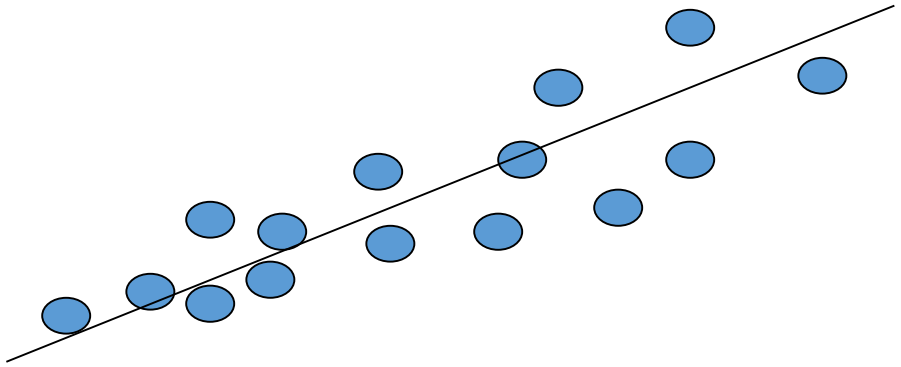
Kernel Principal Component Analysis (Kernel PCA)

Presented by BORAM LEE

September 28, 2015

Algebraic Interpretation

- Given m points in a n dimensional space, for large n, how can we project the points onto a 1 dimensional space?



- Choose a line that fits the data so the points are spread out well along the line.

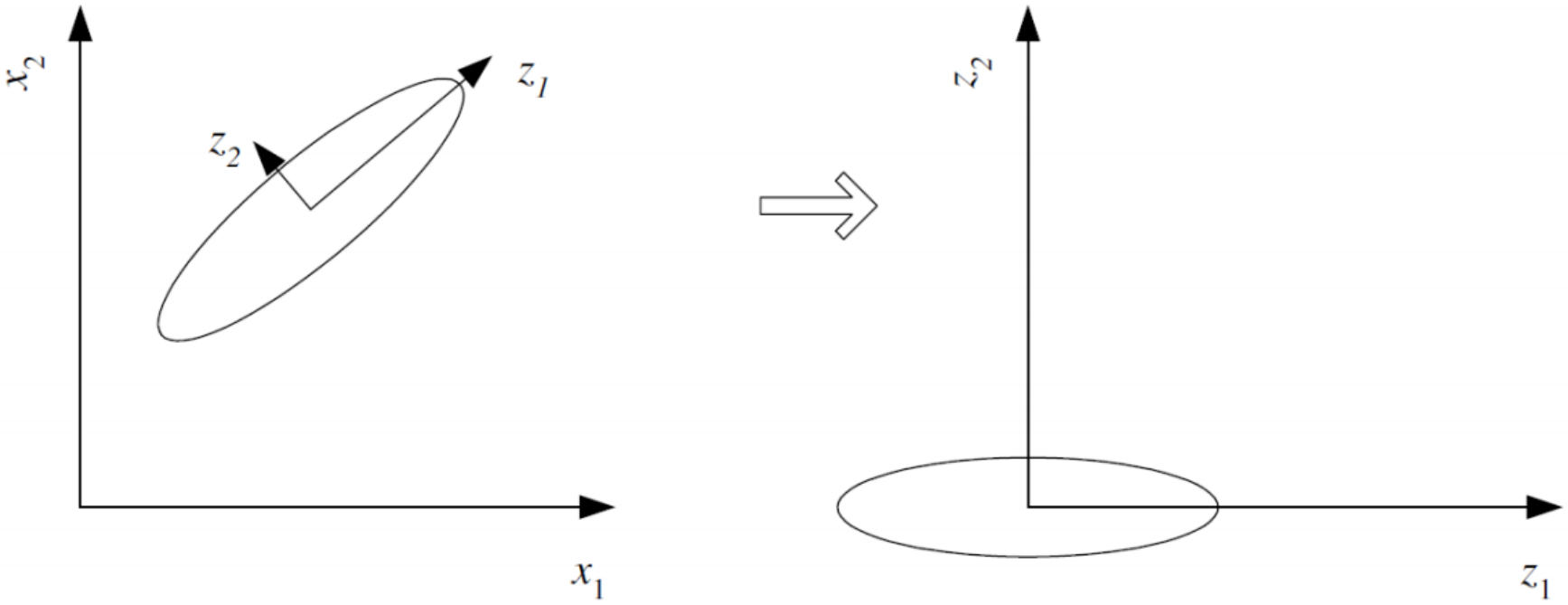
Principal Component Analysis 1

- PCA finds a low-dimensional linear subspace such that when x is projected there information loss is minimized.

- Finds directions of maximal variance.

- Equivalent to finding eigenvalues and eigenvectors of the covariance matrix.

Principal Component Analysis 2



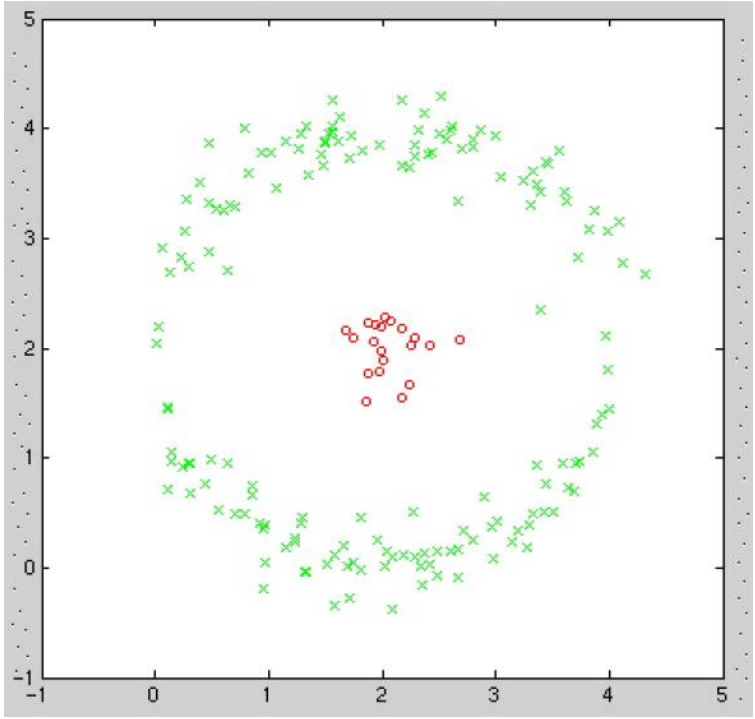
- PCA centers the sample and then rotates the axes to line up with the directions of highest variance. If the variance on Z_2 is too small, it can be ignored and we have dimensionality reduction from two to one.

Principal Component Analysis 3

Solution:

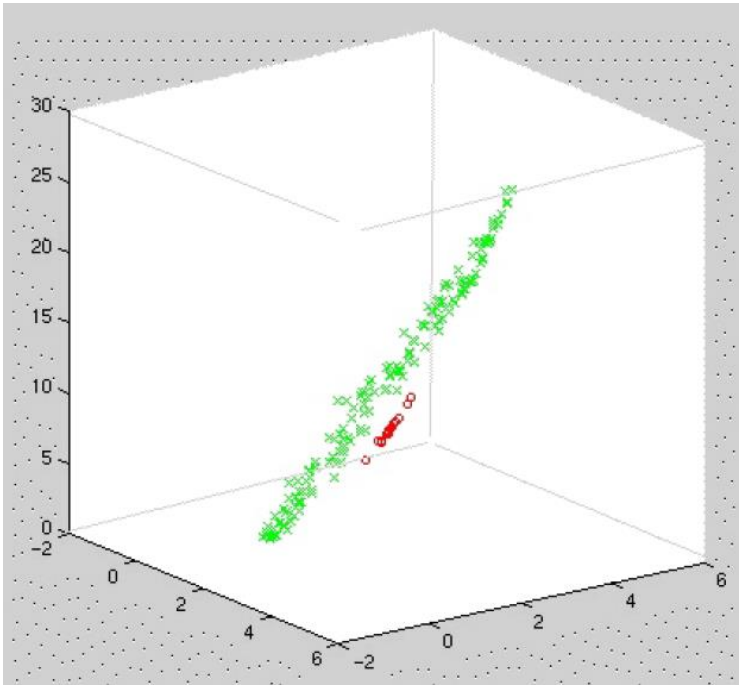
- More formally: data $\mathcal{X} = \{\mathbf{x}^t\}_{t=1}^N$, $\mathbf{x}^t \in \mathbb{R}^d$
- Center data: $\mathbf{y}^t = \mathbf{x}^t - \mathbf{m}$, where $\mathbf{m} = \sum_t \mathbf{x}^t / N$
- Compute the covariance matrix $\mathbf{S} = \sum_t \mathbf{y}\mathbf{y}^T / N$
- Diagonalize **S** using Spectral Decomposition: $\mathbf{C}^T\mathbf{S}\mathbf{C} = \mathbf{D}$ where
  - **C** is an orthogonal (rotation) matrix satisfying $\mathbf{C}\mathbf{C}^T = \mathbf{C}^T\mathbf{C} = \mathbf{1}$ (identity matrix), and
  - **D** is a diagonal matrix whose diagonal elements are the eigenvalues $\lambda_1 \geq \ldots \geq \lambda_d \geq 0$
- *i*:th column of **C** is the *i*:th eigenvector.
- Project data vectors $\mathbf{y}^t$ to principal components $\mathbf{z}^t = \mathbf{C}^T\mathbf{y}^t$ (equivalently $\mathbf{y}^t = \mathbf{C}\mathbf{z}^t$).

Ring with Center (in R^2)



- Two classes are not linearly separable in the input space.

Simple Mapping into R^3



- We can make the problem linearly separable by a simple mapping into R^3

If we think differently …

Problems with High Dimensionality

- Can seriously increase computation time.

- Let's consider the product features used in the paper, which take all possible d-th order products of the elements of the input vector and write them into a new vector.

- 16×16 pixel input images with a monomial degree d =5 yields 10^{10} different monomials.

Kernel Trick

- Can we get around this problem and still get the benefit of high dimensionality? ***Yes***.

- Sometimes it is possible to compute dot-products without explicitly mapping into the high dimensional feature space. We can employ dot products of the form

$$k(x, x') = \langle \Phi(x), \Phi(x') \rangle$$

- The phi function means that we send a data into another space.

$$\Phi : \mathcal{X} \to \mathcal{H}, \mathrm{x} \mapsto \Phi(\mathrm{x})$$

Kernel PCA 1

- Denote the nonlinear feature transformation here by Φ and the set of ℓ transformed input samples by $\Phi(\mathbf{x}_1), \ldots, \Phi(\mathbf{x}_\ell)$
- The covariance matrix we need in PCA is then

$$\bar{C} = \frac{1}{\ell} \sum_{j=1}^{\ell} \Phi(\mathbf{x}_j) \Phi(\mathbf{x}_j)^\top$$

- We must compute each eigenvalue $\lambda \geq 0$ and eigenvector **V** for the matrix satisfying $\lambda \mathbf{V} = \bar{C}\mathbf{V}$.
- It turns out the solutions **V** lie in the span of the data: they are linear combinations of $\Phi(\mathbf{x}_1), \ldots, \Phi(\mathbf{x}_\ell)$. The eigenvalue problem becomes $\lambda(\Phi(\mathbf{x}_k) \cdot \mathbf{V}) = (\Phi(\mathbf{x}_k) \cdot \bar{C}\mathbf{V})$ for all $k = 1, \ldots, \ell$, where

$$\mathbf{V} = \sum_{i=1}^{\ell} \alpha_i \Phi(\mathbf{x}_i) \quad \text{for some coefficients} \quad \alpha_1, \ldots, \alpha_\ell$$

- The eigenvalue problem can be written in terms of the coefficients!

Kernel PCA 2

- Inserting the equation for V in terms of the coefficients, the eigenvalue problem becomes $\ell\lambda K\boldsymbol{\alpha} = K^2\boldsymbol{\alpha}$ where
  K is an ℓ x ℓ inner-product (kernel) matrix: $K_{ij} := (\Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j))$
  $\boldsymbol{\alpha}$ is a column vector with entries $\alpha_1, \ldots, \alpha_\ell$
- To solve the eigenvalue problem, solve instead $\ell\lambda\boldsymbol{\alpha} = K\boldsymbol{\alpha}$ for nonzero eigenvalues.
- Principal component projection directions are normalized to have unit norm, $(\mathbf{V}^k \cdot \mathbf{V}^k) = 1$. Inserting the definitions, that becomes

$$1 = \sum_{i,j=1}^{\ell} \alpha_i^k \alpha_j^k (\Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j)) = (\boldsymbol{\alpha}^k \cdot K\boldsymbol{\alpha}^k) = \lambda_k(\boldsymbol{\alpha}^k \cdot \boldsymbol{\alpha}^k)$$

  thus the square root of λ_k becomes the normalization factor for $\boldsymbol{\alpha}^k$
- To project a test point **x** onto the k:th eigenvector:

$$(\mathbf{V}^k \cdot \Phi(\mathbf{x})) = \sum_{i=1}^{\ell} \alpha_i^k (\Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}))$$

- None of the operations require the actual transformed features, the inner products between them are enough!

Kernel PCA 3

- Many interesting kernels can be defined that satisfy the properties of an inner product between some transformed features, but the transformed features themselves would be expensive/impossible to compute!
- Polynomial kernel of order d: $k(\mathbf{x}, \mathbf{y}) = (\mathbf{x} \cdot \mathbf{y})^d$ (corresponds to a transformation onto all products of d original input values)
- Radial basis function: $k(\mathbf{x}, \mathbf{y}) = \exp\left(-||\mathbf{x} - \mathbf{y}||^2/(2\,\sigma^2)\right)$
- Sigmoid kernel: $k(\mathbf{x}, \mathbf{y}) = \tanh(\kappa(\mathbf{x} \cdot \mathbf{y}) + \Theta)$

parameters of the kernel

- After the kernels have been computed, computational complexity depends on the size of the kernel matrix but not on the original input dimensionality or the transformed input dimensionality

Polynomial Kernel Example

$$k(\mathbf{x}, \mathbf{y}) = \langle \mathbf{x}, \mathbf{y} \rangle^d$$

If $d = 2$ and $\mathbf{x}, \mathbf{y} \in \mathbb{R}^2$, then

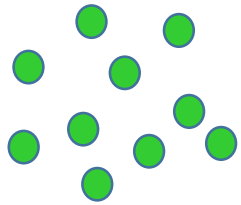
$$\begin{aligned}
\langle \mathbf{x}, \mathbf{y} \rangle^2 &= \left\langle \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}, \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} \right\rangle^2 \\
&= (x_1 y_1 + x_2 y_2)^2 \\
&= \left\langle \begin{bmatrix} x_1^2 \\ x_2^2 \\ \sqrt{2} x_1 x_2 \end{bmatrix}, \begin{bmatrix} y_1^2 \\ y_2^2 \\ \sqrt{2} y_1 y_2 \end{bmatrix} \right\rangle \\
&= \langle \phi(\mathbf{x}), \phi(\mathbf{y}) \rangle
\end{aligned}$$

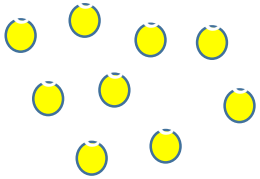
Radial Basis Function Example

```
# Python Code for Radial Basis Function

def rbf(v1,v2,gamma=10):
  dv=[v1[i]-v2[i] for i in range(len(v1))]
  l=veclength(dv)
  return math.e**(-gamma*l)
```

- In high dimension, which side is the red point(v) closer to?



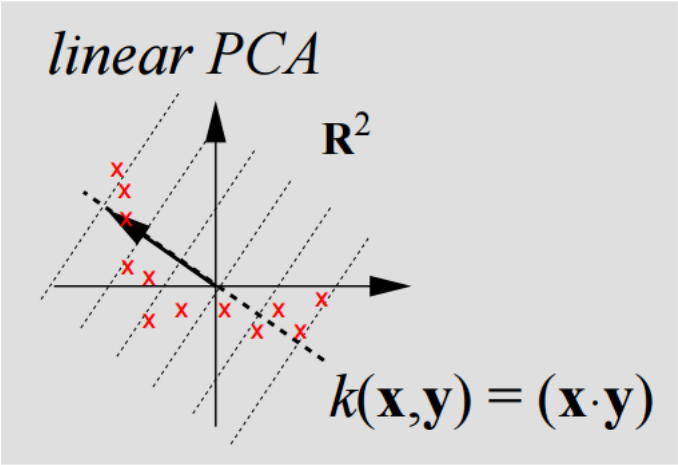


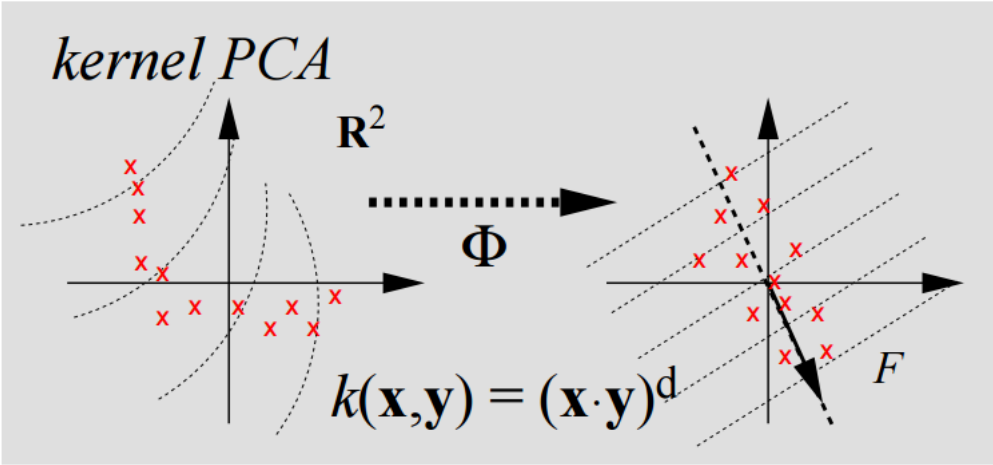
$$\frac{\sum_{i=1}^{M} rbf(v, v_i, 10)}{M}$$

$$\frac{\sum_{j=1}^{N} rbf(v, v_j, 10)}{N}$$

- We only need to compute kernel function rather dot products.

Linear PCA vs Nonlinear PCA

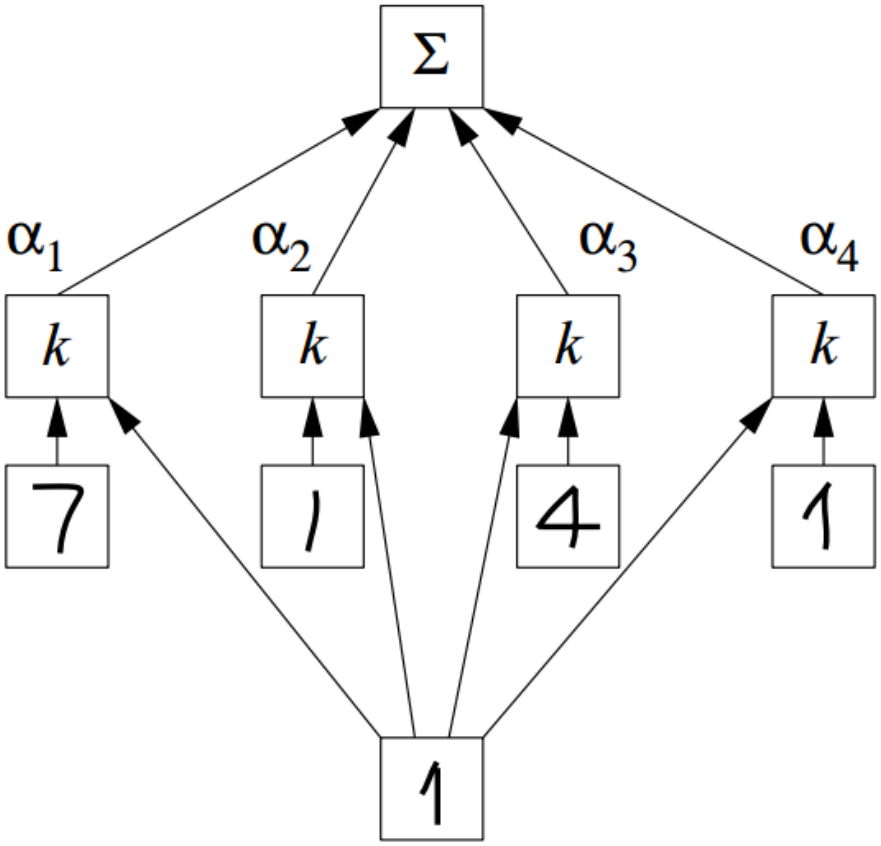




Polynomial Kernel of Order d

- Using nonlinear PCA implicitly causes PCA to be done in a high-dimensional space.
- Dotted lines are contours of lines of constant feature value.

Feature Extraction for an OCR task



feature value $(\Phi(\mathbf{x})\cdot V) = \Sigma\, \alpha_i\, k(\mathbf{x},\mathbf{x}_i)$

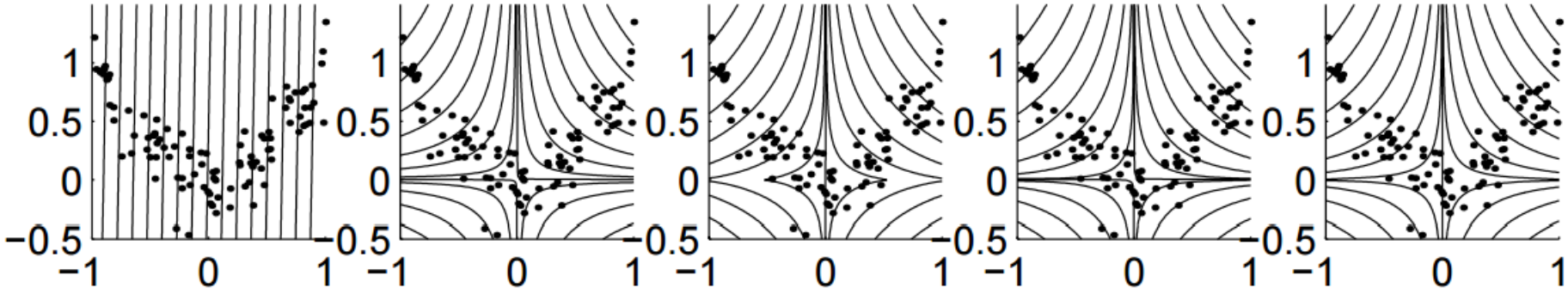
weights (Eigenvector coefficients)

comparison: $k(\mathbf{x},\mathbf{x}_i)$

sample $\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \ldots$

input vector $\mathbf{x}$

Experiment (Polynomial Kernel)



- Displayed are contour lines of constant value of the first principal component.

- Nonlinear kernels (d >= 2) extract features which nicely increase along the direction of main variance in the data.

Pattern Recognition Problem

- By using kernel PCA, a good experiment result was obtained.

Dimension	Components #	Error Rate (%)
1	128	8.6
2, 3, 4, 5, 6	128 2048	6 *4*

Essence of Kernel PCA

- Advantage
  - The kernel method can be applied to any algorithm which can be formulated solely in terms of dot products.
  - The computational complexity does not grow with the dimensionality of the feature space that we are implicitly working in. This makes it possible to work for instance in the space of all possible d-th order products between pixels of an image.

- Disadvantage
  - Can become inefficient when a new pattern is added.
  - Kernel matrix size ($\ell \times \ell$) can become infeasible .

- Future work
  - Many applications in terms of classification, regression, and novelty detection

Reference

- Disadvantage in Kernel PCA
    - Statistical Data Mining and Knowledge Discovery
      (by Hamparsum Bozdogan, Jul 29, 2003)

Rewriting PCA in terms of dot products

First, we need to remember that the eigenvectors lie in the span of $x_1 \dots x_n$ **Proof**:

$$C\boldsymbol{v} = \frac{1}{m}\sum_{j=1}^{m} x_j x_j^\top \boldsymbol{v} = \lambda \boldsymbol{v}$$

Thus,

$$\begin{aligned} \boldsymbol{v} &= \frac{1}{m\lambda}\sum_{j=1}^{m} x_j x_j^\top \boldsymbol{v} \\ &= \frac{1}{m\lambda}\sum_{j=1}^{m} (x_j \cdot \boldsymbol{v}) x_j \end{aligned}$$

Show that $(\boldsymbol{x}\boldsymbol{x}^T)\boldsymbol{v} = (\boldsymbol{x}\cdot\boldsymbol{v})\boldsymbol{x}$

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

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

$$
\begin{aligned}
&= \begin{pmatrix} (x_1 v_1 + x_2 v_2 + \ldots + x_M v_M)\, x_1 \\ (x_1 v_1 + x_2 v_2 + \ldots + x_M v_M)\, x_2 \\ \vdots \\ (x_1 v_1 + x_2 v_2 + \ldots + x_M v_M)\, x_M \end{pmatrix} \\
&= \begin{pmatrix} x_1 v_1 + x_2 v_2 + \ldots + x_M v_M \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_M \end{pmatrix} \\
&= (\boldsymbol{x} \cdot \boldsymbol{v})\boldsymbol{x}
\end{aligned}
$$

So, from before, we had

$$\begin{aligned} \boldsymbol{v} &= \frac{1}{m\lambda} \sum_{j=1}^{m} x_j x_j^\top \boldsymbol{v} \\ &= \frac{1}{m\lambda} \sum_{j=1}^{m} (x_j \cdot \boldsymbol{v}) x_j \end{aligned}$$

But $(x_j \cdot \boldsymbol{v})$ is just a scalar, so this means that all solutions $\boldsymbol{v}$ with $\lambda \neq 0$ lie in the span of $x_1 \ldots x_m$, i.e.,

$$\boldsymbol{v} = \sum_{i=1}^{m} \alpha_i x_i$$