MECS 6616

Dimensionality Reduction

Spring 2020
Matei Ciocarlie

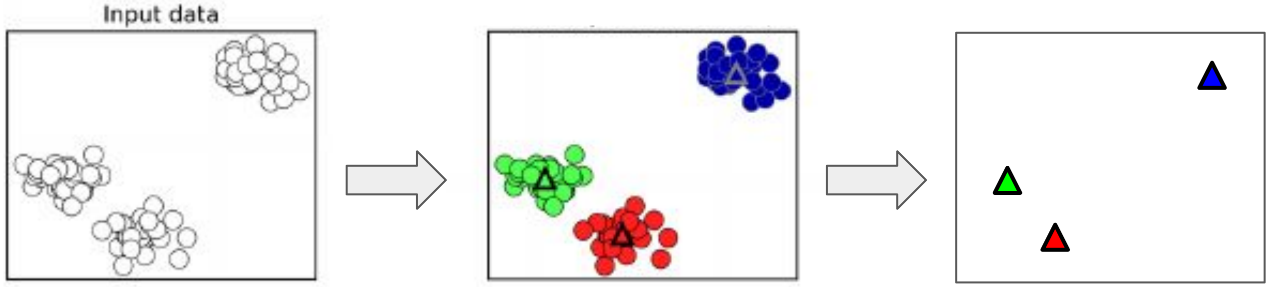
Primary sources: [Muller and Guido, Introduction to Machine Learning with Python]
[Hastie et al., The Elements of Statistical Learning]

Unsupervised Learning

- Finding structure in data
    - "Point cloud" of high-dimensional vectors $\mathbf{x}^T = [x_1, x_2, \ldots, x_d] \subset \Re^d$
    - Is there some hidden structure that I can use to simplify?

Unsupervised Learning

- Previous lecture: clustering
  - Do my points mostly “cluster” together?
  - Assume K clusters, labeled by integer $k \subset \{1, \ldots, K\}$
  - Cluster assignment: $C(i) = k$. If this fits data well, we get:
    - small within-cluster scatter
    - large between-cluster scatter
  - Simpler structure: discrete number of clusters
    - Replace each point with cluster representative: $\mathbf{x}_i \rightarrow \mathbf{r}_k$ where $k \subset \{1, \ldots, K\}, C(i) = k$



Unsupervised Learning

- Finding structure in data
    - "Point cloud" of N high-dimensional vectors $\mathbf{x} = [x_1, x_2, \ldots, x_d] \subset \Re^d$
    - Is there some hidden structure that I can use to simplify?
- Previous lecture: clustering (discrete sub-structure)
- Is there some continuous sub-structure?

Dimensionality Reduction

- Finding structure in data
    - "Point cloud" of N high-dimensional vectors $\mathbf{x}_i = [x_1, x_2, \ldots, x_d] \subset \Re^d$
    - Is there some hidden structure that I can use to simplify?
- Previous lecture: clustering (discrete sub-structure)
- Is there some continuous sub-structure?

$$\mathbf{x}_i \rightarrow f(\mathbf{y}_i) \text{ where } \mathbf{y}_i \subset \Re^m, \mathrm{m} << \mathrm{d}$$

Dimensionality Reduction

- Finding structure in data
  - "Point cloud" of N high-dimensional vectors $\mathbf{x}_i = [x_1, x_2, \ldots, x_d] \subset \Re^d$
  - Is there some hidden structure that I can use to simplify?
- Previous lecture: clustering (discrete sub-structure)
- Is there some continuous sub-structure?

$$\mathbf{x}_i \rightarrow f(\mathbf{y}_i) \text{ where } \mathbf{y}_i \subset \Re^m, \mathrm{m} << \mathrm{d}$$

- What is a good low-dimensional approximation (projection)?
  - reduce $\sum \mathrm{d}(\mathbf{x}_i , f(\mathbf{y}_i))$
  - with Euclidean distance: reduce $\sum \|\mathbf{x}_i - f(\mathbf{y}_i)\|^2$
  - get rid of dimensions without losing information

Linear Dimensionality Reduction

$\mathbf{x}_i \rightarrow f(\mathbf{y}_i)$ where $\mathbf{y}_i \subset \Re^m$, $m << d$

- Function f is linear projection:

$\mathbf{x}_i \rightarrow \mathbf{A}\mathbf{y}_i$, $\mathbf{A} \subset \Re^{d\times m}$

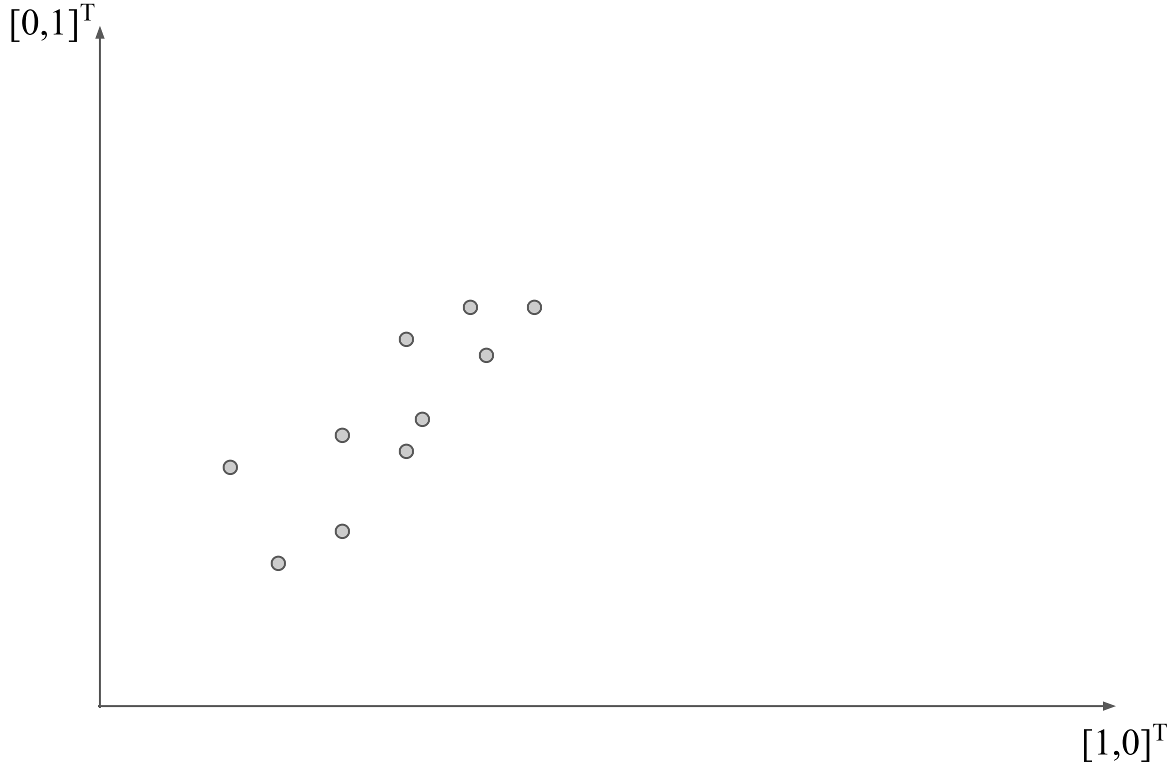
- Without loss of generality, we can focus on orthonormal projection matrices:

$\mathbf{A}^T\mathbf{A} = \mathbf{I}^{m\times m}$ (careful! $\mathbf{A}\mathbf{A}^T \neq \mathbf{I}^{d\times d}$)

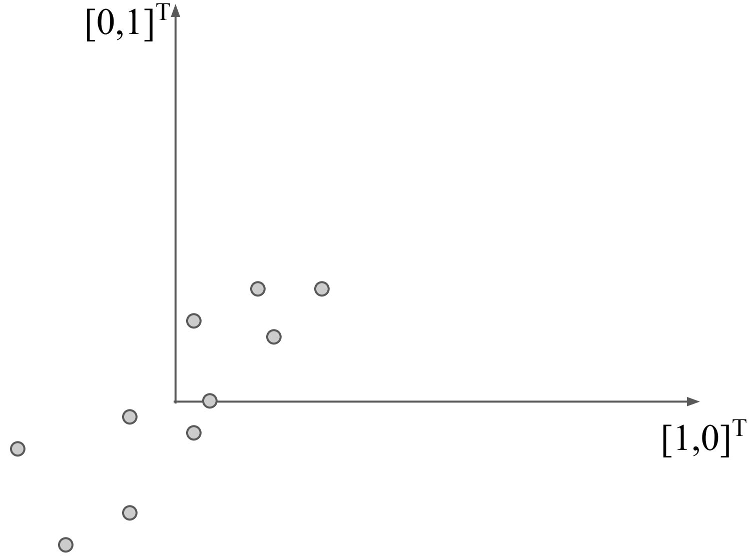
$\mathbf{y}_i = \mathbf{A}^T\mathbf{x}_i$

- The columns of $\mathbf{A}$ (rows of $\mathbf{A}^T$) are the basis vectors of the subspace

Linear Dimensionality Reduction

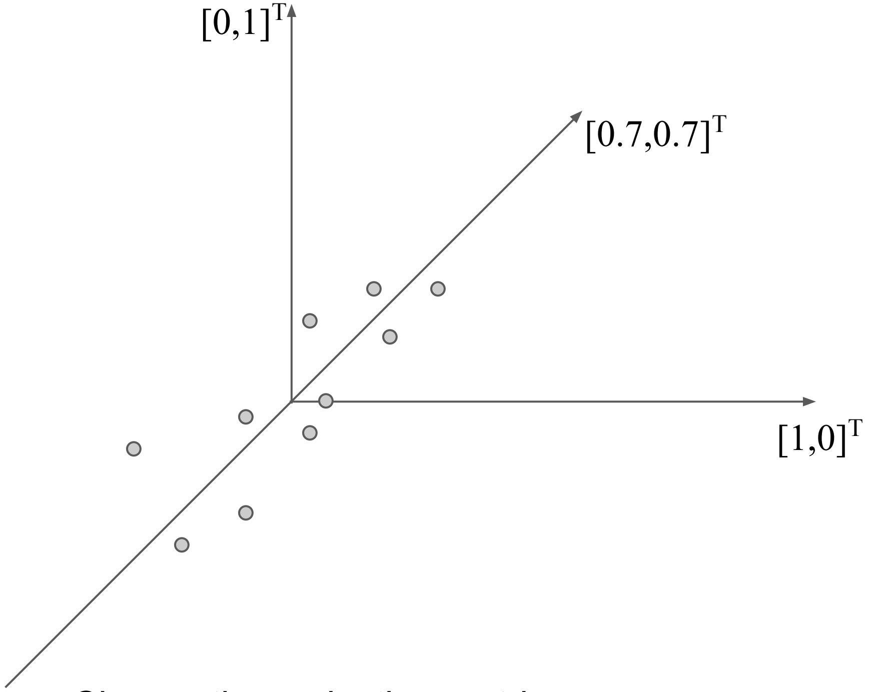


Linear Dimensionality Reduction



Center the data: $\mathbf{x}_i \leftarrow \mathbf{x}_i - \mathbf{x}_{mean}$

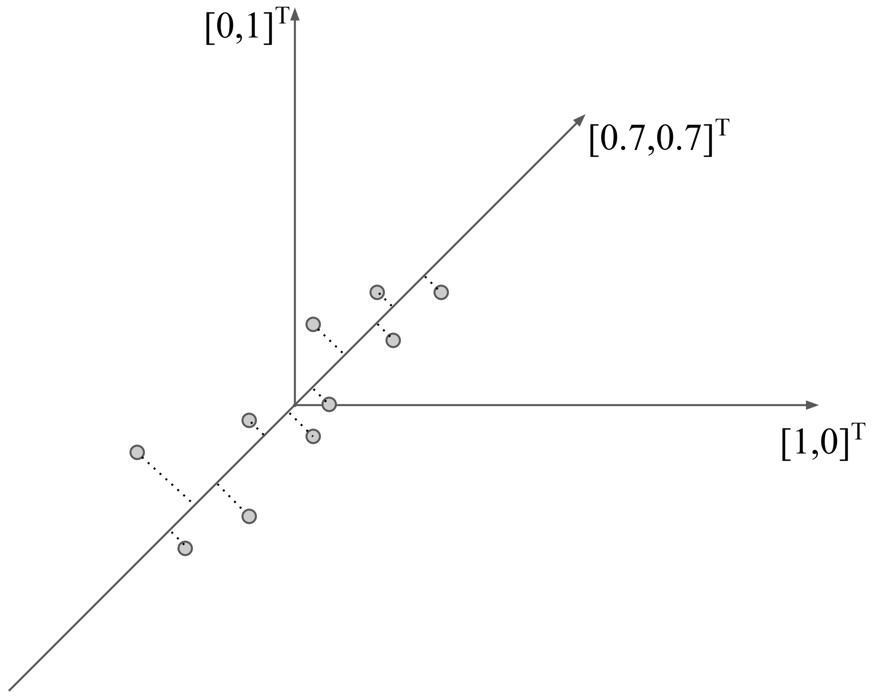
Linear Dimensionality Reduction



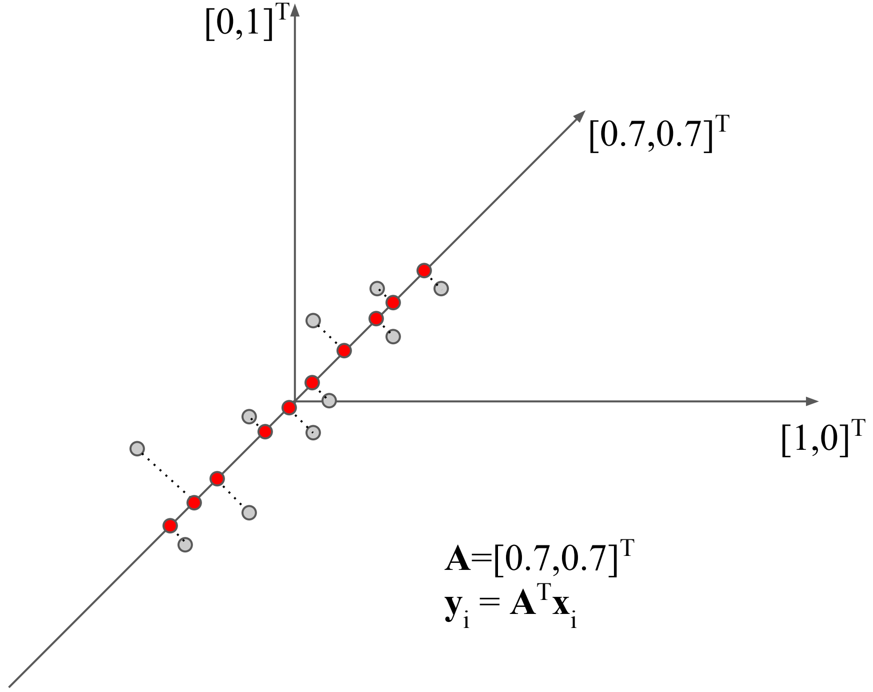
Choose the projection matrix $\mathbf{A}$

Here, $\mathbf{A}=[0.7,0.7]^T$

Linear Dimensionality Reduction



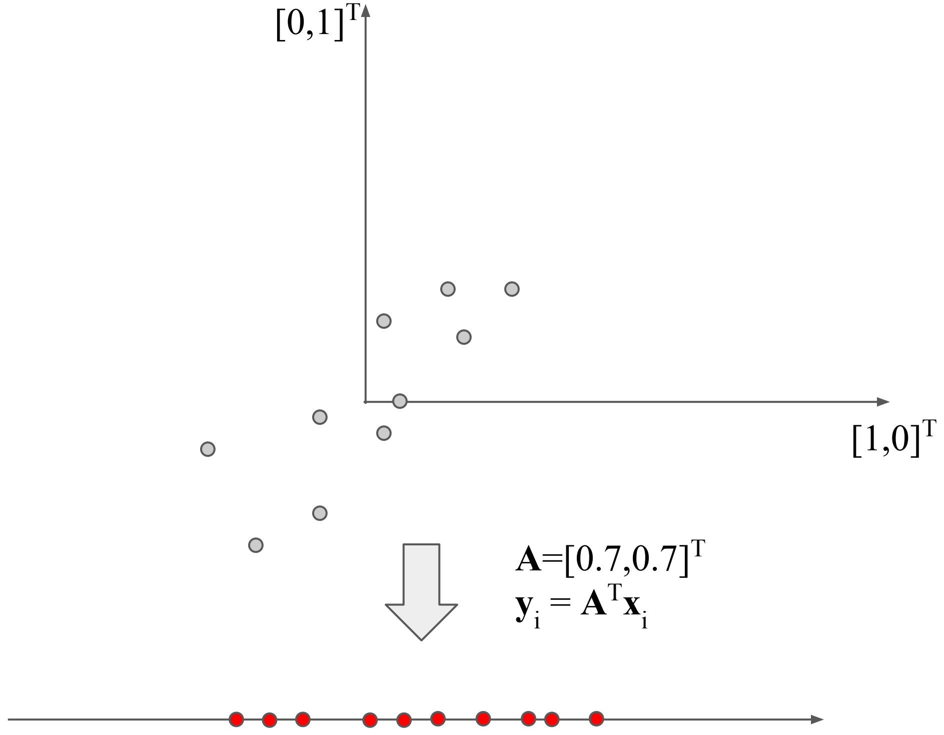
Linear Dimensionality Reduction



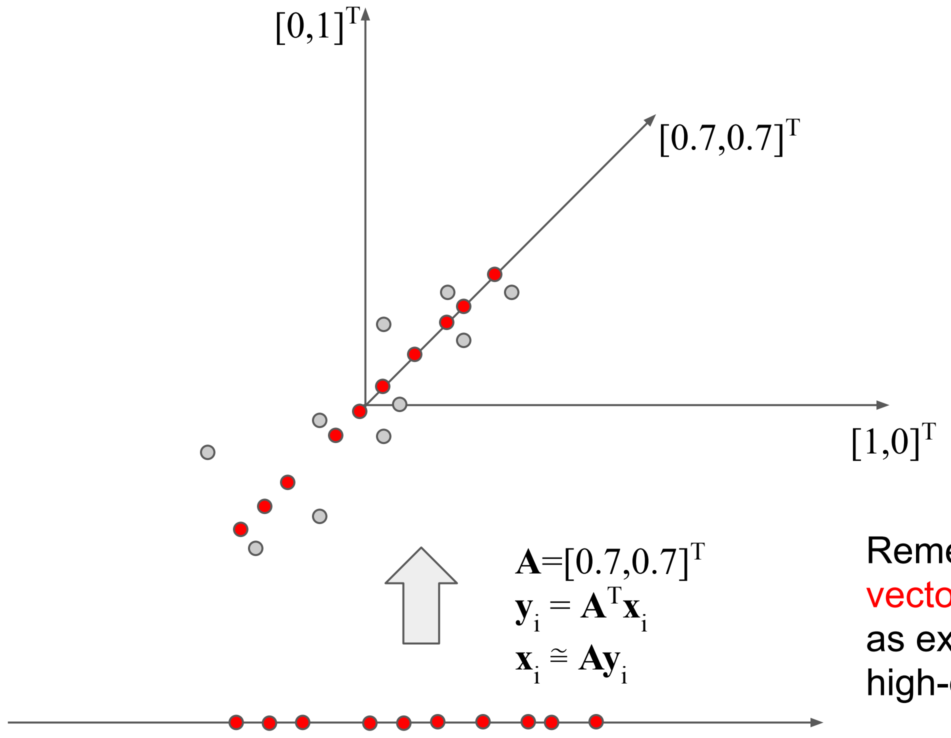
$\mathbf{A}=[0.7,0.7]^T$

$\mathbf{y}_i = \mathbf{A}^T\mathbf{x}_i$

Linear Dimensionality Reduction



Linear Dimensionality Reduction



$\mathbf{A}=[0.7,0.7]^T$

$\mathbf{y}_i = \mathbf{A}^T\mathbf{x}_i$

$\mathbf{x}_i \cong \mathbf{A}\mathbf{y}_i$

Remember: the matrix A contains the basis vectors of your low-dimensional subspace, as expressed in the original, high-dimensional space.

Linear Dimensionality Reduction

$$\mathbf{x}_i \rightarrow \mathbf{A}\mathbf{y}_i,\ \mathbf{A} \subset \Re^{d\times m},\ \mathbf{A}^T\mathbf{A} = \mathbf{I}$$

- When using Euclidean distance, this is a well-known Least Squares problem:

  given d, find $\mathbf{A} \subset \Re^{d\times m}$ to minimize $\sum \|\mathbf{x}_i - \mathbf{A}\mathbf{A}^T\mathbf{x}_i\|^2 = \sum \|(\mathbf{I} - \mathbf{A}\mathbf{A}^T)\mathbf{x}_i\|^2$

- Same as maximizing the variance of the projection:

  given d, find $\mathbf{A} \subset \Re^{d\times m}$ to maximize $\sum \|\mathbf{A}^T\mathbf{x}_i\|^2$

Principal Component Analysis

- Center the data by removing mean from each point
- Assemble the data matrix $\mathbf{X}$, where each row is a (transposed) data point $\mathbf{x}_i$
- The best k directions on which to project the data onto a k-dimensional subspace are given by the first k eigenvectors of the covariance matrix $\mathbf{X}^T\mathbf{X}$

$$\mathbf{X}^T\mathbf{X} = \mathbf{Q}\,\boldsymbol{\Lambda}\,\mathbf{Q}^T$$

- The eigenvalue λ_k corresponding to each eigenvector $\mathbf{q}_k$ tells us what percent of the variance in the data is captured by that respective direction
- The best matrix $\mathbf{A}$ for k dimensions: first k columns of $\mathbf{Q}$

Principal Component Analysis

- Center the data by removing mean from each point
- Assemble the data matrix $\mathbf{X}$, where each row is a (transposed) data point $\mathbf{x}_i$
- Alternative (more robust) computation: SVD on data matrix $\mathbf{X}$

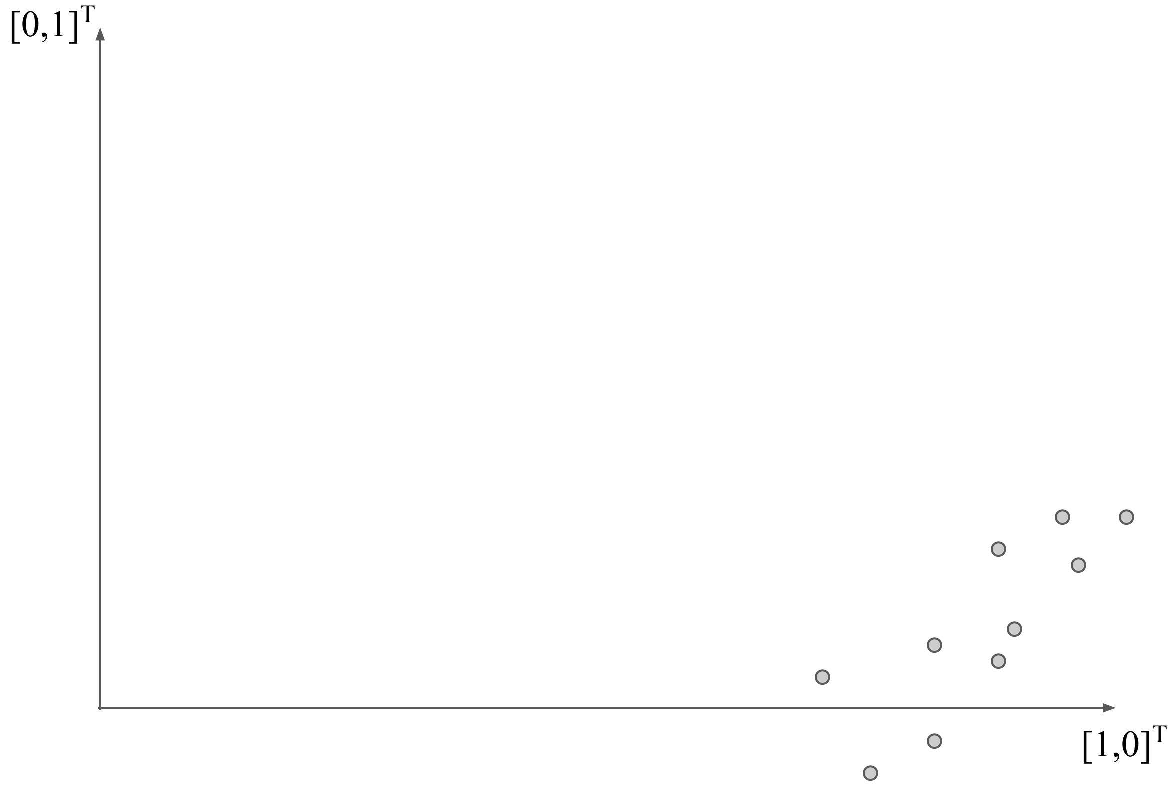
$$\mathbf{X} = \mathbf{U}\ \mathbf{S}\ \mathbf{V}^T$$

- First k columns of $\mathbf{V}$ are the principal directions
- Singular values and eigenvalues are related: $\lambda_i = s^2_i \,/\, (n-1)$

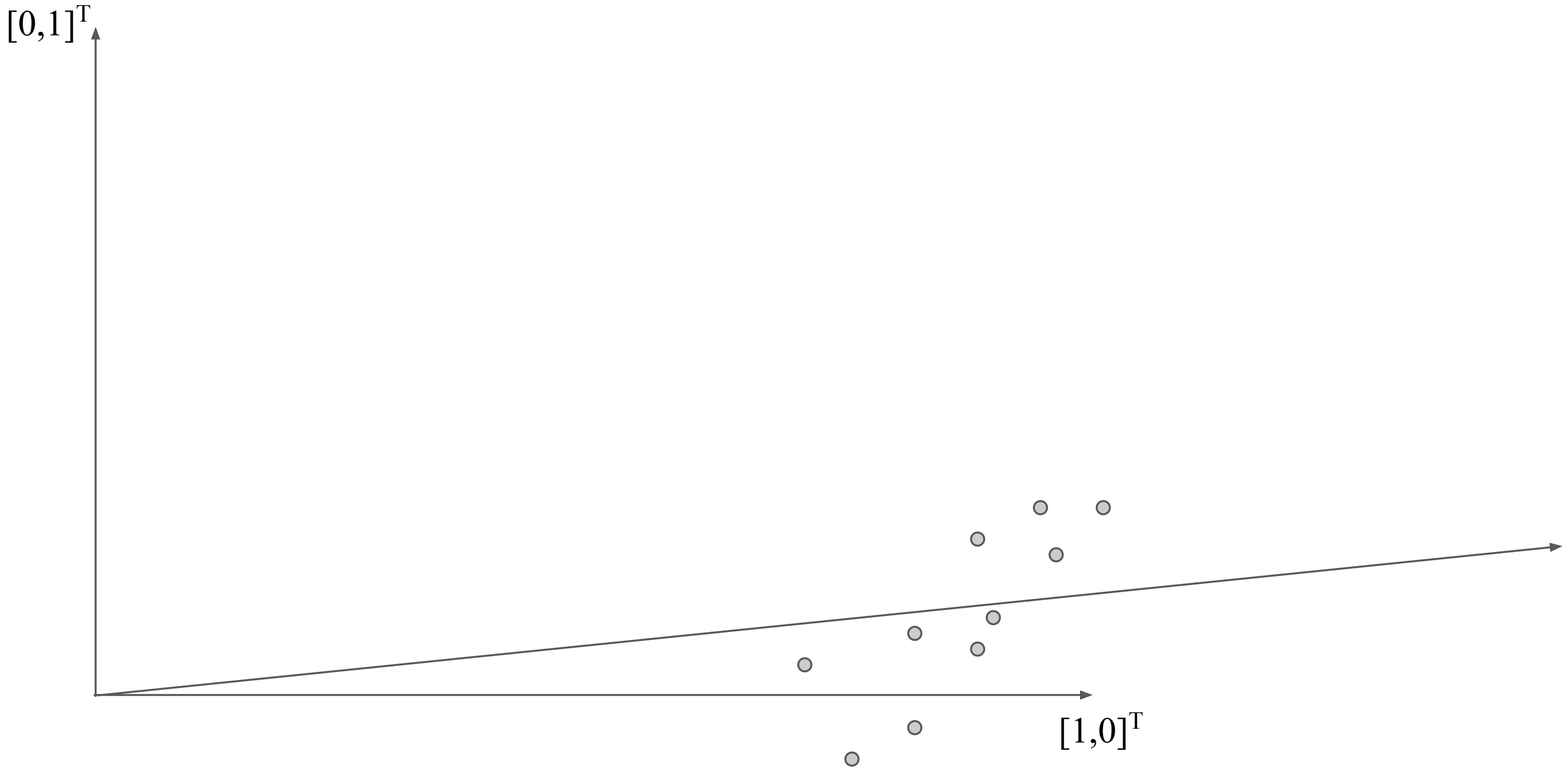
Principal Component Analysis

- Finds the best k-dimensional linear subspace for your original data
  - Basis vectors are directions along which your data has largest variance
  - Maximizes variance of projection
  - Minimizes re-projection error
- Tricky issues:
  - Remember to center your data
  - Depending on the problem, it might help to scale each dimension as well
  - Over-representation will skew your results
  - Keep in mind that principal directions (eigenvectors) are directions, not points

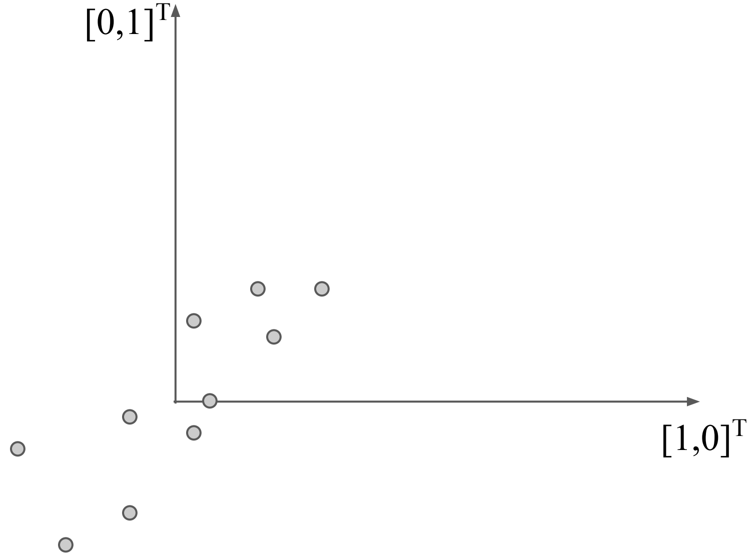
Center the Data!



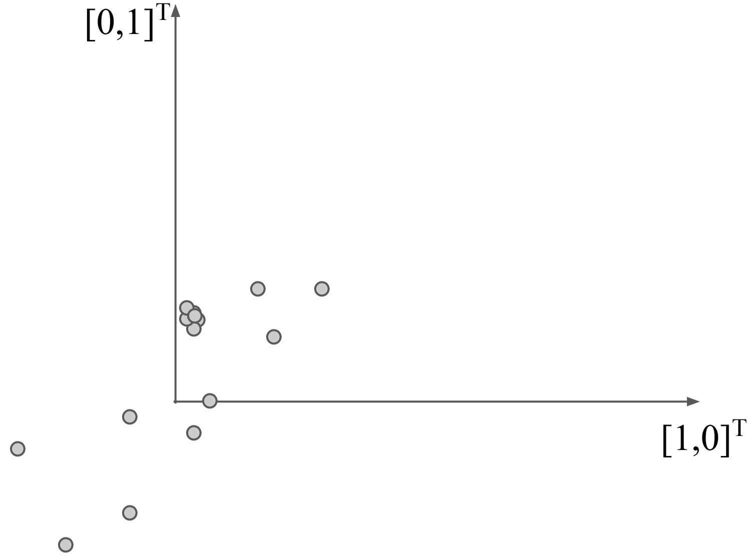
Center the Data!



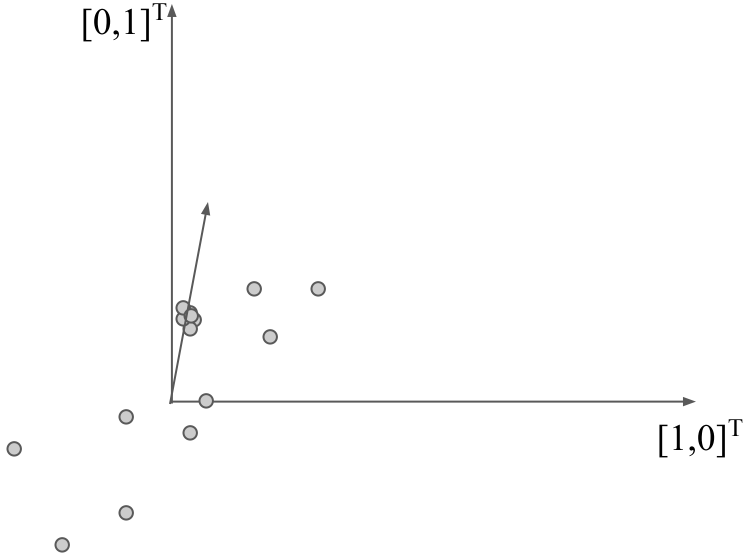
Watch for Over-representation!



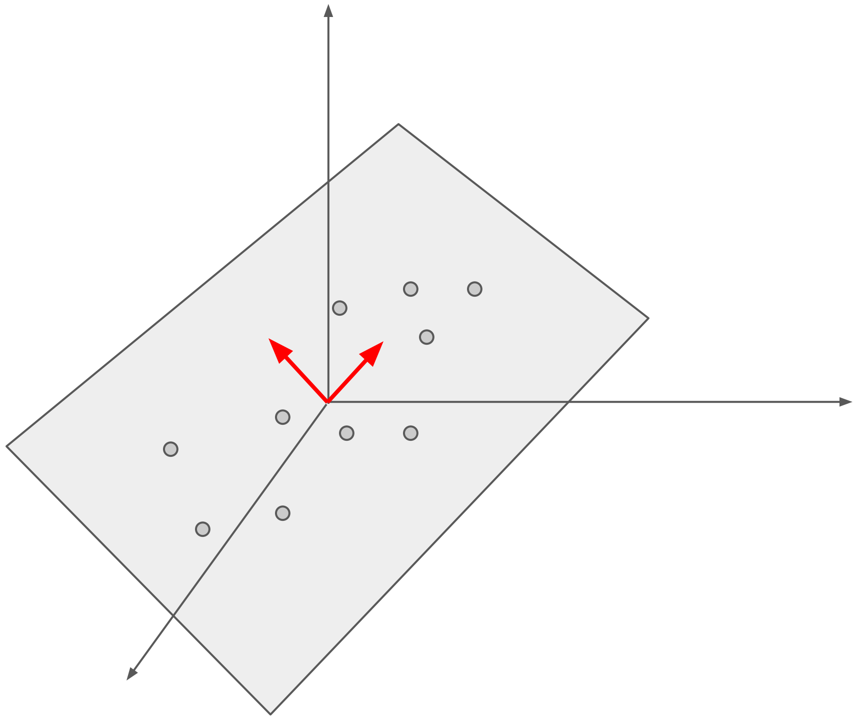
Watch for Over-representation!



Watch for Over-representation!



In Practice: More Dimensions...



Santello et al., Postural Synergies for Tool Use

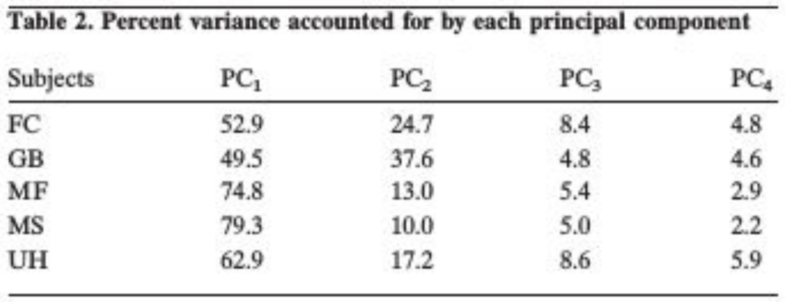
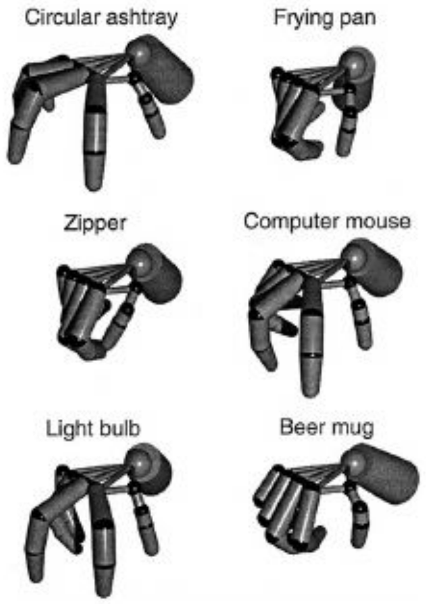
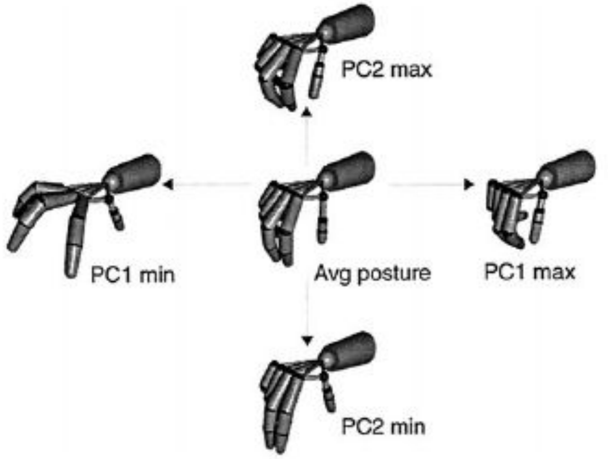
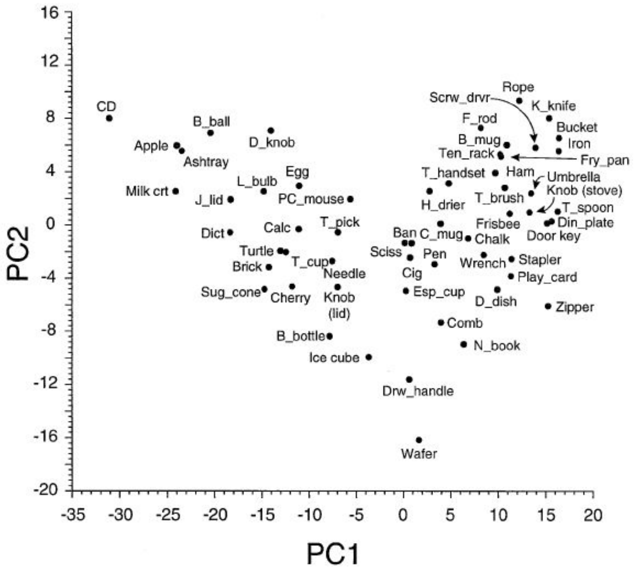


Table 2. Percent variance accounted for by each principal component

Subjects	PC_1	PC_2	PC_3	PC_4
FC	52.9	24.7	8.4	4.8
GB	49.5	37.6	4.8	4.6
MF	74.8	13.0	5.4	2.9
MS	79.3	10.0	5.0	2.2
UH	62.9	17.2	8.6	5.9

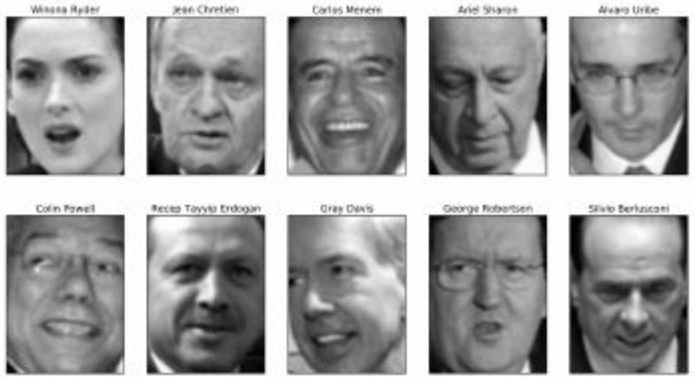






N = 57 grasps X 5 subjects
d = 14 joint angles per grasp

Eigenfaces



N = 3,023
d = 87 x 65 pixels = 5,655

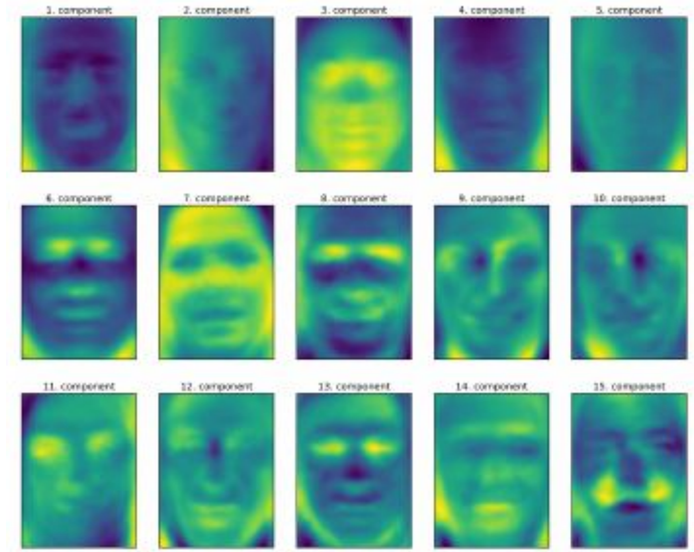
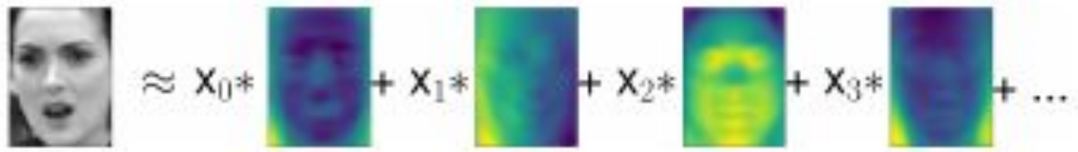
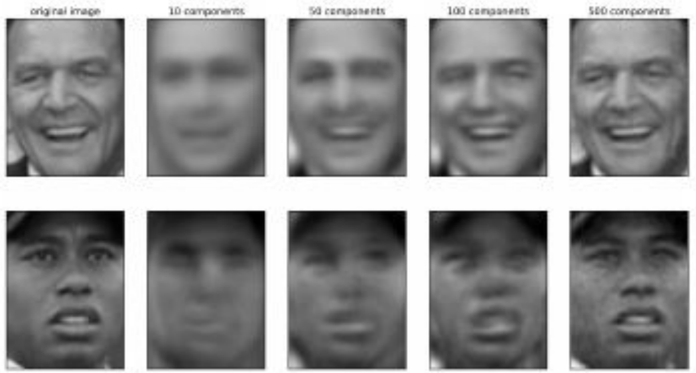


Figure 3-9. Component vectors of the first 15 principal components of the faces dataset

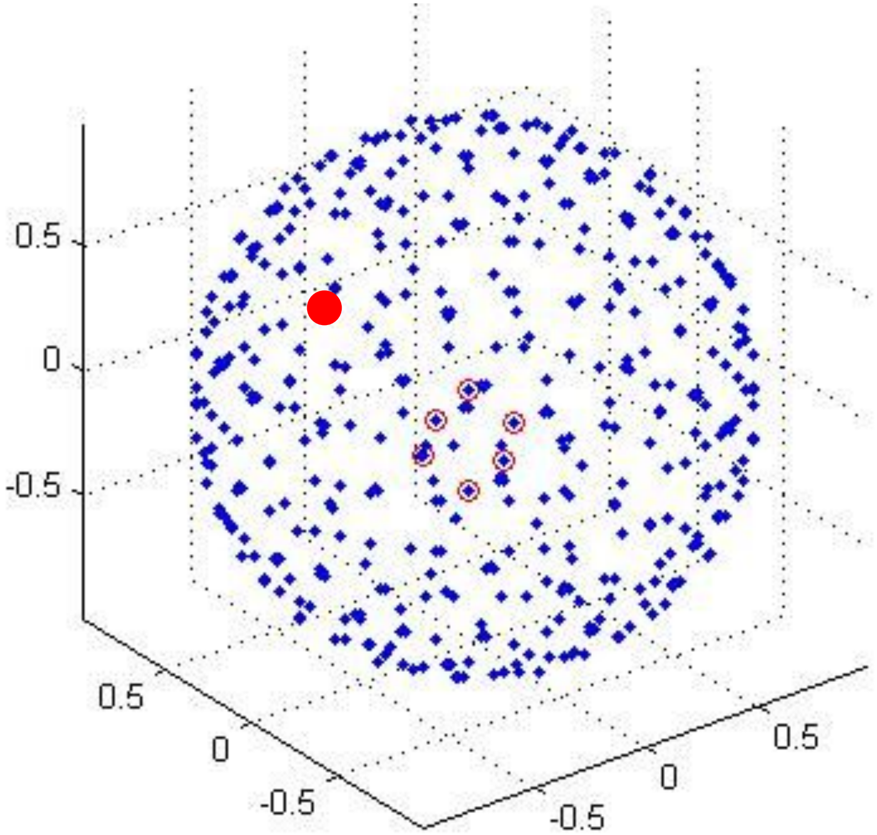




[Muller and Guido, Introduction to Machine Learning with Python]

Non-linear dimensionality reduction

$\mathbf{x}_i = [x_i, y_i, z_i]$

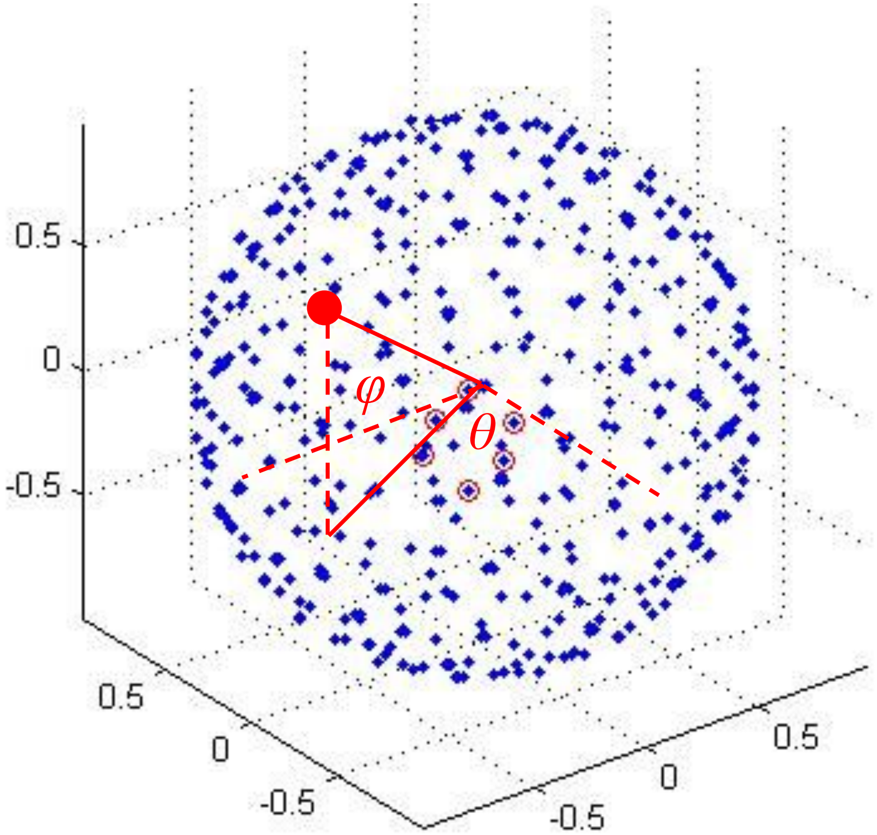


Non-linear dimensionality reduction

$$\mathbf{x}_i = [x_i, y_i, z_i]$$

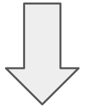
$$\Downarrow$$

$$\mathbf{y}_i = [\varphi_i, \theta_i]$$



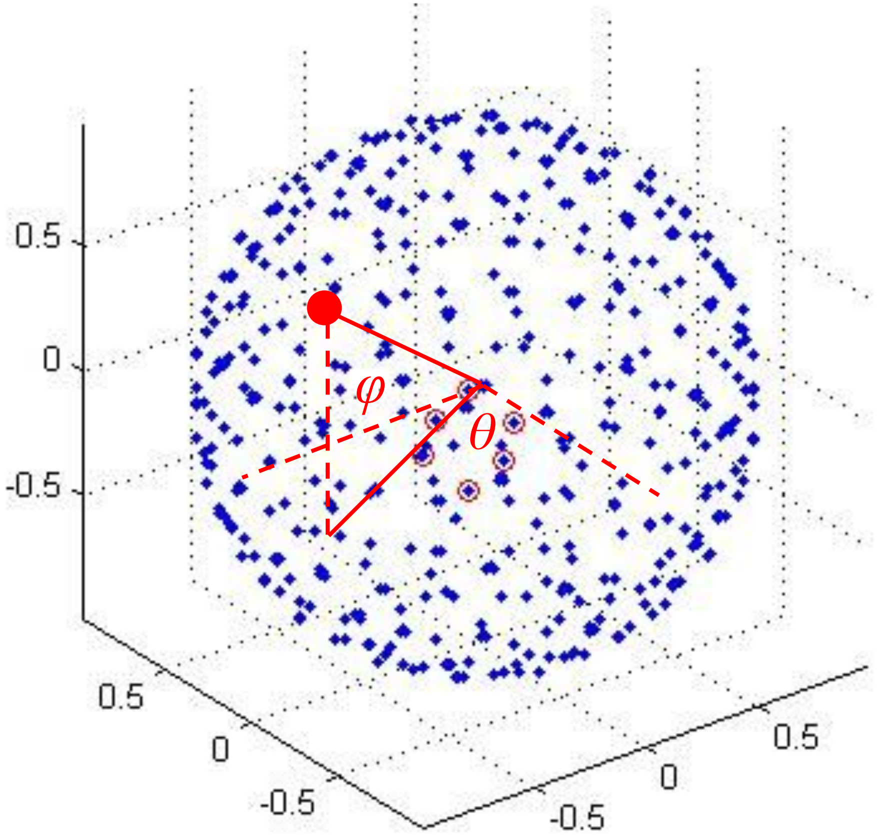
Non-linear dimensionality reduction

$$\mathbf{x}_i = [x_i, y_i, z_i]$$



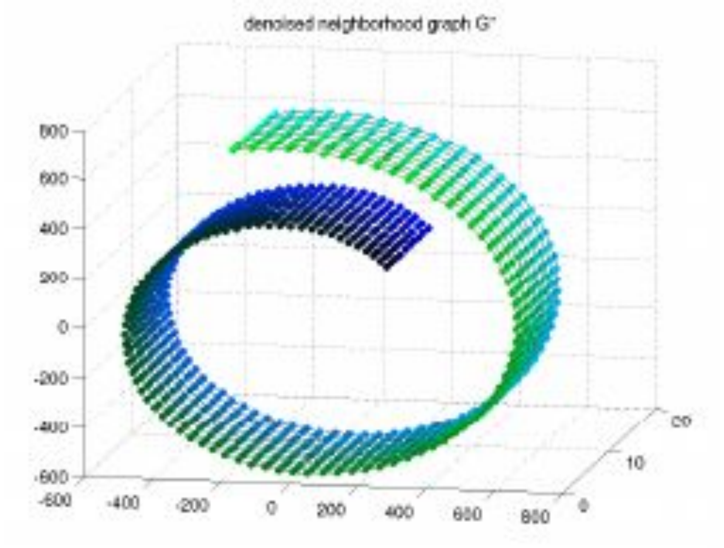
$$\mathbf{y}_i = [\varphi_i, \theta_i]$$

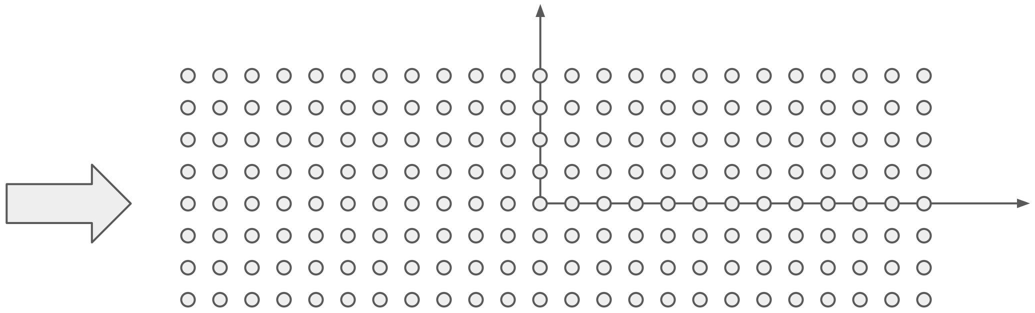
$$\mathbf{x}_i \cong f(\mathbf{y}_i) = [R\cos(\varphi_i)\sin(\theta_i), R\cos(\varphi_i)\cos(\theta_i), R\sin(\varphi_i)]$$



Non-linear Dimensionality Reduction

- Not all subspaces are linear. Can we find a “curved” manifold?
- Also referred to as loop unrolling





Non-linear Dimensionality Reduction

- Key insight: preserve similarity
  - Points that are “close” in the original data should still be close in:
    - the low-dimensional projection
    - the high-dimensional re-projection

minimize $S = \sum_{i,j} (<\mathbf{x}_i,\mathbf{x}_j> - <\mathbf{y}_i, \mathbf{y}_j>)^2$ where $< , >$ denotes dot product

- If we include all points, this is provably identical to PCA

Non-linear Dimensionality Reduction

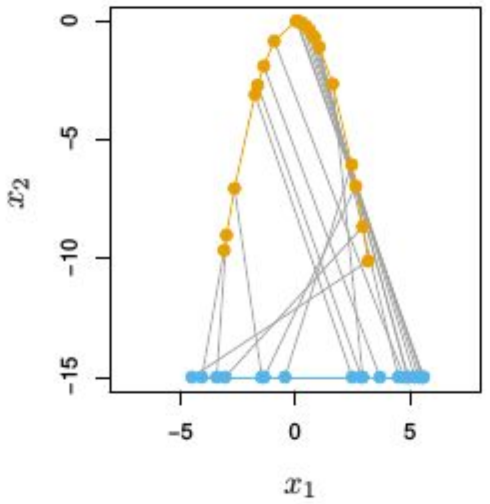
- Key insight: preserve similarity
  - Points that are “close” in the original data should still be close in:
    - the low-dimensional projection
    - the high-dimensional re-projection

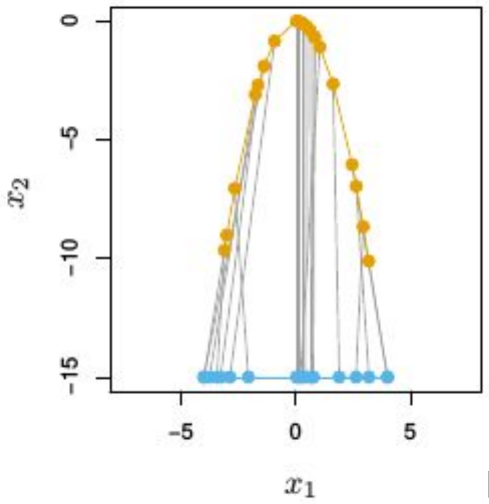
minimize $S = \sum_{i,j} (<\mathbf{x}_i,\mathbf{x}_j> - <\mathbf{y}_i, \mathbf{y}_j>)^2$ where $< , >$ denotes dot product

- If we include all points, this is provably identical to PCA
- For non-linear manifolds:
  - only preserve similarity for points that are close to each other
  - give up on preserving distances for points that are far away

Non-linear Dimensionality Reduction

- Local MDS (Multi-Dimensional Scaling)
  - preserve distances for projections of points that start close by
  - maximize distances for projections of points that start far away





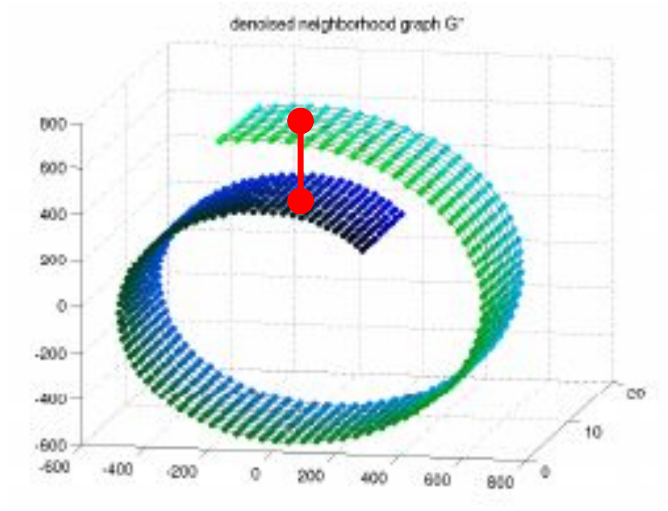
[Hastie et al., The Elements of Statistical Learning]

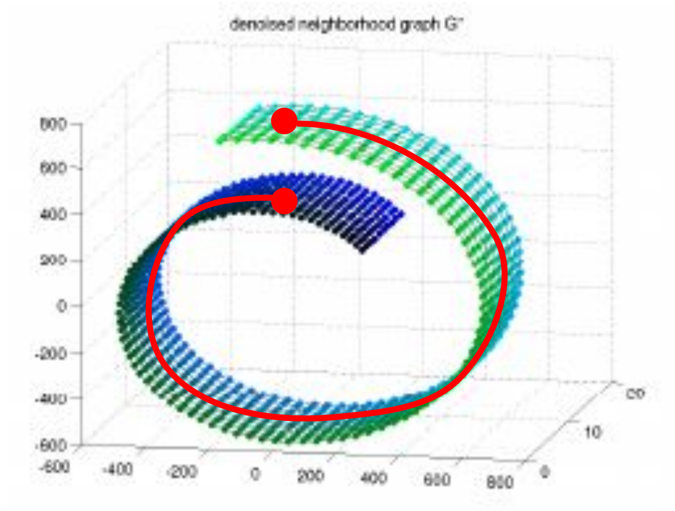
Non-linear Dimensionality Reduction

- Local MDS (Multi-Dimensional Scaling)
- LLE (Locally Linear Embedding)
  - express each point as linear combination of its nearest neighbors
  - find low-dimensional projection (embedding) that preserves this approximation

Non-linear Dimensionality Reduction

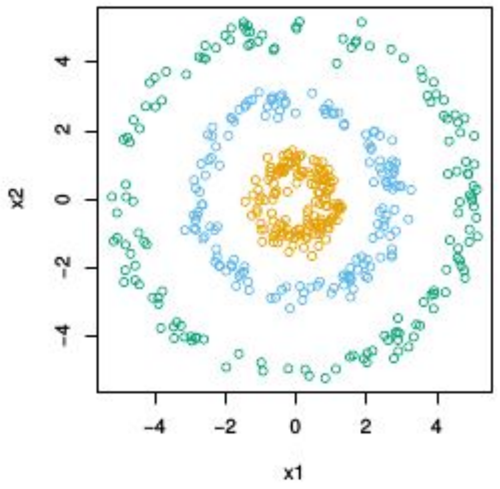
- Local MDS (Multi-Dimensional Scaling)
- LLE (Locally Linear Embedding)
- ISOMAP (ISOmetric feature MAPping)
  - preserve geodesic distance between points





The Kernel Trick

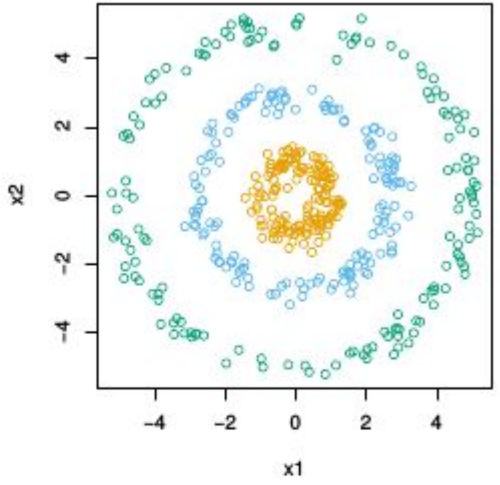
- The problem:
  - I have a linear method for finding structure in my data
  - In my original space, the data has some structure, but it is not linear!



The Kernel Trick

- The problem:
  - I have a linear method for finding structure in my data
  - In my original space, the data has some structure, but it is not linear!
- Idea: perform a non-linear projection of my data into a higher-dimensional space

  e.g.: $\mathbf{x}_i = (x_{i,1}, x_{i,2}) \rightarrow \Phi(\mathbf{x}_i) = (x_{i,1}^2,\ x_{i,1}x_{i,2}\ ,\ x_{i,2}x_{i,1}\ ,\ x_{i,2}^2)$
- Maybe in this new space, the structure is linear
- Problem: the new space is too high-dimensional!



The Kernel Trick

- Perform a non-linear projection of my data into a higher-dimensional space

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

- Problem: the new space is too high-dimensional!

Principal Component Analysis

- Recall: PCA is done via eigenvalue decomposition of the covariance matrix $\mathbf{X}^T\mathbf{X}$, computed based on the centered data matrix $\mathbf{X}$

$$\mathbf{X}^T\mathbf{X} = \begin{bmatrix} \mathbf{x}_1 & \cdots & \mathbf{x}_n \end{bmatrix} \begin{bmatrix} \mathbf{x}_1^T \\ \hline \cdots \\ \hline \mathbf{x}_n^T \end{bmatrix}$$

Principal Component Analysis

- Recall: PCA is done via eigenvalue decomposition of the covariance matrix $\mathbf{X}^T\mathbf{X}$, computed based on the centered data matrix $\mathbf{X}$

$$\mathbf{X}^T\mathbf{X} = \left[\begin{array}{c|c|c} \mathbf{x}_1 & \cdots & \mathbf{x}_n \end{array}\right] \left[\begin{array}{c} \mathbf{x}_1^T \\ \hline \cdots \\ \hline \mathbf{x}_n^T \end{array}\right]$$

- We could also use the Gram matrix, which is larger:

$$\mathbf{X}\mathbf{X}^T = \left[\begin{array}{c} \mathbf{x}_1^T \\ \hline \cdots \\ \hline \mathbf{x}_n^T \end{array}\right] \left[\begin{array}{c|c|c} \mathbf{x}_1 & \cdots & \mathbf{x}_n \end{array}\right] = \begin{bmatrix} \mathbf{x}_1^T\mathbf{x}_1 & \mathbf{x}_1^T\mathbf{x}_2 & \cdots & \\ \mathbf{x}_2^T\mathbf{x}_1 & \mathbf{x}_2^T\mathbf{x}_2 & & \\ \cdots & & \cdots & \\ & & & \mathbf{x}_n^T\mathbf{x}_n \end{bmatrix}$$

Principal Component Analysis

- Recall: PCA is done via eigenvalue decomposition of the covariance matrix $\mathbf{X}^T\mathbf{X}$, computed based on the centered data matrix $\mathbf{X}$

$$\mathbf{X}^T\mathbf{X} = \left[\begin{array}{c|c|c} \mathbf{x}_1 & \cdots & \mathbf{x}_n \end{array}\right] \left[\begin{array}{c} \mathbf{x}_1^T \\ \hline \cdots \\ \hline \mathbf{x}_n^T \end{array}\right]$$

- We could also use the Gram matrix, which is larger:

$$\mathbf{X}\mathbf{X}^T = \left[\begin{array}{c} \mathbf{x}_1^T \\ \hline \cdots \\ \hline \mathbf{x}_n^T \end{array}\right] \left[\begin{array}{c|c|c} \mathbf{x}_1 & \cdots & \mathbf{x}_n \end{array}\right] = \begin{bmatrix} <\mathbf{x}_1, \mathbf{x}_1> & <\mathbf{x}_1, \mathbf{x}_2> & \cdots & \\ <\mathbf{x}_2, \mathbf{x}_1> & <\mathbf{x}_2, \mathbf{x}_2> & & \\ \cdots & & \cdots & \\ & & & <\mathbf{x}_n, \mathbf{x}_n> \end{bmatrix}$$

The Kernel Trick

- Perform a non-linear projection of my data into a higher-dimensional space

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

- Problem: the new space is too high-dimensional!

The Kernel Trick

- Perform a non-linear projection of my data into a higher-dimensional space

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

- Problem: the new space is too high-dimensional!
- Key idea:
  - We don't actually need the high-dimensional points $\Phi(\mathbf{x}_i)$
  - We only need the distances between them $< \Phi(\mathbf{x}_i) , \Phi(\mathbf{x}_j) >$

Gram matrix $\mathbf{K} \subset \Re^{N \times N}, \mathbf{K}_{i,j} = < \Phi(\mathbf{x}_i) , \Phi(\mathbf{x}_j) >$

- Can we compute $\mathbf{K}$ without actually computing each $\Phi(\mathbf{x}_i)$?

The Kernel Trick

Gram matrix $\mathbf{K} \subset \Re^{N \times N}$, $\mathbf{K}_{i,j} = < \Phi(\mathbf{x}_i) , \Phi(\mathbf{x}_j) >$

- Distances can be much easier to compute than high-dimensional points

e.g.: $k(\mathbf{x}_i,\mathbf{x}_j) = <\mathbf{x}_i,\mathbf{x}_j>^2 = < (x_{i,1}{}^2 ,\ x_{i,1}x_{i,2} ,\ x_{i,2}x_{i,1} ,\ x_{i,2}{}^2), (x_{j,1}{}^2 ,\ x_{j,1}x_{j,2} ,\ x_{j,2}x_{j,1} ,\ x_{j,2}{}^2) >$

The Kernel Trick

Gram matrix $\mathbf{K} \subset \Re^{N \times N}$, $\mathbf{K}_{i,j} = < \Phi(\mathbf{x}_i) , \Phi(\mathbf{x}_j) >$

- Distances can be much easier to compute than high-dimensional points

e.g.: $k(\mathbf{x}_i,\mathbf{x}_j) = <\mathbf{x}_i,\mathbf{x}_j>^2 = < (x_{i,1}{}^2 ,\ x_{i,1}x_{i,2} ,\ x_{i,2}x_{i,1} ,\ x_{i,2}{}^2), (x_{j,1}{}^2 ,\ x_{j,1}x_{j,2} ,\ x_{j,2}x_{j,1} ,\ x_{j,2}{}^2) >$

$k(\mathbf{x}_i,\mathbf{x}_j) = <\mathbf{x}_i,\mathbf{x}_j>^n = < (x_{i,1}{}^n ,\ \dots ,\ x_{i,2}{}^n), (x_{j,1}{}^n ,\ \dots ,\ x_{j,2}{}^n) >$

The Kernel Trick

Gram matrix $\mathbf{K} \subset \Re^{N \times N}$, $\mathbf{K}_{i,j} = < \Phi(\mathbf{x}_i) , \Phi(\mathbf{x}_j) >$

- Distances can be much easier to compute than high-dimensional points

e.g.: $k(\mathbf{x}_i,\mathbf{x}_j) = <\mathbf{x}_i,\mathbf{x}_j>^2 = < (x_{i,1}{}^2 , \ x_{i,1}x_{i,2} \ , \ x_{i,2}x_{i,1} \ , \ x_{i,2}{}^2), (x_{j,1}{}^2 , \ x_{j,1}x_{j,2} \ , \ x_{j,2}x_{j,1} \ , \ x_{j,2}{}^2) >$

$k(\mathbf{x}_i,\mathbf{x}_j) = <\mathbf{x}_i,\mathbf{x}_j>^n = < (x_{i,1}{}^n , \ \ldots \ , \ x_{i,2}{}^n), (x_{j,1}{}^n , \ \ldots \ , \ x_{j,2}{}^n) >$

- $k(\mathbf{x}_i,\mathbf{x}_j)$ is referred to as a Kernel function
    - it is equivalent to a distance between high-dimensional projections
    - but much faster to compute than the actual high-dim projections

The Kernel Trick

Gram matrix $\mathbf{K} \subset \Re^{N \times N}$, $\mathbf{K}_{i,j} = < \Phi(\mathbf{x}_i) , \Phi(\mathbf{x}_j) > = k(\mathbf{x}_i , \mathbf{x}_j)$

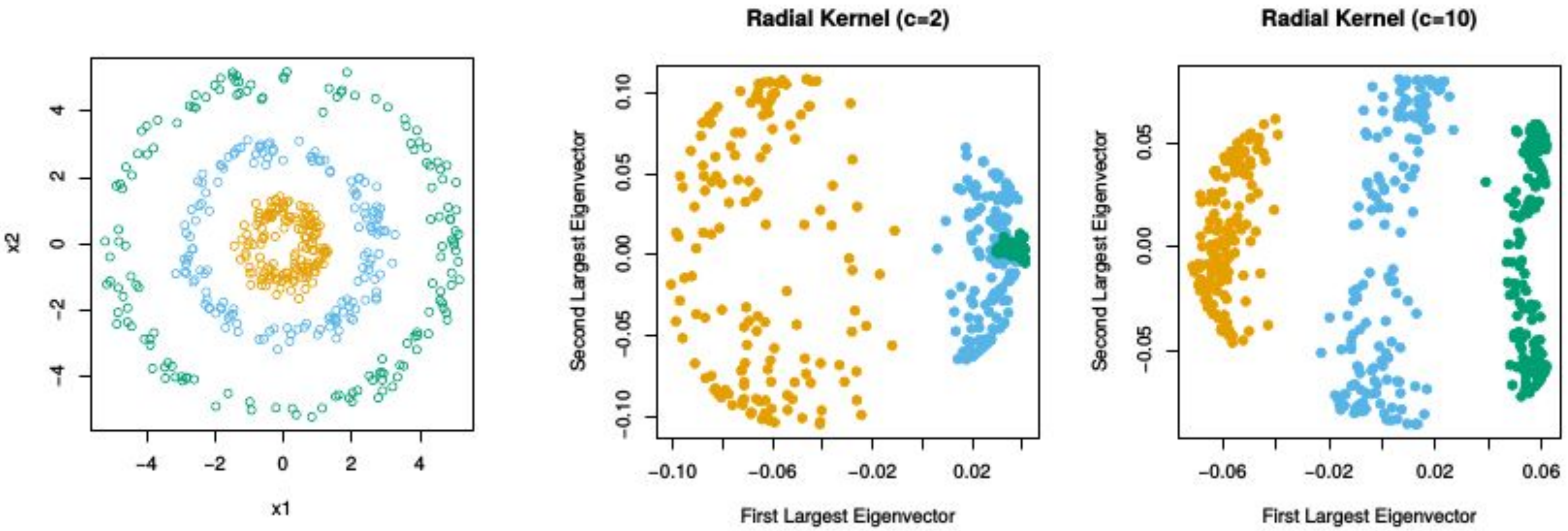
- Commonly used Kernels:
  - Polynomial $k(\mathbf{x}_i,\mathbf{x}_j) = (<\mathbf{x}_i,\mathbf{x}_j>+c)^n$
  - Radial Basis Function $k(\mathbf{x}_i,\mathbf{x}_j) = \exp(- \|x_i - x_j\|^2 / c)$
  - ...

The Kernel Trick

Gram matrix $\mathbf{K} \subset \Re^{N \times N}$, $\mathbf{K}_{i,j} = < \Phi(\mathbf{x}_i) , \Phi(\mathbf{x}_j) > = k(\mathbf{x}_i , \mathbf{x}_j)$

- Commonly used Kernels:
  - Polynomial $k(\mathbf{x}_i,\mathbf{x}_j) = (<\mathbf{x}_i,\mathbf{x}_j>+c)^n$
  - Radial Basis Function $k(\mathbf{x}_i,\mathbf{x}_j) = \exp(- \|x_i - x_j\|^2 / c)$
  - …
- In the end, just apply normal (linear) PCA as eigen-decomposition of matrix $\mathbf{K}$
  - Warning: $\mathbf{K} \subset \Re^{N \times N}$
  - Linear PCA on original data: eigen-decomposition of matrix $\mathbf{X}^T\mathbf{X} \subset \Re^{d \times d}$
  - Kernel PCA yields a subspace of an N-dimensional space, not of the original d-dimensional space.

Kernel PCA



[Hastie et al., The Elements of Statistical Learning]

Recap

- Find a low-dimensional manifold that approximates my data
  - Preserves variance
  - Minimizes re-projection error
- PCA: optimal linear manifold (for Euclidean distance)
  - Eigenvalue decomposition of covariance matrix $\mathbf{X}^T\mathbf{X}$
  - Singular value decomposition of data matrix $\mathbf{X}$
- Non-linear manifolds
  - Try to preserve linear structure “locally”, in small neighborhoods

Recap

- The Kernel trick
  - Equivalent to non-linear projection of data into higher-dimensional space
  - … but with less computation.
  - Linear methods might work well in this new space
- Kernel PCA: PCA after applying the Kernel trick