UNSUPERVISED LEARNING 2011

LECTURE :KERNEL PCA

Rita Osadchy

Some slides are due to Scholkopf, Smola, Muller, and Precup

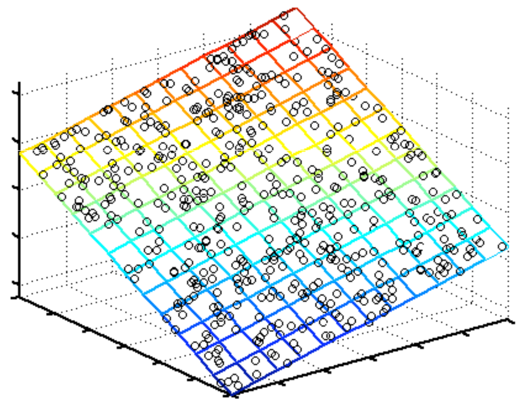
Dimensionality Reduction

- Data representation

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

- Linear structure

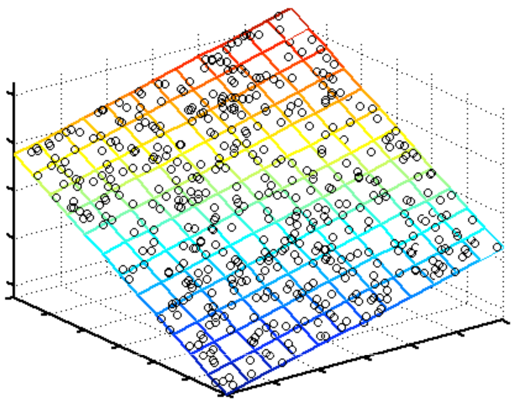
  Does the data live in a low dimensional subspace?



- Nonlinear structure

  Does the data live on a low dimensional submanifold?

Dimensionality Reduction so far



PCA

Manifold learning methods

for non linear structure

Kernel PCA

but does not unfold the data

Notations

- Inputs (**high dimensional**)

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

- Outputs **(low dimensional)**

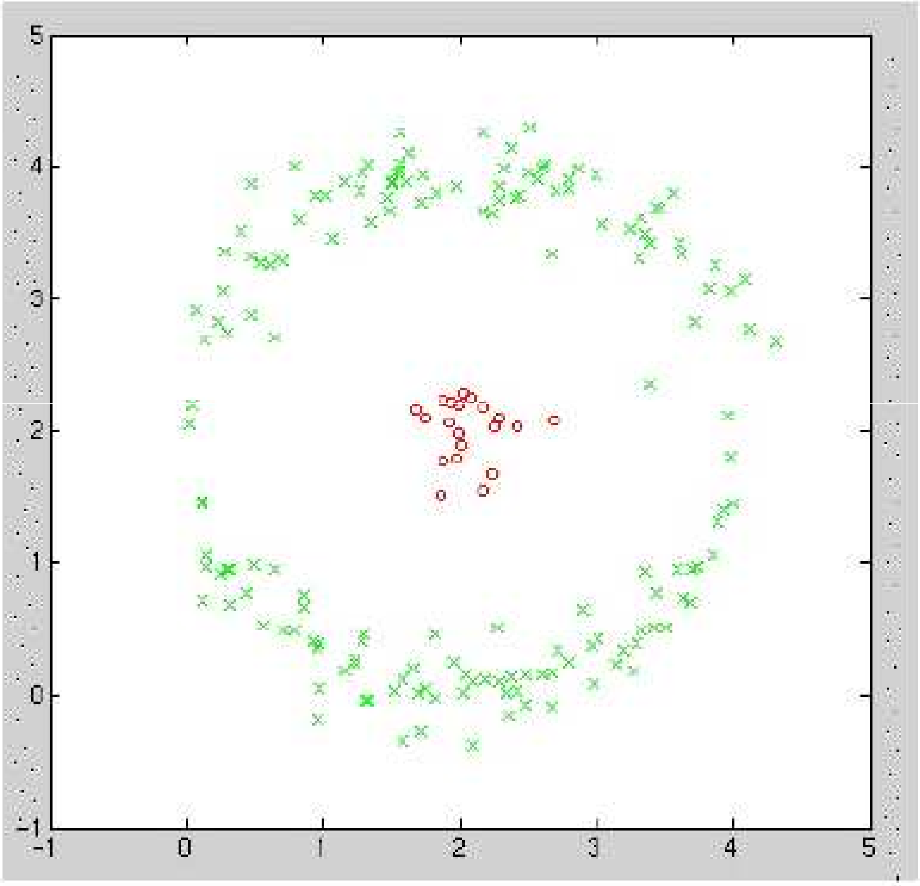
  $\boldsymbol{y_1},\boldsymbol{y_2},\ldots,\boldsymbol{y_n}$ points in R^d ($d<<D$)

The “magic” of high dimensions

- Given some problem, how do we know what classes of functions are capable of solving that problem?

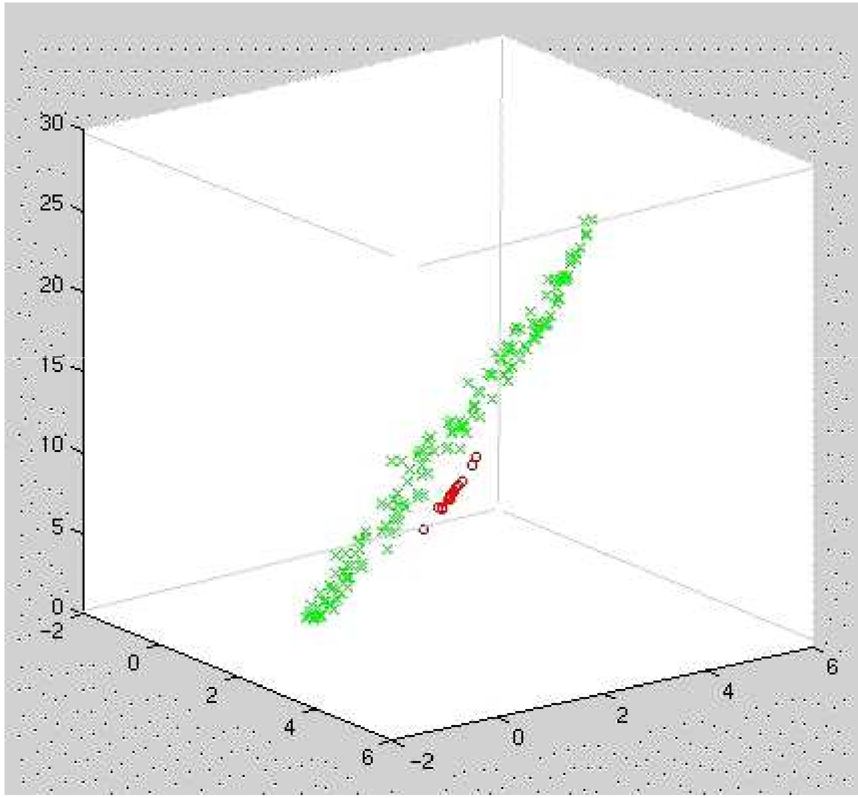
- VC (Vapnik-Chervonenkis) theory tells us that often mappings which take us into a higher dimensional space than the dimension of the input space provide us with greater classification power.

Example in $\mathbb{R}^2$



These classes are linearly inseparable in the input space.

Example: High-Dimensional Mapping



We can make the problem linearly separable by a simple mapping

$$\Phi : \mathbf{R}^2 \rightarrow \mathbf{R}^3$$

$$(x_1, x_2) \mapsto (x_1, x_2, x_1^2 + x_2^2)$$

Kernel Trick

- High-dimensional mapping can seriously increase computation time.
- Can we get around this problem and still get the benefit of high-D?
- Yes! Kernel Trick

$$K\left(x_i, x_j\right) = \phi(x_i)^T \phi(x_j)$$

- Given *any* algorithm that can be expressed solely in terms of dot products, this trick allows us to construct different nonlinear versions of it.

Popular Kernels

Gaussian $\boxed{K(\vec{x},\vec{x}') = \exp\left(-\beta \| \vec{x} - \vec{x}' \|^2\right)}$

Polynomial $\boxed{K(\vec{x},\vec{x}') = (1 + \vec{x} \cdot \vec{x}')^p}$

Hyperbolic tangent $\boxed{K(\vec{x},\vec{x}') = \tanh(\vec{x} \cdot \vec{x}' + \delta)}$

Kernel Principal Component Analysis (KPCA)

- Extends conventional principal component analysis (PCA) to a high dimensional feature space using the “kernel trick”.
- Can extract up to n (number of samples) nonlinear principal components without expensive computations.

Making PCA Non-Linear

- Suppose that instead of using the points x_i we would first map them to some nonlinear feature space $\phi(x_i)$

  E.g. using polar coordinates instead of cartesian coordinates would help us deal with the circle.

- Extract principal component in that space (PCA)
- The result will be non-linear in the original data space!

Derivation

- Suppose that the mean of the data in the feature space is

$$\mu = \frac{1}{n}\sum_{i=1}^{n}\phi(x_i) = 0$$

- Covariance:

$$\mathrm{C} = \frac{1}{n}\sum_{i=1}^{n}\phi(x_i)\phi(x_i)^T$$

- Eigenvectors

$$\mathrm{C}v = \lambda v$$

Derivation Cont.

- Eigenvectors can be expressed as linear combination of features:

$$v = \sum_{i=1}^{n} \alpha_i \phi(x_i)$$

- Proof:

$$\mathrm{C}v = \frac{1}{n} \sum_{i=1}^{n} \phi(x_i)\phi(x_i)^T v = \lambda v$$

thus

$$v = \frac{1}{\lambda n} \sum_{i=1}^{n} \phi(x_i)\phi(x_i)^T v = \frac{1}{\lambda n} \sum_{i=1}^{n} (\phi(x_i) \cdot v)\phi(x_i)^T$$

Showing that $xx^T v = (x \cdot v)x^T$

$$
(xx^T)v = \begin{pmatrix} x_1x_1 & x_1x_2 & \dots & x_1x_M \\ x_2x_1 & x_2x_2 & \dots & x_2x_M \\ \vdots & \vdots & \ddots & \vdots \\ x_Mx_1 & x_Mx_2 & \dots & x_Mx_M \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_M \end{pmatrix}
$$

$$
= \begin{pmatrix} x_1x_1v_1 + x_1x_2v_2 + \dots + x_1x_Mv_M \\ x_2x_1v_1 + x_2x_2v_2 + \dots + x_2x_Mv_M \\ \vdots \\ x_Mx_1v_1 + x_Mx_2v_2 + \dots + x_Mx_Mv_M \end{pmatrix}
$$

Showing that $xx^T v = (x \cdot v)x^T$

$$
= \begin{pmatrix} (x_1 v_1 + x_2 v_2 + \ldots + x_M v_M)\, x_1 \\ (x_1 v_1 + x_2 v_2 + \ldots + x_M v_M)\, x_2 \\ \vdots \\ (x_1 v_1 + x_2 v_2 + \ldots + x_M v_M)\, x_M \end{pmatrix}
$$

$$
= \begin{pmatrix} x_1 v_1 + x_2 v_2 + \ldots + x_M v_M \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_M \end{pmatrix}
$$

$$
= (\boldsymbol{x} \cdot \boldsymbol{v})\boldsymbol{x} \qquad \square
$$

Derivation Cont.

- So, from before we had,

$$v = \frac{1}{n\lambda}\sum_{i=1}^{n}\phi(x_i)\phi(x_i)^T v = \frac{1}{n\lambda}\sum_{i=1}^{n}\underbrace{(\phi(x_i)\cdot v)}_{\text{just a scalar}}\phi(x_i)^T$$

- this means that all solutions v with $\lambda = 0$ lie in the span of $\phi(x_1),..,\phi(x_n)$, i.e.,

$$v = \sum_{i=1}^{n}\alpha_i\phi(x_i)$$

- Finding the eigenvectors is equivalent to finding the coefficients α_i

Derivation Cont.

- By substituting this back into the equation we get:

$$\frac{1}{n}\sum_{i=1}^{n}\phi(x_i)\phi(x_i)^T\left(\sum_{l=1}^{n}\alpha_{jl}\phi(x_l)\right)=\lambda_j\sum_{l=1}^{n}\alpha_{jl}\phi(x_l)$$

- We can rewrite it as

$$\frac{1}{n}\sum_{i=1}^{n}\phi(x_i)\left(\sum_{l=1}^{n}\alpha_{jl}K(x_i,x_l)\right)=\lambda_j\sum_{l=1}^{n}\alpha_{jl}\phi(x_l)$$

- Multiply this by $\phi(x_k)$ from the left:

$$\frac{1}{n}\sum_{i=1}^{n}\phi(x_k)^T\phi(x_i)\left(\sum_{l=1}^{n}\alpha_{jl}K(x_i,x_l)\right)=\lambda_j\sum_{l=1}^{n}\alpha_{jl}\phi(x_k)^T\phi(x_l)$$

Derivation Cont.

- By plugging in the kernel and rearranging we get:

$$\mathrm{K}^2\alpha_j = n\lambda_j \mathrm{K}\alpha_j$$

We can remove a factor of K from both sides of the matrix (this will only affects the eigenvectors with zero eigenvalue, which will not be a principle component anyway):

$$\mathrm{K}\alpha_j = n\lambda_j \alpha_j$$

- We have a normalization condition for the α_j vectors:

$$v_j^T v_j = 1 \;\Rightarrow\; \sum_{k=1}^{n}\sum_{l=1}^{n} \alpha_{jl}\alpha_{jk}\phi(x_l)^T\phi(x_k) = 1 \;\Rightarrow\; \alpha_j^T \mathrm{K}\alpha_j = 1$$

Derivation Cont.

- By multiplying $K\alpha_j = n\lambda_j\alpha_j$ by α_j and using the normalization condition we get:

$$\lambda_j n \alpha_j^T \alpha_j = 1, \quad \forall j$$

- For a new point x, its projection onto the principal components is:

$$\phi(x)^T v_j = \sum_{i=1}^{n} \alpha_{ji} \phi(x)^T \phi(x_i) = \sum_{i=1}^{n} \alpha_{ji} K(x, x_i)$$

Normalizing the feature space

- In general, $\phi(x_i)$ may not be zero mean.
- Centered features:

$$\tilde{\phi}(x_k) = \phi(x_i) - \frac{1}{n}\sum_{k=1}^{n}\phi(x_k)$$

- The corresponding kernel is:

$$\tilde{K}(x_i, x_j) = \tilde{\phi}(x_i)^T \tilde{\phi}(x_j)$$

$$= \left(\phi(x_i) - \frac{1}{n}\sum_{k=1}^{n}\phi(x_k)\right)^T \left(\phi(x_j) - \frac{1}{n}\sum_{k=1}^{n}\phi(x_k)\right)$$

$$= K(x_i, x_j) - \frac{1}{n}\sum_{k=1}^{n}K(x_i, x_k) - \frac{1}{n}\sum_{k=1}^{n}K(x_j, x_k) + \frac{1}{n^2}\sum_{l,k=1}^{n}K(x_l, x_k)$$

Normalizing the feature space (cont)

$$\tilde{K}(x_i, x_j) = K(x_i, x_j) - \frac{1}{n}\sum_{k=1}^{n} K(x_i, x_k) - \frac{1}{n}\sum_{k=1}^{n} K(x_j, x_k) + \frac{1}{n^2}\sum_{l,k=1}^{n} K(x_l, x_k)$$

- In a matrix form

$$\tilde{\mathrm{K}} = \mathrm{K} - 2\mathbf{1}_{1/\mathrm{n}}\mathrm{K} + \mathbf{1}_{1/\mathrm{n}}\mathrm{K}\mathbf{1}_{1/\mathrm{n}}$$

- where $\mathbf{1}_{1/\mathrm{n}}$ is a matrix with all elements 1/n.

Summary of kernel PCA

- Pick a kernel
- Construct the normalized kernel matrix of the data (dimension m x m):

$$\tilde{K} = K - 2\mathbf{1}_{1/n}K + \mathbf{1}_{1/n}K\mathbf{1}_{1/n}$$

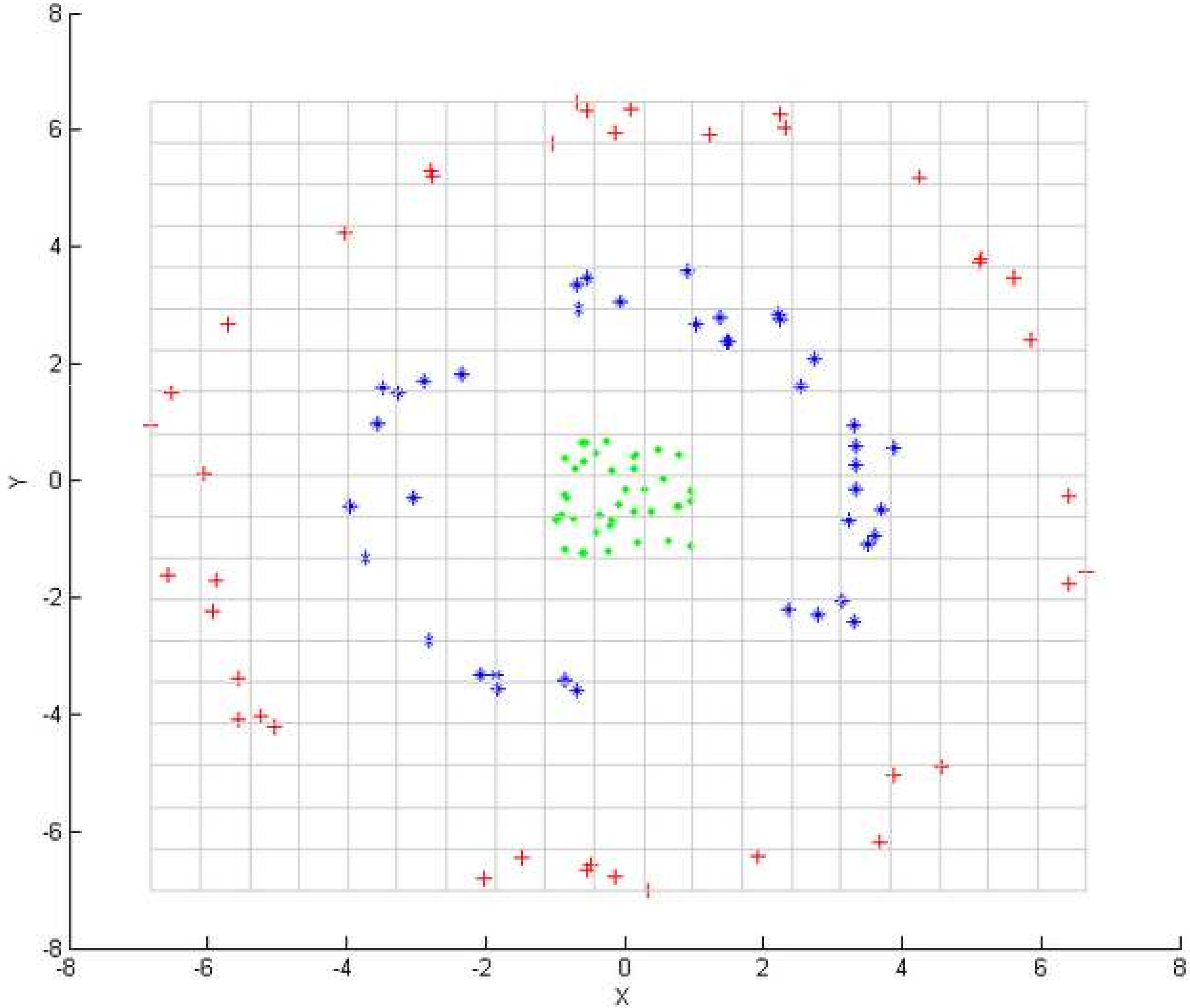
- Solve an eigenvalue problem:

$$\tilde{K}\alpha_i = \lambda_i \alpha_i$$

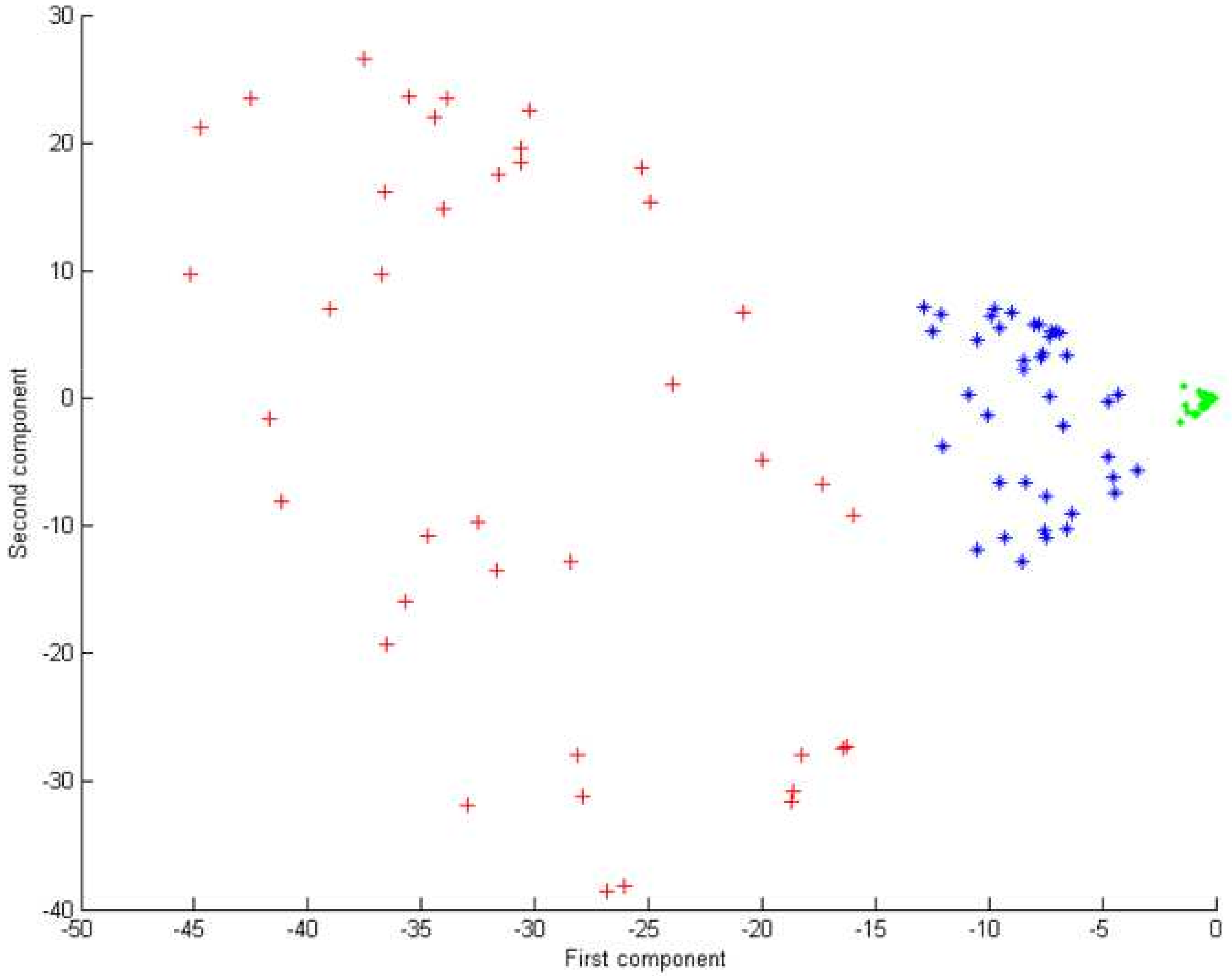
- For any data point (new or old), we can represent it as

$$y_j = \sum_{i=1}^{n} \alpha_{ji} K(x, x_i), \ \mathrm{j} = 1, .., d$$

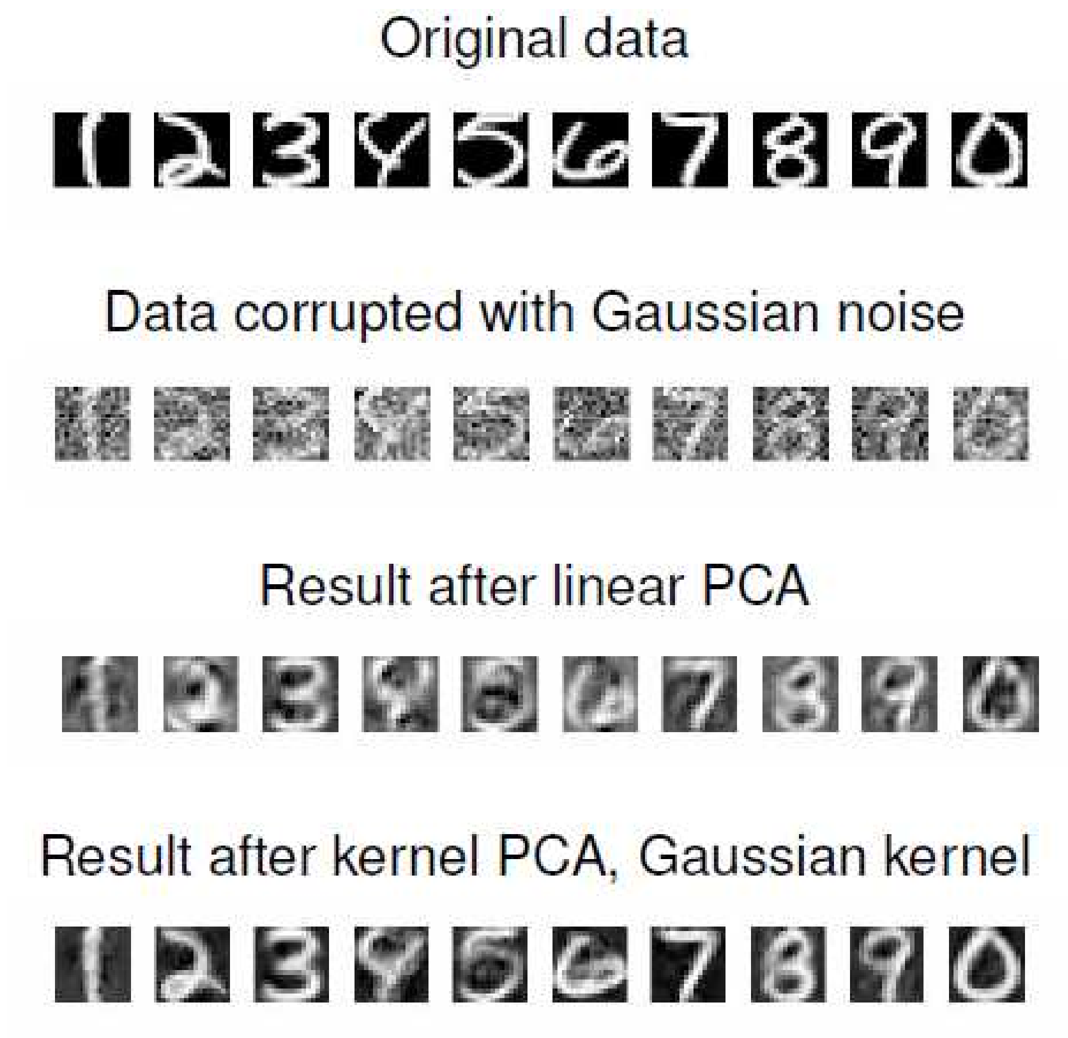
Example: Input Points



Example: KPCA



Example: De-noising images



Properties of KPCA

- Kernel PCA can give a good re-encoding of the data when it lies along a non-linear manifold.
- The kernel matrix is n x n, so kernel PCA will have difficulties if we have lots of data points.