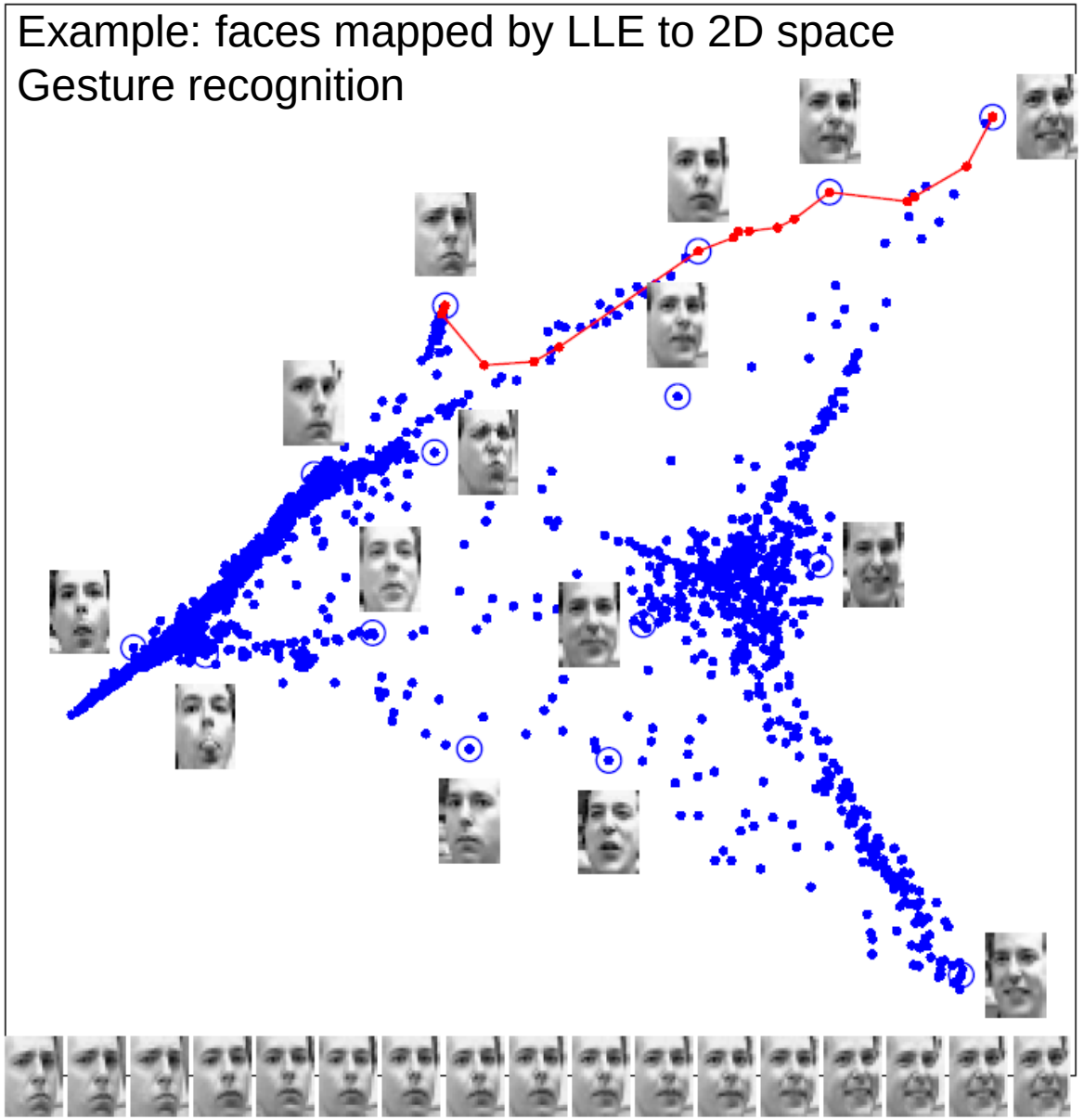
- $\mathbf{M} = (\mathbf{I} - \mathbf{W}^t)(\mathbf{I} - \mathbf{W}^t)$
- $\mathbf{U}_{M,k}^{t}$ are the bottom k singular vectors (associated smallest eigenvalues)
  excluding the singular value 0

- LLE is equivalent to KPCA if the kernel matrix $\mathbf{K}_{LLE}$ is given by the pseudoinverse of $\mathbf{M}$

Variant of LLE:

- Choose $\ell < n$ random points constrained to equal the corresponding inputs
- For ℓ sufficiently large, (2) can be minimized by solving a MSE problem

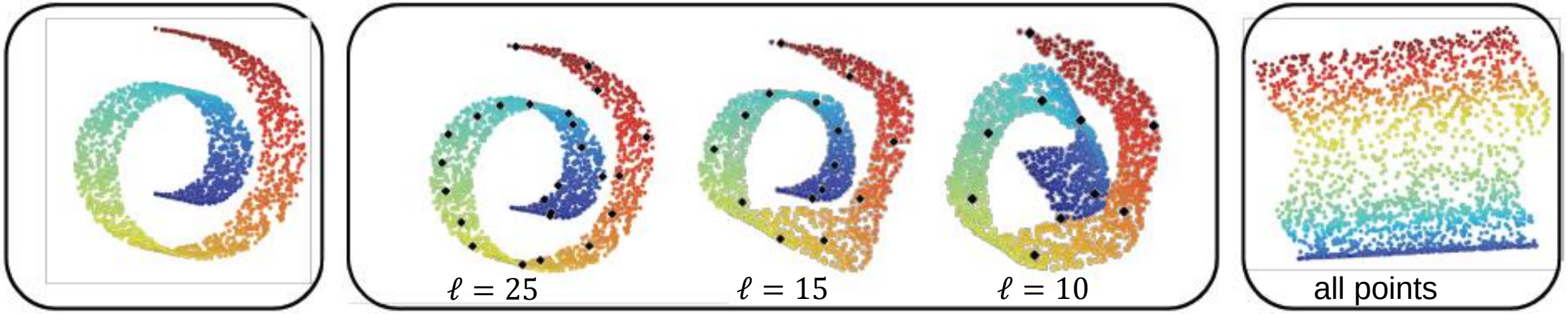
Example: faces mapped by LLE to 2D space
Gesture recognition



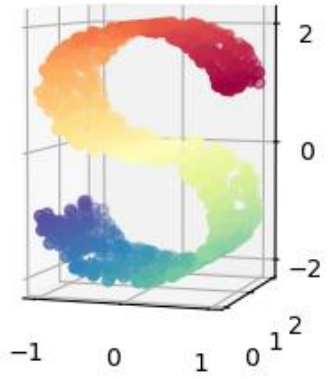
LLE Example – Effect of ℓ

Swiss roll data

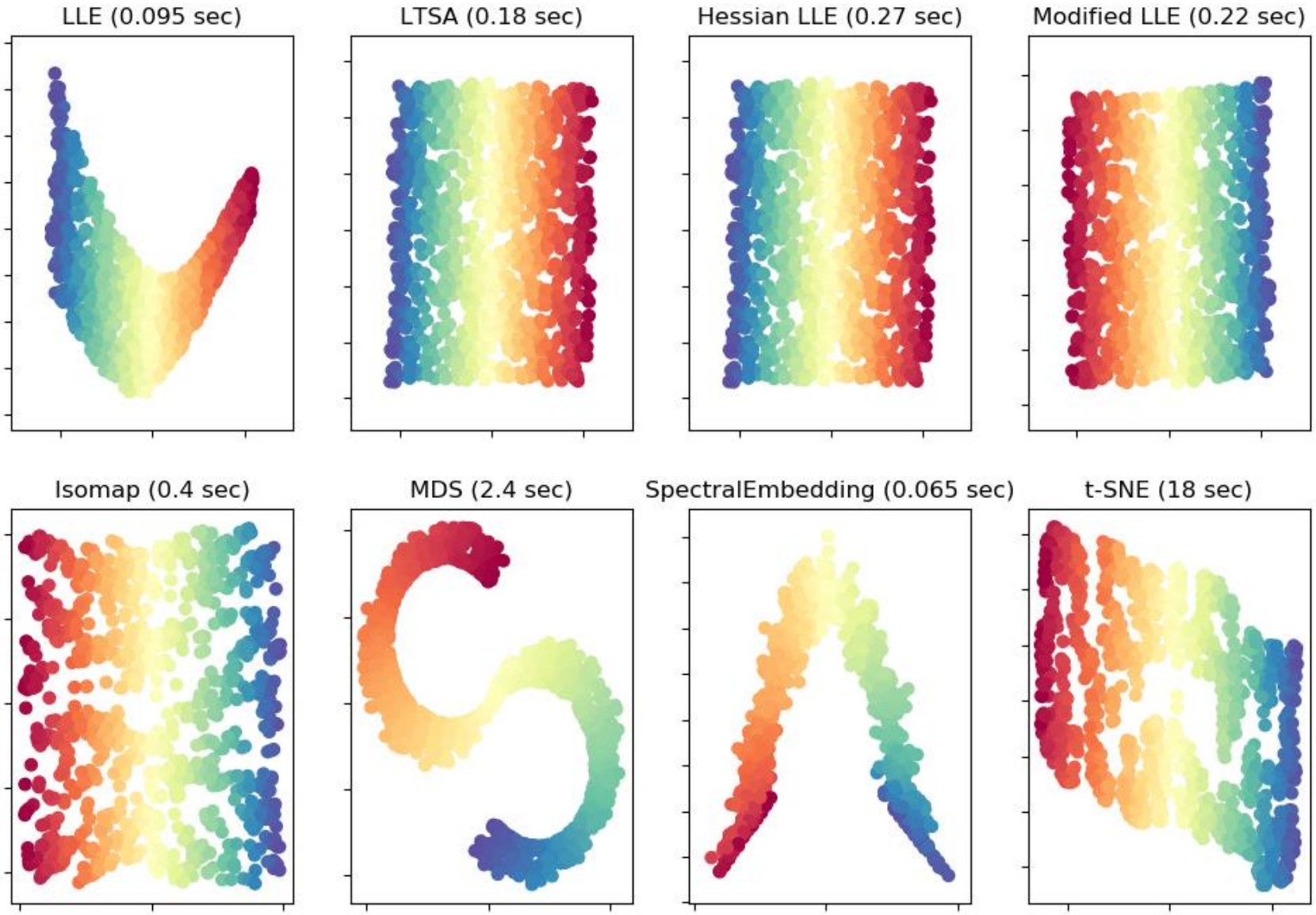
$t = 20$ nearest neighbors



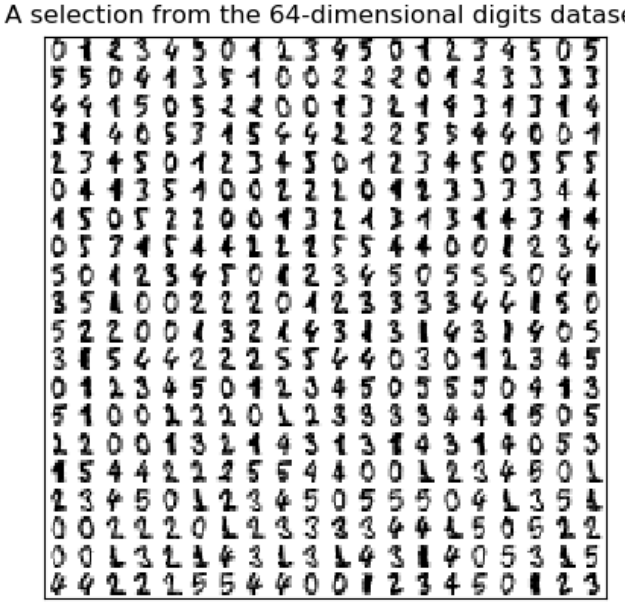
Manifold – S data - Comparison

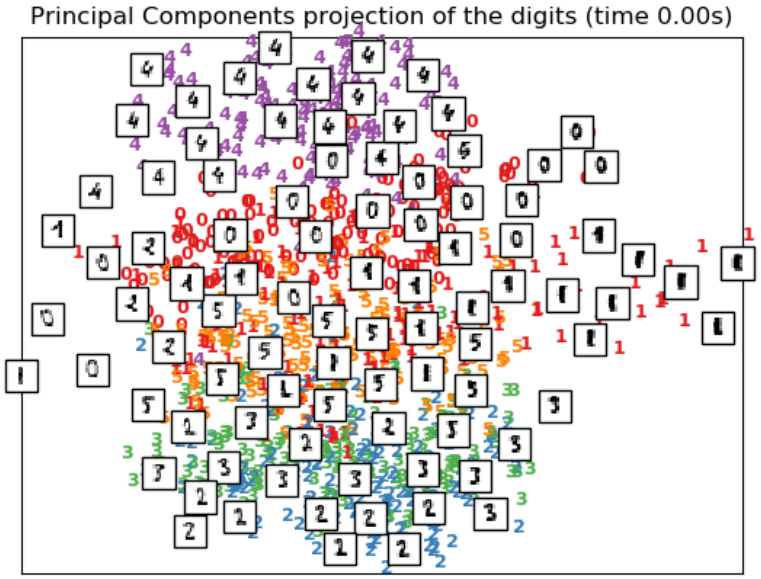


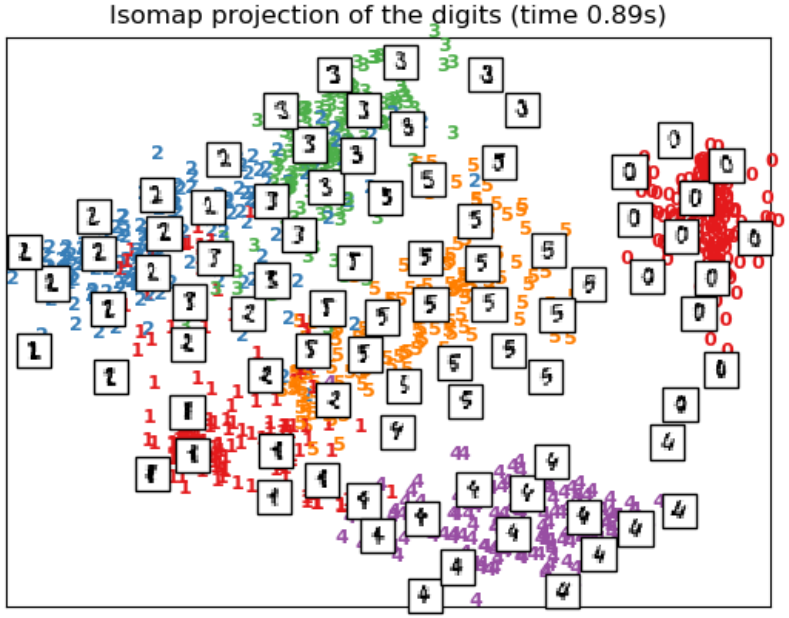
$t = 10$
nearest neighbors

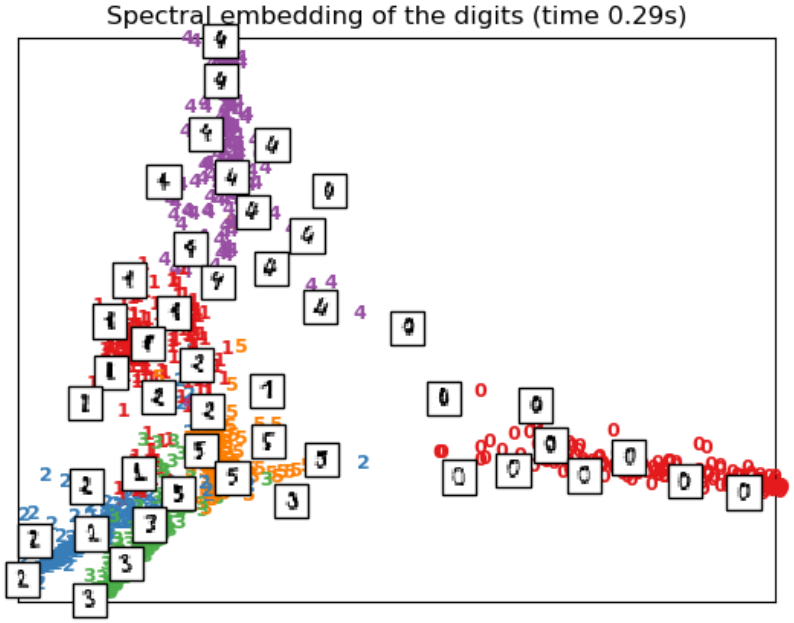


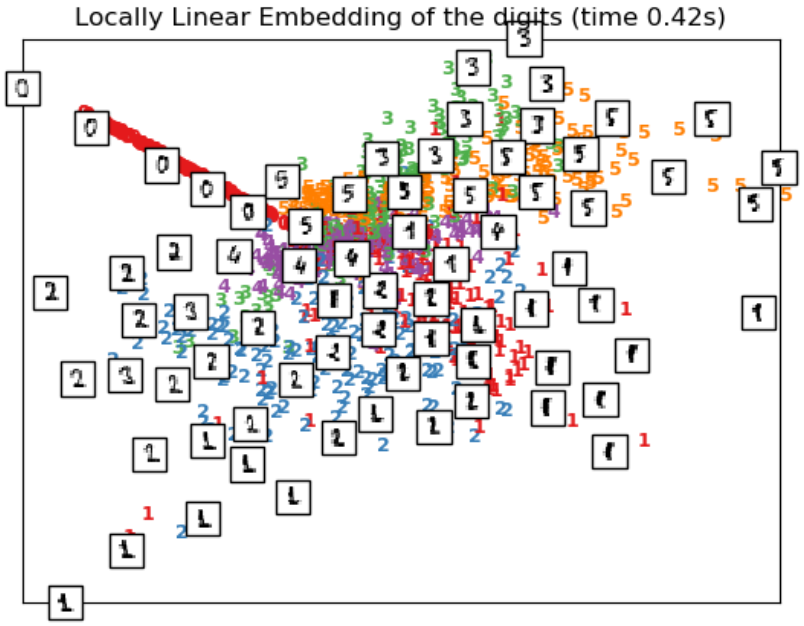
Digits: PCA vs Isomap, LLE, Laplacian, MDS

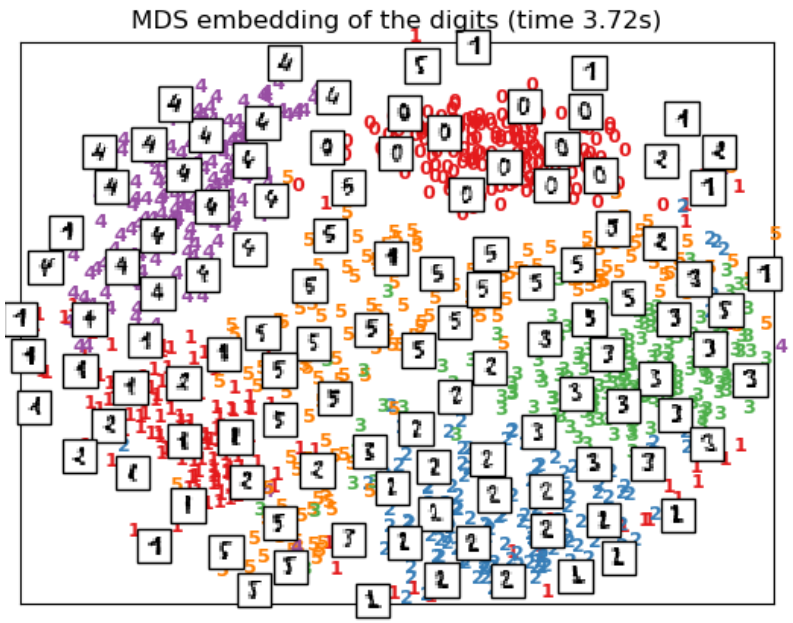












Nonlinear Autoencoder

Autoencoder:

- Each layer parameterizes a nonlinear transformation

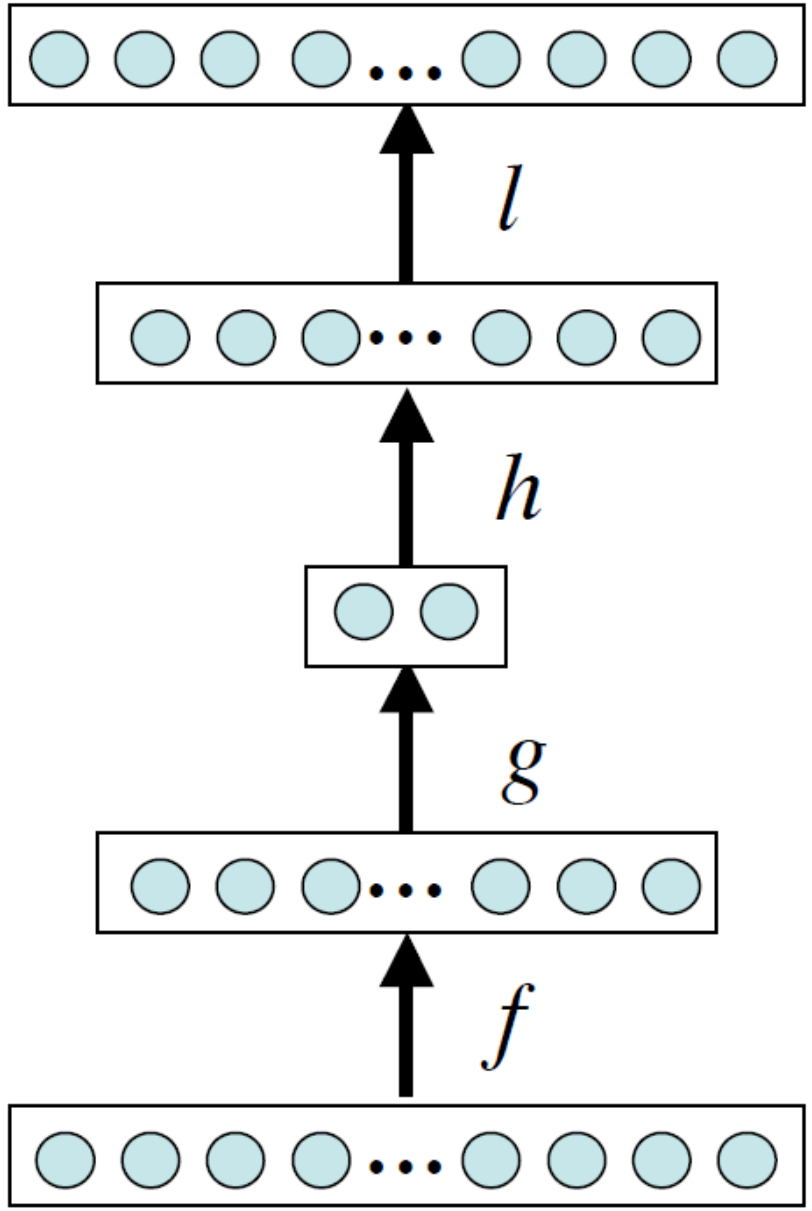
Cost function:

- Minimize the reconstruction error

$$\mathcal{L} = \sum_{\mathbf{x}_i \in D} \left\| \mathbf{x}_i - l_w(h_w(g_w f(\mathbf{x}_i))) \right\|_2^2$$

here, f and g are encoding functions

while h and l are decoding functions



Linear Discriminant Analysis

- Given a dataset $D = D_1 \cup D_2 \cup \ldots \cup D_k$ of n labeled samples, which belong to k classes:

$$D_1 = \{\mathbf{x}_{11}, \mathbf{x}_{12}, \ldots, \mathbf{x}_{1n_1}\}$$

$$D_2 = \{\mathbf{x}_{21}, \mathbf{x}_{22}, \ldots, \mathbf{x}_{2n_2}\}$$

$$\ldots$$

$$D_k = \{\mathbf{x}_{k1}, \mathbf{x}_{k2}, \ldots, \mathbf{x}_{kn_k}\}$$

where:

- each $\mathbf{x}_{ij}$ that $\in$ to class ω_i

$$n = \sum_{i=1}^{k} n_i$$

Consider the linear transformation $\mathbf{y} = \mathbf{A}\,\mathbf{x}$,

where $\mathbf{A}$ is an $m \times d$ matrix

- The projection is onto the m-dimensional space
- Then, the samples $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n$ in the d-dim. space
  - correspond to $\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_n$ in the m-dim. space
  - where $m < d$ (ideally, $m <<< d$)
- Once projection/transformation is performed:
  - Classify the data using *any* classifier

Assumptions

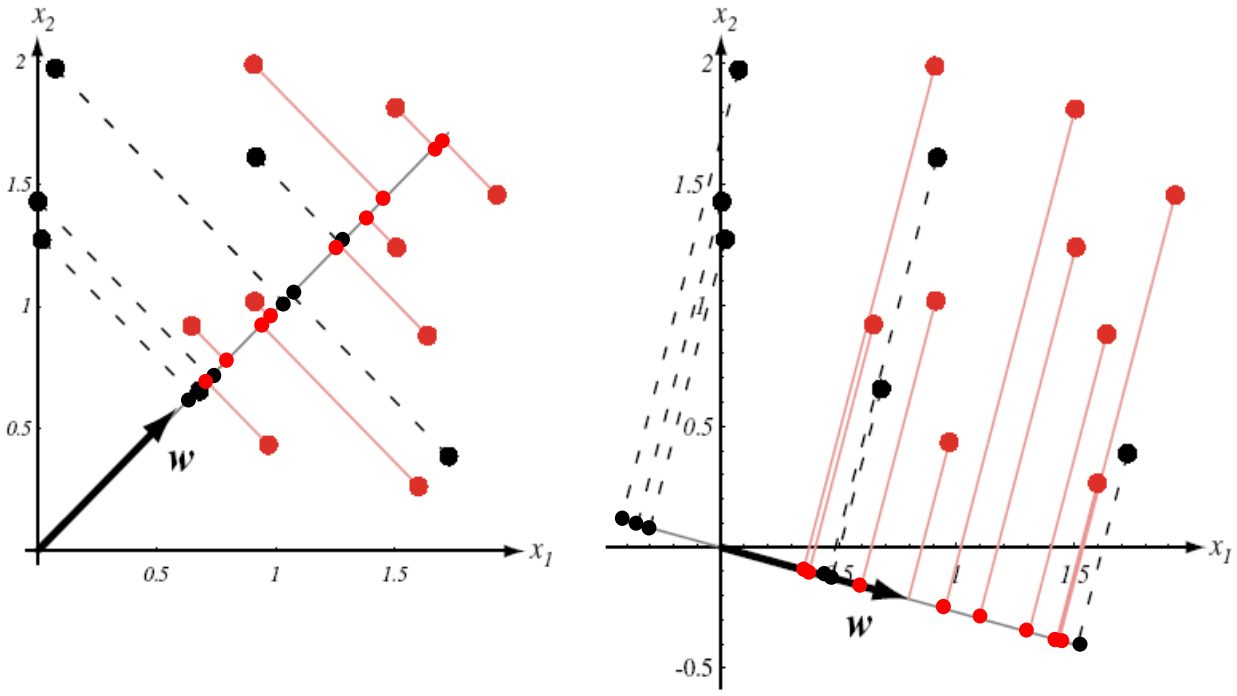
- Prior probabilities known:
  - $p_1, p_2, \ldots, p_k$
- Means known:
  - $\mathbf{m}_1, \mathbf{m}_2, \ldots, \mathbf{m}_k$
- Covariances known:
  - $\mathbf{S}_1, \mathbf{S}_2, \ldots, \mathbf{S}_k$
- If not known, estimate them!
- So, we have:

$$p_i = \frac{|D_i|}{|D|}, \quad \mathbf{m}_i = \frac{1}{|D_i|}\sum_{\mathbf{x}\in D_i}\mathbf{x}, \qquad \mathbf{S}_i = \frac{1}{|D_i|}\sum_{\mathbf{x}\in D_i}(\mathbf{x}-\mathbf{m}_i)(\mathbf{x}-\mathbf{m}_i)^t$$

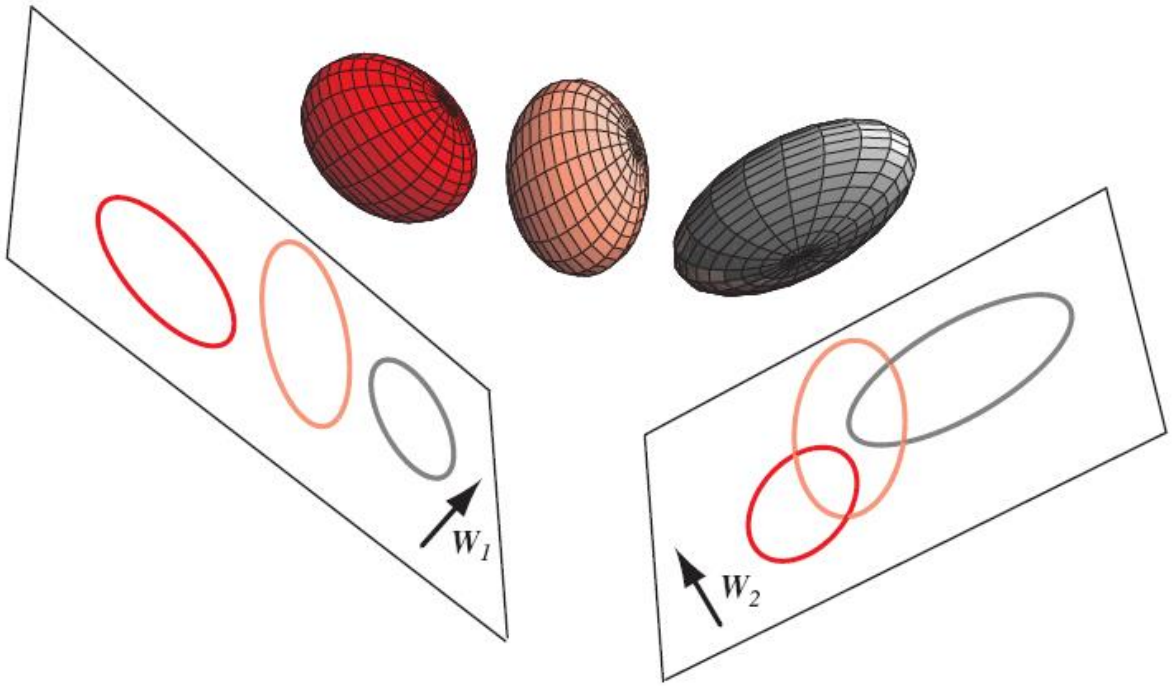
- **Important:**
  - Can apply LDR even when data are *not* normal

Examples

From 2D to 1D:



From 3D to 2D:



Fisher's Discriminant Analysis (FDA)

- FDA can be seen as an LDR method as well
- **Fisher's criterion** aims to maximize:

$$J(\mathbf{w}) = \frac{\mathbf{w}^t \mathbf{S}_E \mathbf{w}}{\mathbf{w}^t \mathbf{S}_W \mathbf{w}}$$

$$J_F(\mathbf{A}) = tr\left\{(\mathbf{A}\mathbf{S}_W\mathbf{A}^t)^{-1}(\mathbf{A}\mathbf{S}_E\mathbf{A}^t)\right\}$$

where $\mathbf{S}_E$ is the *between-class* scatter given by:

$$\mathbf{S}_E = \sum_{i=1}^{k} p_i(\mathbf{m}_i - \mathbf{m})(\mathbf{m}_i - \mathbf{m})^t$$

- with **m** being the *total mean vector* computed as:

$$\mathbf{m} = \sum_{i=1}^{c} p_i \mathbf{m}_i$$

- $\mathbf{S}_W$ is the *within-class* scatter matrix (as in the two-class case):

$$\mathbf{S}_W = \sum_{i=1}^{k} p_i \mathbf{S}_i$$

- The solution is given by the eigenvectors that correspond to the *largest* eigenvalues of:

$$\mathbf{S}_W^{-1}\mathbf{S}_E$$

Note:

- Found only if $\mathbf{S}_W^{-1}$ exists
- For *c* classes *at most c* – 1 eigenvalues are > 0, while the others are 0 and yield **no classification** at all

Heteroscedastic Discriminant Analysis (HDA)

- FDA is a homoscedastic criterion:
  - Maximizes the Mahalanobis distance between the means
  - and when the covariance matrices are equal
- HDA is heteroscedastic:
  - Takes correlations between random variables to project data onto lower dimensional spaces
  - Uses directed distance matrices
  - Starts from the Chernoff distance in original space
  - Also called the “Chernoff criterion” or “Loog-Duin (LD)”
  - More details in ref. [1]

Mathematical Formulation

- For two classes, maximize:

$$J_{LD_2}(\mathbf{A}) = tr\left\{(\mathbf{A}\mathbf{S}_W\mathbf{A}^t)^{-1}\left[\mathbf{A}\mathbf{S}_E\mathbf{A}^t - \mathbf{A}\mathbf{S}_W^{\frac{1}{2}}\frac{p_1\log(\mathbf{S}_W^{-\frac{1}{2}}\mathbf{S}_1\mathbf{S}_W^{-\frac{1}{2}}) + p_2\log(\mathbf{S}_W^{-\frac{1}{2}}\mathbf{S}_2\mathbf{S}_W^{-\frac{1}{2}})}{p_1p_2}\mathbf{S}_W^{\frac{1}{2}}\mathbf{A}^t\right]\right\}$$

  where log(**B**) = **F** log(**L**) **F**$^{-1}$, with **F** being the eigenvectors of **B** and **L** the eigenvalues.

- Solution given by the eigenvectors (whose eigenvalues are the *largest*) of:

$$\mathbf{S}_{LD_2} = \mathbf{S}_W^{-1}\left[\mathbf{S}_E - \mathbf{S}_W^{\frac{1}{2}}\frac{p_1\log(\mathbf{S}_W^{-\frac{1}{2}}\mathbf{S}_1\mathbf{S}_W^{-\frac{1}{2}}) + p_2\log(\mathbf{S}_W^{-\frac{1}{2}}\mathbf{S}_2\mathbf{S}_W^{-\frac{1}{2}})}{p_1p_2}\mathbf{S}_W^{\frac{1}{2}}\right]$$

Multi-class case

The heteroscedastic criterion (HDA):

$$J_{LD}(\mathbf{A}) = \sum_{i=1}^{k-1}\sum_{j=i+1}^{k} p_i p_j tr\left\{(\mathbf{A}\mathbf{S}_W\mathbf{A}^t)^{-1}\mathbf{A}\mathbf{S}_W^{\frac{1}{2}} \left[(\mathbf{S}_W^{-\frac{1}{2}}\mathbf{S}_{ij}\mathbf{S}_W^{-\frac{1}{2}})^{-\frac{1}{2}}\mathbf{S}_W^{-\frac{1}{2}}\mathbf{S}_{E_{ij}}\mathbf{S}_W^{-\frac{1}{2}}(\mathbf{S}_W^{-\frac{1}{2}}\mathbf{S}_{ij}\mathbf{S}_W^{-\frac{1}{2}})^{-\frac{1}{2}} + \frac{1}{\pi_i\pi_j}\left(\log(\mathbf{S}_W^{-\frac{1}{2}}\mathbf{S}_{ij}\mathbf{S}_W^{-\frac{1}{2}}) - \pi_i\log(\mathbf{S}_W^{-\frac{1}{2}}\mathbf{S}_i\mathbf{S}_W^{-\frac{1}{2}}) - \pi_j\log(\mathbf{S}_W^{-\frac{1}{2}}\mathbf{S}_j\mathbf{S}_W^{-\frac{1}{2}})\right)\right]\mathbf{S}_W^{\frac{1}{2}}\mathbf{A}^t\right\},$$

where:

$$\mathbf{S}_{E_{ij}} = (\mathbf{m}_i - \mathbf{m}_j)(\mathbf{m}_i - \mathbf{m}_j)^t,\ \pi_i = \frac{p_i}{p_i+p_j},\ \pi_j = \frac{p_j}{p_i+p_j},\ \text{and}\ \mathbf{S}_{ij} = \pi_i\mathbf{S}_i + \pi_j\mathbf{S}_j$$

$$\mathbf{S}_W = \sum_{i=1}^{k} p_i\mathbf{S}_i$$

- Solution given by the eigenvectors (whose eigenvalues are the *largest*) of:

$$\mathbf{S}_{LD} = \sum_{i=1}^{k-1} \sum_{j=i+1}^{k} p_i p_j \mathbf{S}_W^{-1} \mathbf{S}_W^{\frac{1}{2}} \Big[(\mathbf{S}_W^{-\frac{1}{2}} \mathbf{S}_{ij} \mathbf{S}_W^{-\frac{1}{2}})^{-\frac{1}{2}} \mathbf{S}_W^{-\frac{1}{2}} \mathbf{S}_{E_{ij}} \mathbf{S}_W^{-\frac{1}{2}} (\mathbf{S}_W^{-\frac{1}{2}} \mathbf{S}_{ij} \mathbf{S}_W^{-\frac{1}{2}})^{-\frac{1}{2}} + \frac{1}{\pi_i \pi_j} \Big(\log(\mathbf{S}_W^{-\frac{1}{2}} \mathbf{S}_{ij} \mathbf{S}_W^{-\frac{1}{2}}) - \pi_i \log(\mathbf{S}_W^{-\frac{1}{2}} \mathbf{S}_i \mathbf{S}_W^{-\frac{1}{2}}) - \pi_j \log(\mathbf{S}_W^{-\frac{1}{2}} \mathbf{S}_j \mathbf{S}_W^{-\frac{1}{2}}) \Big) \Big] \mathbf{S}_W^{\frac{1}{2}}$$

Chernoff Discriminant Analysis (CDA)

- HDA uses Chernoff distance in original space:
  - To create directed distance matrices
  - This does not, however, guarantee to maximize the Chernoff distance in transformed space
- CDA (unlike HDA) aims to maximize Chernoff distance in transformed space:
  - Chernoff distance is a good approximation of Bayes' classification error (for normal distributions)
  - Can be used even if distributions are not normal
  - Maximizing Chernoff distance in transformed space may not guarantee minimizing error
  - But works very well in many cases, including practical cases. See ref. [2]

Mathematical Formulation

Two-class criterion:

- Maximize:

$$J^*_{c_{12}}(\mathbf{A}) = p_1 p_2 (\mathbf{A}\mathbf{m}_1 - \mathbf{A}\mathbf{m}_2)^t [\mathbf{A}\mathbf{S}_W\mathbf{A}^t]^{-1} (\mathbf{A}\mathbf{m}_1 - \mathbf{A}\mathbf{m}_2) + \log\left(\frac{|\mathbf{A}\mathbf{S}_W\mathbf{A}^t|}{|\mathbf{A}\mathbf{S}_1\mathbf{A}|^{p_1}|\mathbf{A}\mathbf{S}_2\mathbf{A}|^{p_2}}\right)$$

- or equivalently:

$$J^*_{c_{12}}(\mathbf{A}) = tr\left\{p_1 p_2 (\mathbf{A}\mathbf{S}_W\mathbf{A}^t)^{-1}\mathbf{A}\mathbf{S}_E\mathbf{A}^t + \log(\mathbf{A}\mathbf{S}_W\mathbf{A}^t) - p_1\log(\mathbf{A}\mathbf{S}_1\mathbf{A}^t) - p_2\log(\mathbf{A}\mathbf{S}_2\mathbf{A}^t)\right\}$$

- The gradient:

$$\begin{aligned}\frac{\partial J^*_{c_{12}}}{\partial \mathbf{A}} = {} & 2p_1p_2\left[\mathbf{S}_E\mathbf{A}^t(\mathbf{A}\mathbf{S}_W\mathbf{A}^t)^{-1} - \mathbf{S}_W\mathbf{A}^t(\mathbf{A}\mathbf{S}_W\mathbf{A}^t)^{-1}(\mathbf{A}\mathbf{S}_E\mathbf{A}^t)(\mathbf{A}\mathbf{S}_W\mathbf{A}^t)^{-1}\right]^t \\ & +2\left[\mathbf{S}_W\mathbf{A}^t(\mathbf{A}\mathbf{S}_W\mathbf{A}^t)^{-1} - p_1\mathbf{S}_1\mathbf{A}^t(\mathbf{A}\mathbf{S}_1\mathbf{A}^t)^{-1} - p_2\mathbf{S}_2\mathbf{A}^t(\mathbf{A}\mathbf{S}_2\mathbf{A}^t)^{-1}\right]^t\end{aligned}$$

- The criterion is maximized via the gradient algorithm.

Algorithm **Chernoff_LDA_Two**

Input: Threshold τ

begin

$\mathbf{A}^{(0)} \leftarrow \max_{\mathbf{A}}\{J^*_{c_{12}}(\mathbf{A}_F), J^*_{c_{12}}(\mathbf{A}_{LD})\}$ // Max. of Fisher's and

$k \leftarrow 0$

repeat

$\eta_k \leftarrow max_{\eta>0}\phi_{k_{12}}(\eta)$

$\mathbf{B} \leftarrow \mathbf{A}^{(k)} + \eta_k \nabla J^*_{c_{12}}(\mathbf{A}^{(k)})$

Decompose $\mathbf{B}$ into $\mathbf{R}$ and $\mathbf{Q}$

$\mathbf{A}^{(k+1)} \leftarrow \mathbf{Q}$

$k \leftarrow k + 1$

until $|J^*_{c_{12}}(\mathbf{A}^{(k-1)}) - J^*_{c_{12}}(\mathbf{A}^{(k)})| < \tau$

return $\mathbf{A}^{(k)}$, $J^*_{c_{12}}(\mathbf{A}^{(k)})$

end

Multi-class case

- Maximize:

$$J_c^*(\mathbf{A}) = \sum_{i=1}^{k-1} \sum_{j=i+1}^{k} J_{c_{ij}}^*(\mathbf{A})$$

where:

$$J_{c_{ij}}^*(\mathbf{A}) = tr\left\{p_i p_j (\mathbf{A}\mathbf{S}_{W_{ij}}\mathbf{A}^t)^{-1}\mathbf{A}\mathbf{S}_{E_{ij}}\mathbf{A}^t + \log(\mathbf{A}\mathbf{S}_{W_{ij}}\mathbf{A}^t) - p_i \log(\mathbf{A}\mathbf{S}_i\mathbf{A}^t) - p_j \log(\mathbf{A}\mathbf{S}_j\mathbf{A}^t)\right\}$$

- Gradient:

$$\nabla J_c^*(\mathbf{A}) = \frac{\partial J_c^*(\mathbf{A})}{\partial \mathbf{A}} = \frac{\partial}{\partial \mathbf{A}} \sum_{i=1}^{k-1} \sum_{j=i+1}^{k} J_{c_{ij}}^*(\mathbf{A}) = \sum_{i=1}^{k-1} \sum_{j=i+1}^{k} \nabla J_{c_{ij}}^*(\mathbf{A})$$

$$\nabla J_{c_{ij}}^*(\mathbf{A}) = \frac{\partial J_{c_{ij}}^*}{\partial \mathbf{A}} = 2p_i p_j \left[\mathbf{S}_{E_{ij}}\mathbf{A}^t(\mathbf{A}\mathbf{S}_{W_{ij}}\mathbf{A}^t)^{-1} - \mathbf{S}_{W_{ij}}\mathbf{A}^t(\mathbf{A}\mathbf{S}_{W_{ij}}\mathbf{A}^t)^{-1}(\mathbf{A}\mathbf{S}_{E_{ij}}\mathbf{A}^t)(\mathbf{A}\mathbf{S}_{W_{ij}}\mathbf{A}^t)^{-1}\right]^t$$
$$+2\left[\mathbf{S}_{W_{ij}}\mathbf{A}^t(\mathbf{A}\mathbf{S}_{W_{ij}}\mathbf{A}^t)^{-1} - p_i\mathbf{S}_i\mathbf{A}^t(\mathbf{A}\mathbf{S}_i\mathbf{A}^t)^{-1} - p_j\mathbf{S}_j\mathbf{A}^t(\mathbf{A}\mathbf{S}_j\mathbf{A}^t)^{-1}\right]^t$$

Algorithm (multi-class)

Algorithm **Chernoff_LDA_Multi**

Input: Threshold τ

begin

$\mathbf{A}^{(0)} \leftarrow \max_{\mathbf{A}}\{J^*_{c12}(\mathbf{A}_F), J^*_{c12}(\mathbf{A}_{LD})\}$ // Max. of Fisher's and Loog-Duin

$k \leftarrow 0$

repeat

$\eta_k \leftarrow max_{\eta>0}\phi_k(\eta)$

$\mathbf{B} \leftarrow \mathbf{A}^{(k)} + \eta_k \sum_{i=1}^{k-1}\sum_{j=i+1}^{k} \nabla J^*_{c_{ij}}(\mathbf{A}^{(k)})$

Decompose $\mathbf{B}$ into $\mathbf{R}$ and $\mathbf{Q}$

$\mathbf{A}^{(k+1)} \leftarrow \mathbf{Q}$

$k \leftarrow k+1$

until $|J^*_{c_{ij}}(\mathbf{A}^{(k-1)}) - J^*_{c_{ij}}(\mathbf{A}^{(k)})| < \tau$

return $\mathbf{A}^{(k)}, \sum_{i=1}^{k-1}\sum_{j=i+1}^{k} J^*_{c_{ij}}(\mathbf{A}^{(k)})$

end

Dealing with Singular Matrices

- Find a function of A as follows $f(\mathbf{A}) = \mathbf{\Phi}\, f(\mathbf{\Lambda})\, \mathbf{\Phi}^{-1}$, where

$$f(\mathbf{\Lambda}) = \begin{bmatrix} f(\lambda_1) & 0 & 0 & 0 & \cdots & 0 \\ 0 & \ddots & 0 & \vdots & \cdots & \vdots \\ 0 & 0 & f(\lambda_k) & 0 & \cdots & 0 \\ 0 & 0 & 0 & \mathbf{0} & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & \cdots & \mathbf{0} \end{bmatrix}$$

Make zero all these values in the diagonal

- 0's in bold replace singular values of $f(\lambda_{k+1}), \ldots, f(\lambda_d)$
- For example, log(0), 0^{-1}, $0^{-1/2}$, etc.
- This will let us compute $\mathbf{S}_W^{-1}$, log($\mathbf{S}_W$), etc.
- It's similar to SVD

Example 1:

- Given: $\mathbf{x}_1 \sim N(\mathbf{m}_1, \mathbf{S}_1)$ and $\mathbf{x}_2 \sim N(\mathbf{m}_2, \mathbf{S}_2)$

$$\mathbf{m}_1 = [0.5001, 0.4947]^t, \quad \mathbf{m}_2 = [2.1069, 1.4324]^t$$

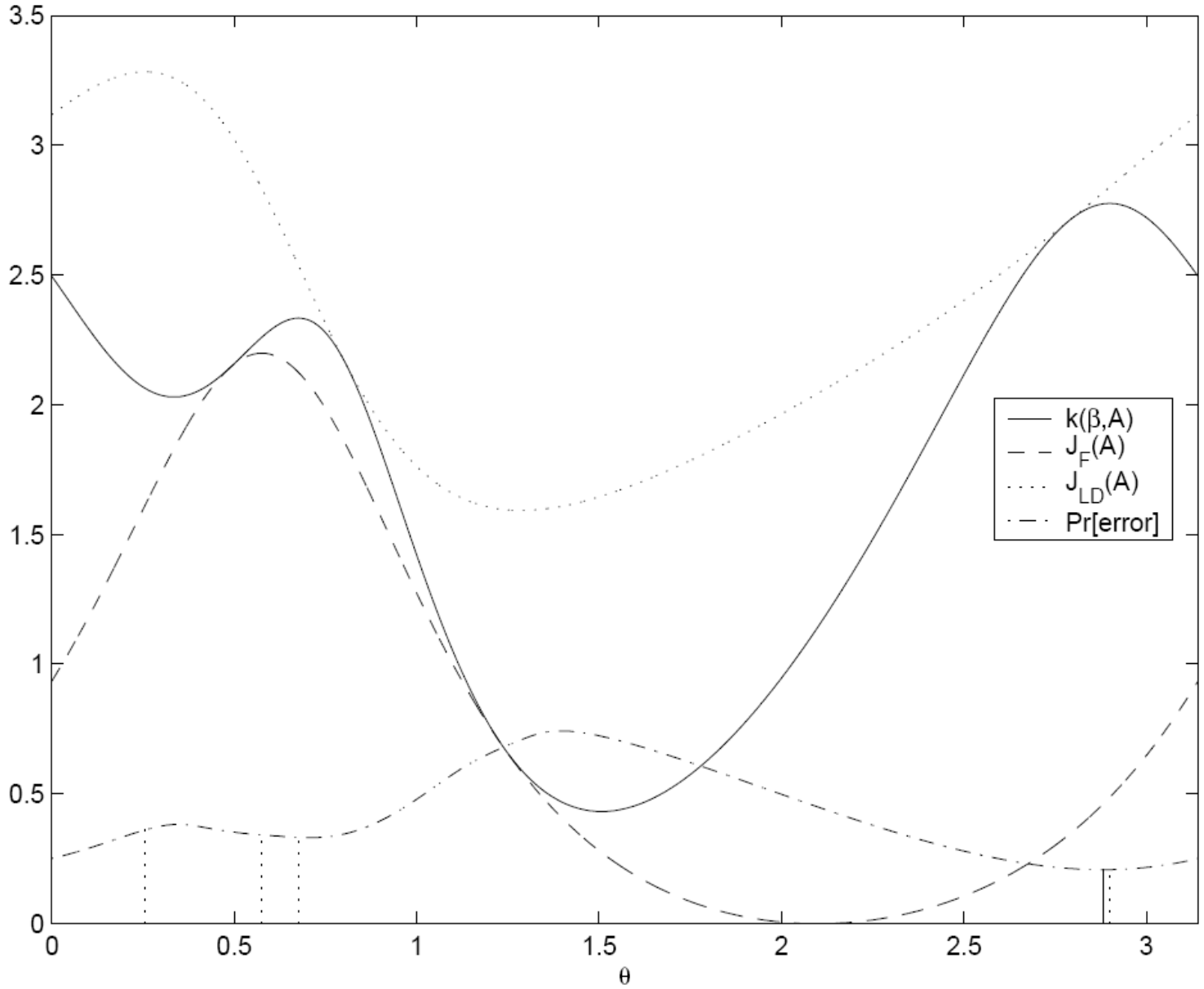
$$\mathbf{S}_1 = [0.8205, 0.4177; 0.4177, 2.8910]$$

$$\mathbf{S}_2 = [5.1150, -4.3990; -4.3990, 5.7119]$$

$$p_1 = p_2 = 0.5$$

- Linear transformation: **y** = **A x**
  where **x** is 2D, **y** is 1D, and **A** is 1×2.
- Let $\beta = 1/2$
- Plot J_F, J_{LD}, and J_c^* or $k(\beta,\mathbf{A})$ for all values of θ
  where θ is the angle between **A** and $[1,0]^t$

Plots of Criterion Functions



Example 2:

Given: $\mathbf{x}_1 \sim N(\mathbf{m}_1, \mathbf{S}_1)$ and $\mathbf{x}_2 \sim N(\mathbf{m}_2, \mathbf{S}_2)$

$$\mathbf{m}_1 = [0.6083 \quad 2.2414]^t, \quad \mathbf{m}_2 = [3.5014 \quad 6.3859]^t$$

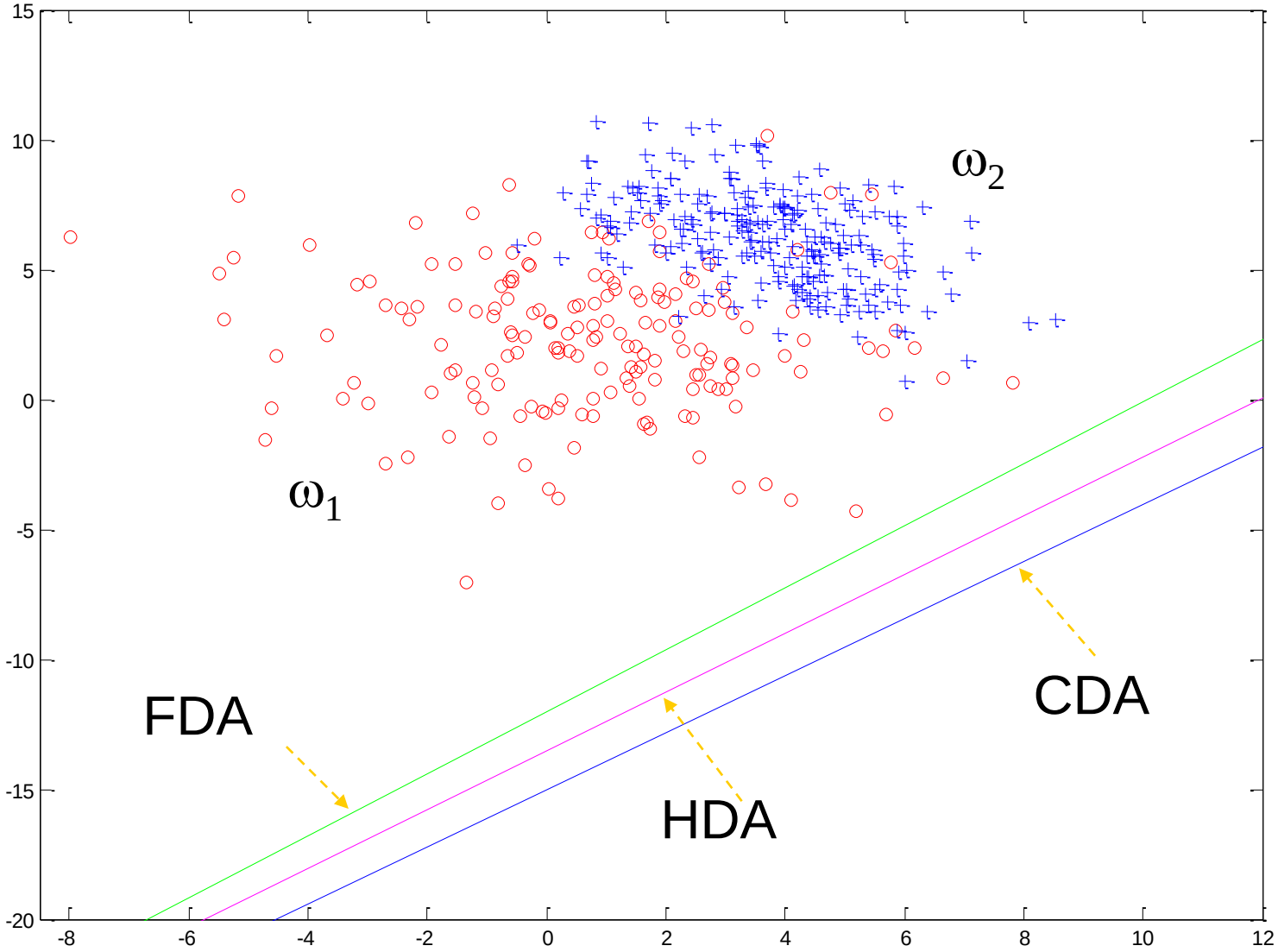
$$\mathbf{S}_1 = \begin{bmatrix} 6.9311 & -0.4091 \\ -0.4091 & 6.7122 \end{bmatrix}, \quad \mathbf{S}_2 = \begin{bmatrix} 2.7743 & -1.5834 \\ -1.5834 & 3.4662 \end{bmatrix}$$

$p_1 = 0.4358, \quad p_2 = 0.5642$

	Chernoff Dist.	**A**
Fisher	7.7708	[0.6431 0.7658]
Loog-Duin	7.7880	[0.6620 0.7495]
Gradient*	7.7907	[0.6731 0.7397]

***** Optimal for this example.

Graphically



- Difference almost not noticeable in 2D → 1D
- But significant in larger dimensions:
  e.g., 50D → 10D

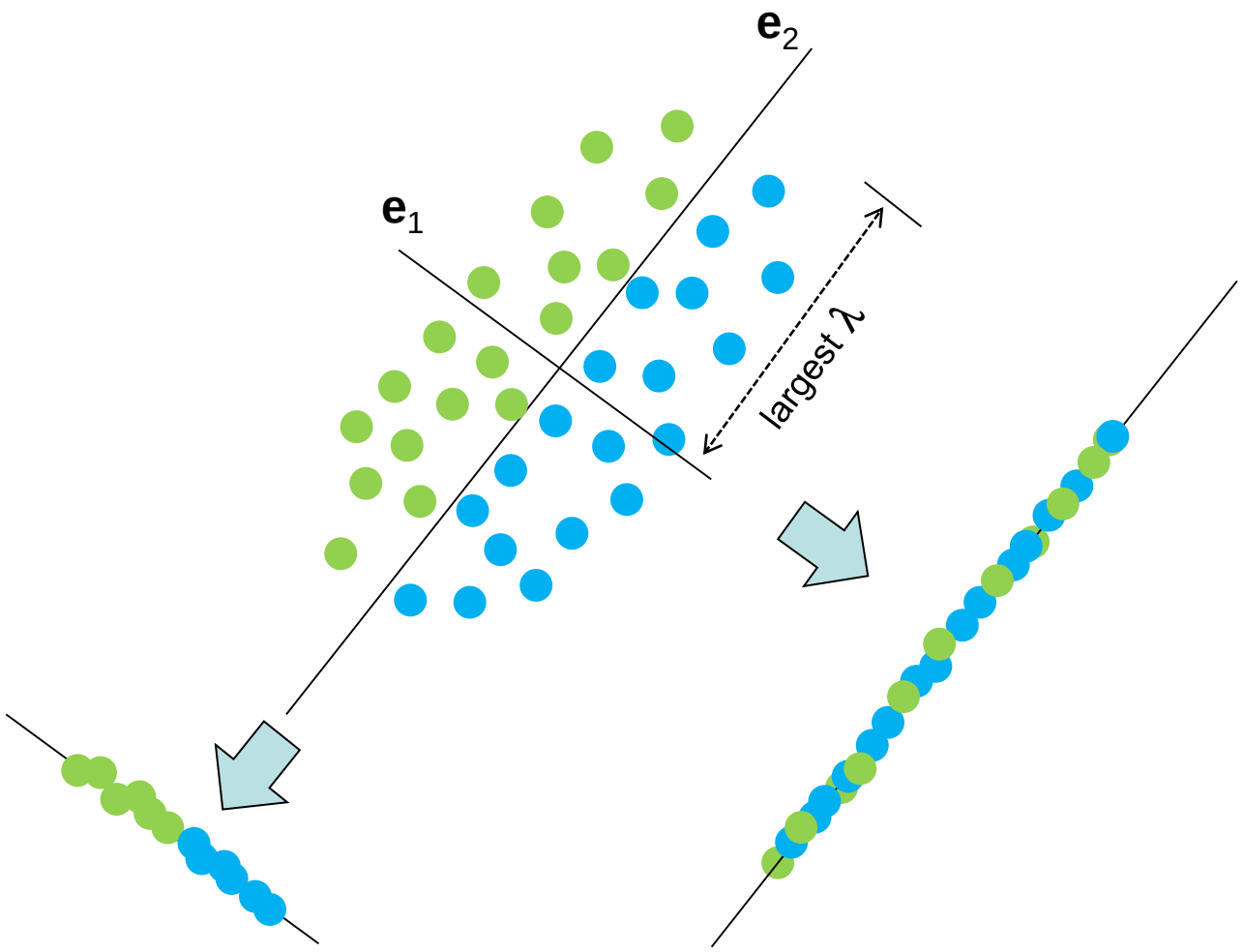
PCA vs LDA

PCA considers *all* classes *together* as a source of information about how *disperse* the data points are

- Better applicable to:
  - *Unsupervised* learning problems (classes not known)
  - When no LDA method can be applied (due to ill-conditioned matrices, for example)

LDA takes information about individual classes into account to maximize separability in new space

- Better applicable to:
  - *Supervised* classification (classes are *known*)
  - Lower dimension still gives good (desired) separability and sometimes even better!



Dimensionality Reduction in Scikit

- The sklearn.decomposition module includes matrix decomposition algorithms, including among others PCA, NMF or ICA.
- It also includes:
  - kernel PCA
  - SVD
  - Manifold learning
    - https://scikit-learn.org/stable/modules/manifold.html#manifold
- Main link:
  - http://scikit-learn.org/stable/modules/classes.html#module-sklearn.decomposition
- LDA is included in sklearn.discriminant_analysis:
  - http://scikit-learn.org/stable/modules/classes.html#module-sklearn.discriminant_analysis

Sample Dataset Generation in Scikit

- The sklearn.datasets module includes utilities for generating and loading datasets.
- Typical datasets:
  - Synthetic: S-shape, Swiss roll, Circles, Blobs, Moons, etc.
  - Real datasets: Iris, Breast cancer, Faces, News groups, etc.

- Main link:
  - https://scikit-learn.org/stable/modules/classes.html#module-sklearn.datasets

References - Acknowledgments

The material presented in this chapter has been taken from the following sources (among others):

1. Loog M, Duin RPW. Linear Dimensionality Reduction via a Heteroscedastic Extension of LDA: The Chernoff Criterion. IEEE Transactions on Pattern Analysis and Machine Intelligence. 2004;PAMI:732-739.
2. L. Rueda and M. Herrera, “Linear Dimensionality Reduction by Maximizing the Chernoff Distance in the Transformed Space”, Pattern Recognition, Vol. 41, Issue 10, 2008, pp. 3138-3152.
3. C. Bishop, “Pattern Recognition and Machine Learning”, Springer, 2006.
4. Y. Liu et al, “Regularized Non-negative Matrix Factorization for Identifying Differentially Expressed Genes and Clustering Samples: a Survey”, IEEE TCBB, Vol 15, Issue 3, pp. 974-987, 2017. https://ieeexplore.ieee.org/document/7845582/
5. J. Brunet et al., “Metagenes and molecular pattern discovery using matrix factorization”, PNAS, 2004, 101(12):4164-9
6. M. Mohri et al. Foundations of Machine Learning. Second Edition, 2018
7. D. Lubo-Robles. Development of Independent Component Analysis for Reservoir Geomorphology and Unsupervised Seismic Facies Classification in the Taranaki Basin, New Zealand. MSc Thesis, University of Oklahoma, 2018.
8. S. Lawence et al. Spectral Methods for Dimensionality Reduction. In “Semi-Supervised Learning” by O. Chapelle et al., MIT Press, 2013.
9. Mikolov et al. Distributed Representations of Words and Phrases and their Compositionality. NIPS 2013.
10. W. Hamilton et al. Representation Learning on Graphs: Methods and Applications. 2018. https://arxiv.org/abs/1709.05584
11. Bramley, T. (2014). Multivariate representations of subject difficulty. Research Matters: A Cambridge Assessment Publication, 18, 42-47
12. T. Hastie et al. The Elements of Statistical Learning. Second Edition. Springer, 2008.
13. C. Aggarwal. Neural Networks and Deep Learning, Springer, 2018.
14. S. Skansi. Introduction to Deep Learning. Springer, 2018.