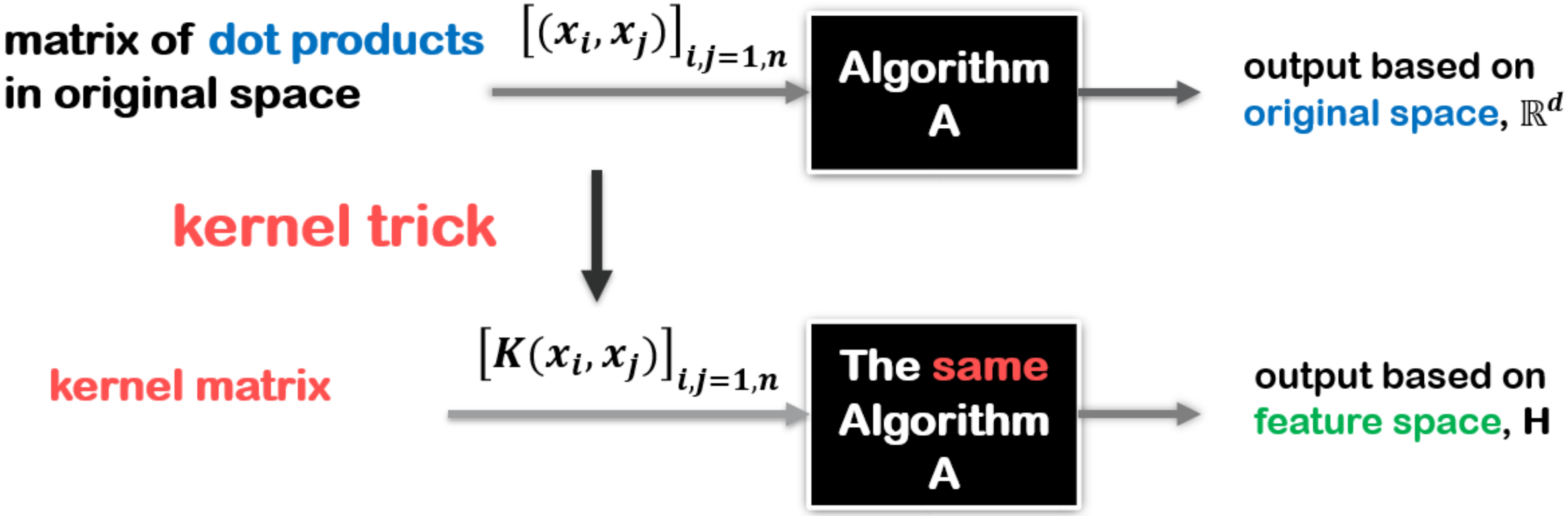
Kernels and Kernel Tricks: Non-linearity



- Kernel
- Kernel Matrix
- Kernel Trick
- Kernelization

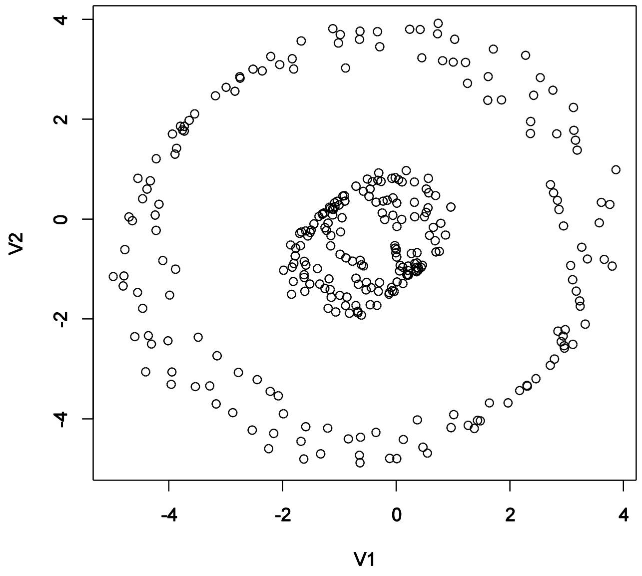
Prof. Nagiza F. Samatova
samatova@csc.ncsu.edu
Department of Computer Science
North Carolina State University

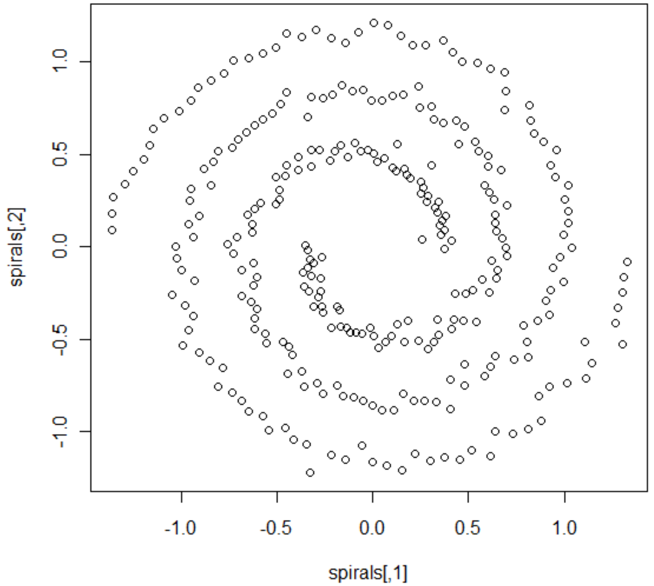
NC STATE Executive Education

February 2019

Motivation: PCA-based Dimension Reduction

- **Standard PCA does not work for all datasets**

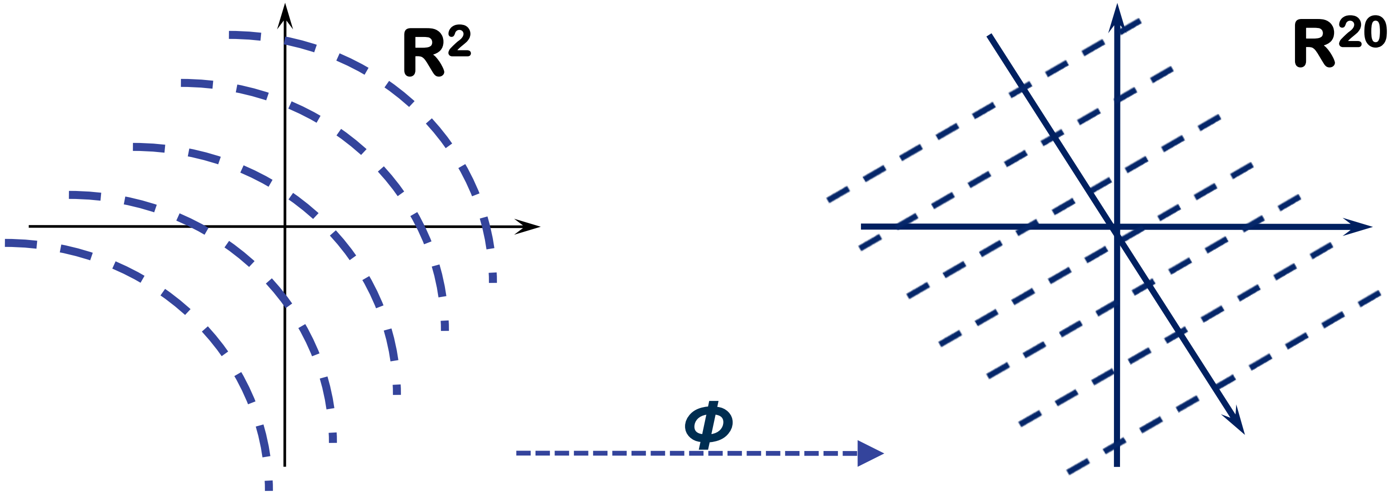




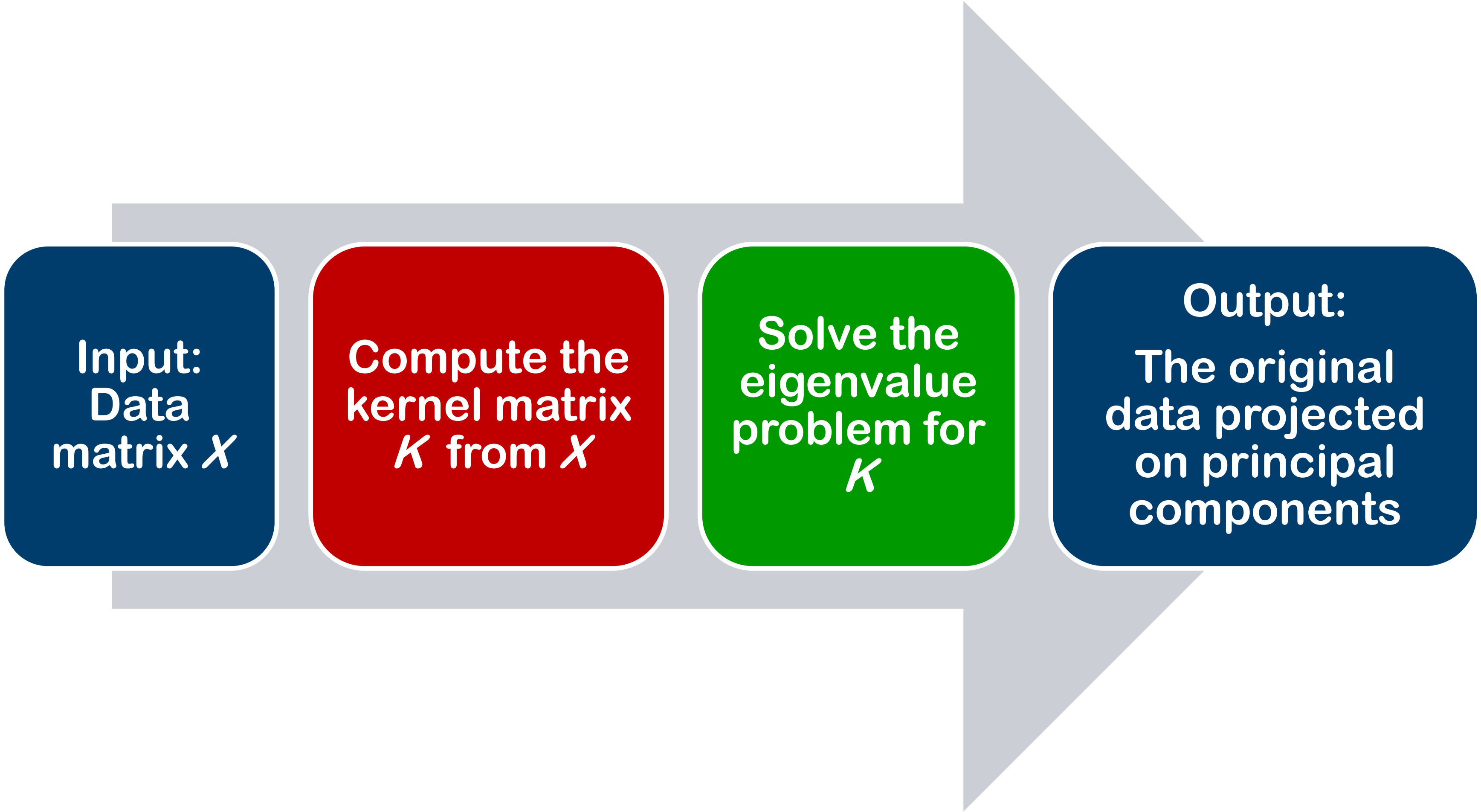
- **Absence of a hyper-plane or principal components to separate data clearly**

Solution: Input space → Feature space

- **Map data objects in non-linear input space to a linear feature space (Hilbert)**
  - Potentially increases dimensionality of the data



Kernel-PCA


Input: Data matrix X
Compute the kernel matrix K from X
Solve the eigenvalue problem for K
Output:
The original data projected on principal components

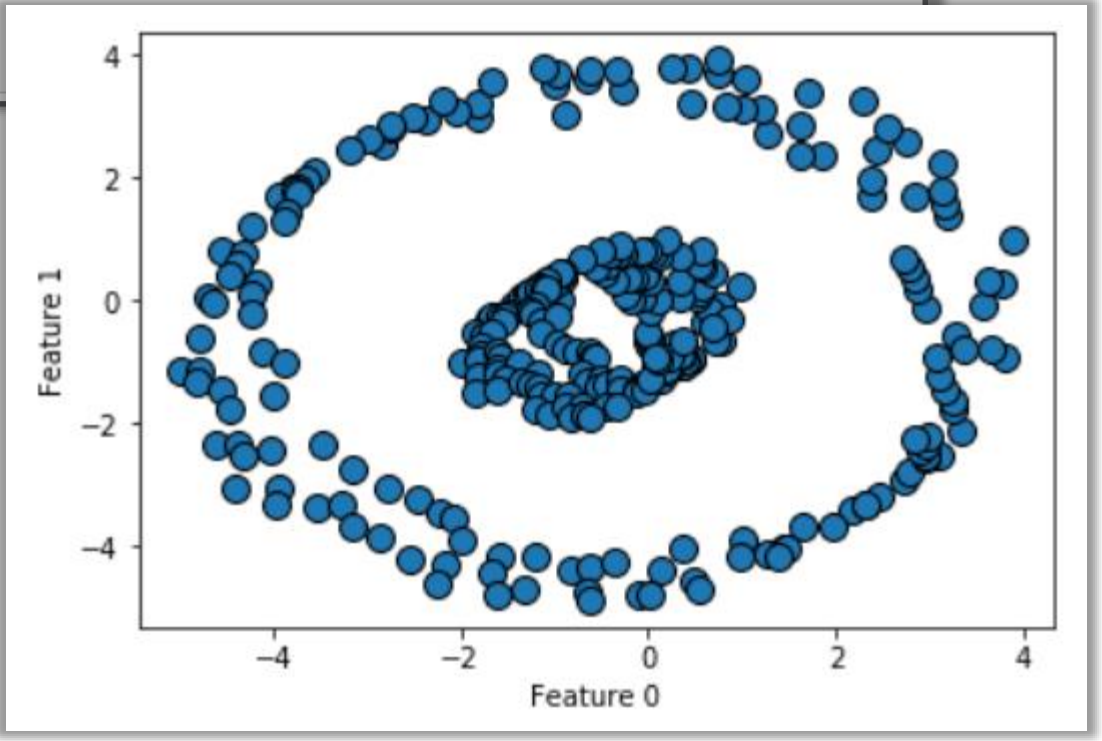
What was the Trick?

- **Kernel-PCA performs PCA in a feature space without explicitly mapping the input space to a feature space!**

DR technique without a kernel	DR using a kernel
Requires an explicit mapping	Does not require an explicit mapping
Maps each data object to feature space	Does not explicitly map each data object
Requires a covariance matrix in feature space	Requires a kernel matrix
Solves the eigenvalue problem for the covariance matrix	Solves the eigenvalue problem on the kernel matrix

Code: Load and Plot the Non-Linear Data

```python
import mglearn
import pandas as pd
circles = pd.read_csv("../data_raw/data_prep_kernels_circles.csv")
mglearn.discrete_scatter(circles.iloc[:,0], circles.iloc[:,1])
plt.xlabel("Feature 0")
plt.ylabel ("Feature 1")
plt.show()
```



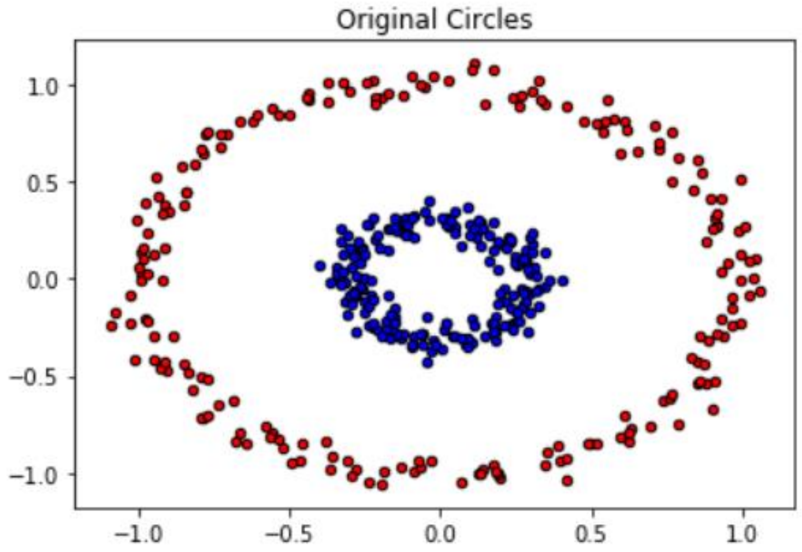
Linear dimension reduction:

- **any projection onto Feature-0 axis or Feature-1 will fail to separate the points on two different circles**

Kernel PCA

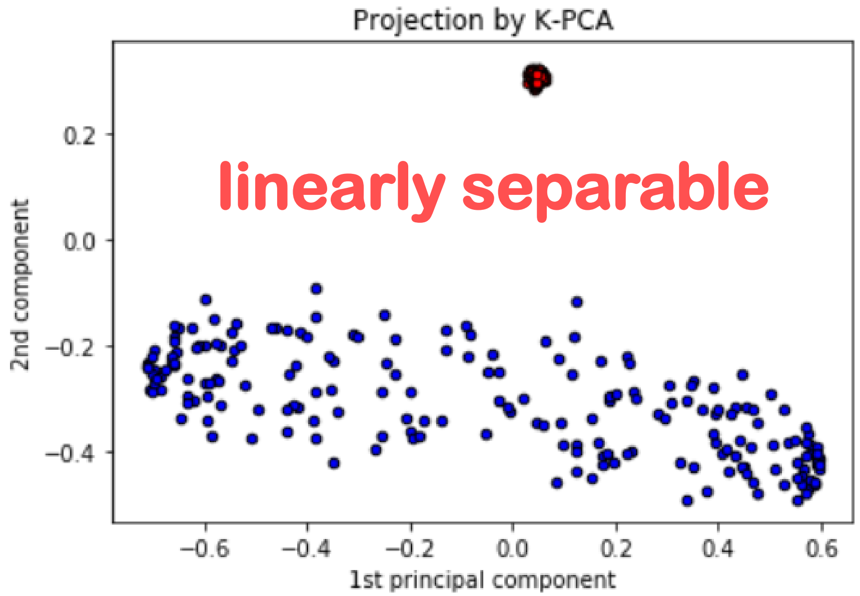
```python
from sklearn.datasets import make_circles
np.random.seed(0)
X, y = make_circles(n_samples=400, factor=.3, noise=.05)
```

```python
plt.figure()
plt.title("Original Circles")
reds = y == 0
blues = y == 1
plt.scatter(X[reds, 0], X[reds, 1], c="red",
            s=20, edgecolor='k')
plt.scatter(X[blues, 0], X[blues, 1], c="blue",
            s=20, edgecolor='k')
plt.show()
```



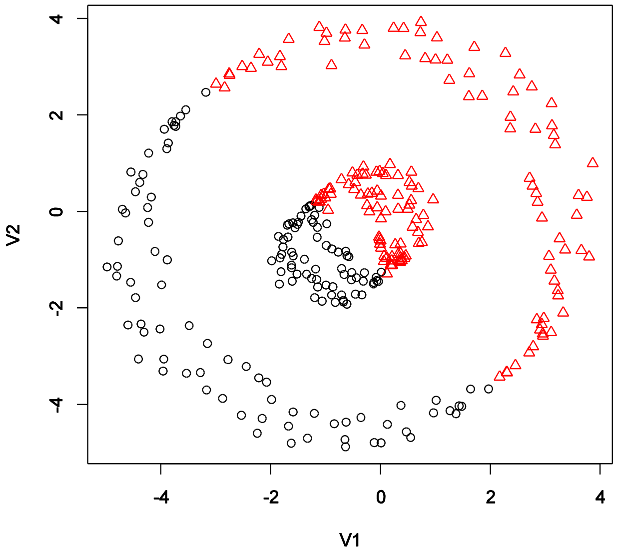

```python
from sklearn.decomposition import KernelPCA
kpca = KernelPCA(kernel="rbf", fit_inverse_transform=True, gamma=10)
X_kpca = kpca.fit_transform(X)
```

```python
plt.scatter(X_kpca[reds, 0], X_kpca[reds, 1], c="red",
            s=20, edgecolor='k')
plt.scatter(X_kpca[blues, 0], X_kpca[blues, 1], c="blue",
            s=20, edgecolor='k')
plt.title("Projection by K-PCA")
plt.xlabel("1st principal component")
plt.ylabel("2nd component")
plt.show()
```



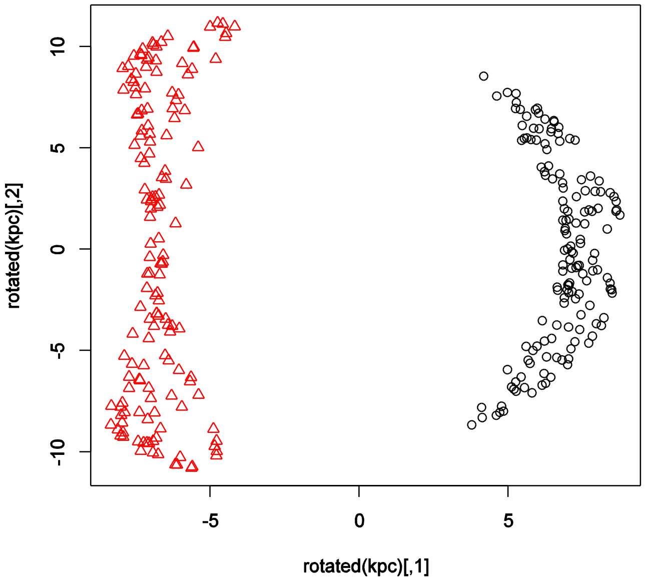
Example: Kernel-PCA assisting Clustering

k-means clustering before kernel-PCA



Incorrect Clusters

k-means clustering after kernel-PCA

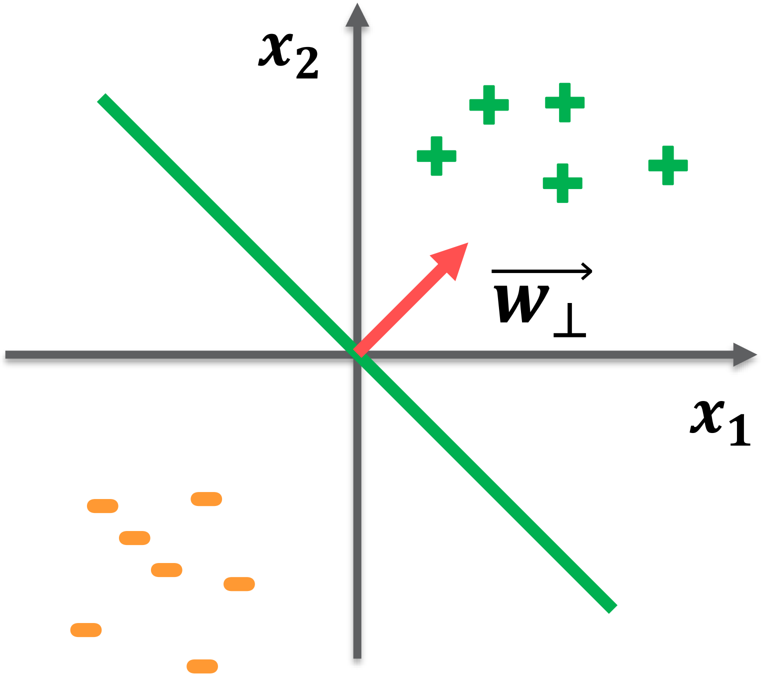


Correct Clusters

Kernels

UNDERLYING MATH CONCEPTS

Preliminaries: Linear Decision Boundary



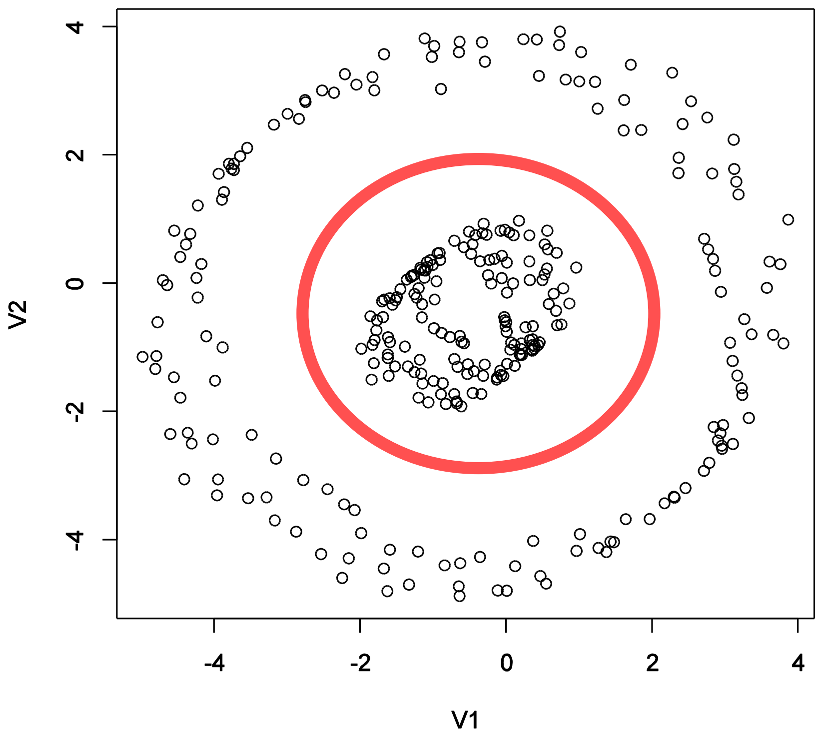
1. $<\vec{u}\cdot\vec{v}> = \vec{u}\cdot\vec{v} = (\vec{u},\vec{v})$ is a similarity measure between points/vectors (scalar product, inner product, dot product)

2. $\overrightarrow{w_\perp}\cdot\vec{x} + \mathrm{b} = 0$ – equation of a hyper-plane w/ the normal (perpendicular) vector $\overrightarrow{w_\perp}$

3. $\overrightarrow{w_\perp}\cdot\vec{x} + \mathrm{b} \geq 0$ – positive hyper-space
   $\overrightarrow{w_\perp}\cdot\vec{x} + \mathrm{b} < 0$ – negative hyper-space

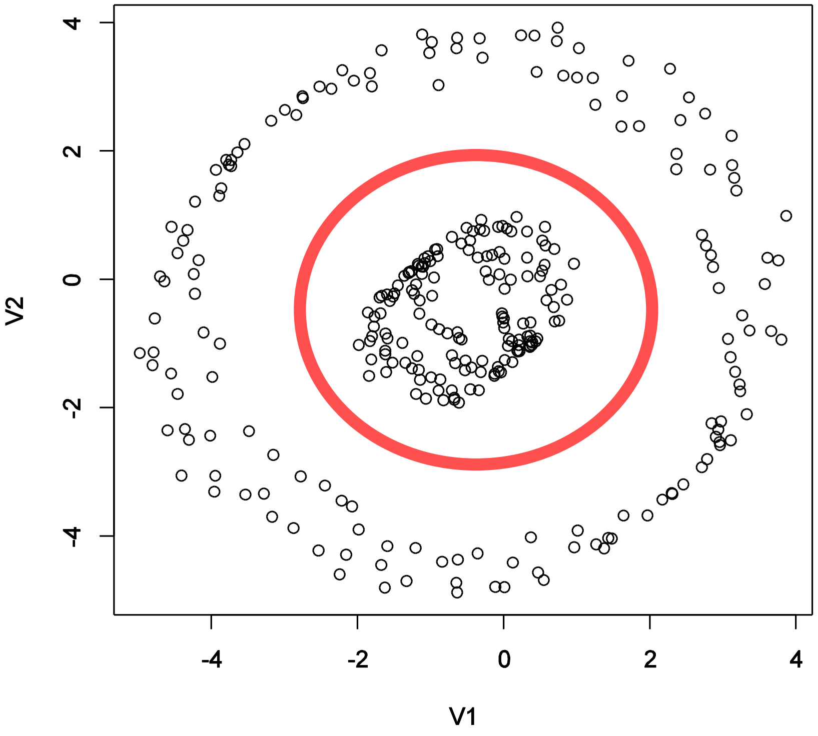
Thus, if a hyper-plane separates (linearly) different classes of points, then ($\overrightarrow{w_\perp}$, b) is all we need to know to decide the class for an unknown object $\hat{x}$: $\boldsymbol{sign}(\overrightarrow{\boldsymbol{w}_\perp}\cdot\vec{\boldsymbol{x}} + \mathbf{b})$, sign = (+/-)

Non-linear Decision Boundary



Solving non-linear problems is usually much harder than solving linear optimization problems.

Explicit Mapping to Deal with Non-linearity



Imagine that we could transform such non-linearly separable data into some higher-dimensional space, where transformed data becomes linearly separated:

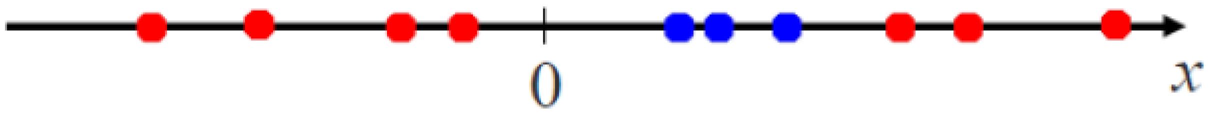
- reduce a non-linear problem in the original space to the linear problem in a higher-dimensional, transformed space.

Why transform?

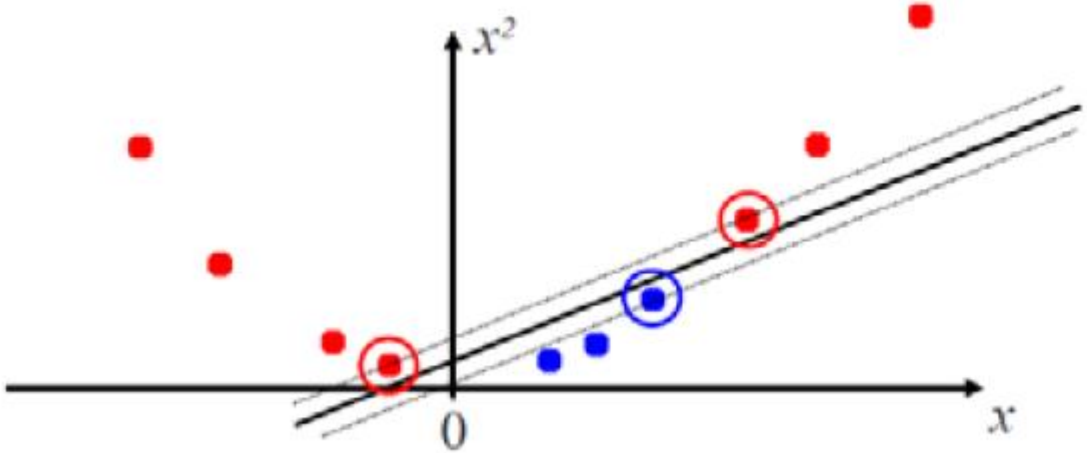
- Linear operation in the feature space is equivalent to non-linear operation in input/original space
- Classification can become easier with a proper transformation

Linear Separability Example

- **Is this data linearly separable?**



- **How about a quadratic mapping: $\Phi(x^2)$**



Ex #1: Explicit Mapping

Example: $\Phi: \mathbb{R}^2 \rightarrow \mathbb{R}^3$

$$p = (x_1, x_2) \in \mathbb{R}^2 \rightarrow p_{new} = \left(x_1^2, \sqrt{2}x_1x_2, x_2^2\right) \in \mathbb{R}^3$$

$$u = (1, -2)$$

$$v = (2, 3)$$

inner / dot / scalar product

$$(u, v) =$$
$$= 1 * 2 + (-2) * 3$$
$$= -4$$

$$u_{new} = \left(1, -2\sqrt{2}, 4\right) \in \mathbb{R}^3$$

$$v_{new} = \left(4, 6\sqrt{2}, 9\right) \in \mathbb{R}^3$$

inner / dot / scalar product

$$(u_{new}, v_{new}) =$$
$$= 1 * 4 + (-12) * 2 + 4 * 9$$
$$= 4 - 24 + 36 = 16$$
$$= \ (u, v)^2$$

Example: Explicit Mapping

Example: $\boldsymbol{\Phi}: \mathbb{R}^2 \rightarrow \mathbb{R}^3$

$$\boldsymbol{p} = (\boldsymbol{x_1}, \boldsymbol{x_2}) \in \mathbb{R}^2 \rightarrow \boldsymbol{p_{new}} = \left(\boldsymbol{x_1^2}, \sqrt{\mathbf{2}}\boldsymbol{x_1 x_2}, \boldsymbol{x_2^2}\right) \in \mathbb{R}^3$$

New coordinates are explicitly defined through original coordinates:

$$\overrightarrow{\boldsymbol{\Phi}(\boldsymbol{u})} = \boldsymbol{u_{new}} = (\boldsymbol{u_1^2}, \sqrt{\mathbf{2}}\boldsymbol{u_1 u_2}, \boldsymbol{u_2^2})$$
$$\overrightarrow{\boldsymbol{\Phi}(\boldsymbol{v})} = \boldsymbol{v_{new}} = (\boldsymbol{v_1^2}, \sqrt{\mathbf{2}}\boldsymbol{v_1 v_2}, \boldsymbol{v_2^2})$$

Since $(\vec{u}, \vec{v})$, the inner product, is a similarity between points in original space:

- **what happens with the similarity between newly transformed points**:

$$\left(\overrightarrow{\boldsymbol{\Phi}(\boldsymbol{u})}, \overrightarrow{\boldsymbol{\Phi}(\boldsymbol{v})}\right) = (\boldsymbol{u_{new}}, \boldsymbol{v_{new}}) =$$
$$\boldsymbol{u_1^2 v_1^2} + \mathbf{2}\boldsymbol{u_1 u_2 v_1 v_2} + \boldsymbol{v_1^2 v_2^2} =$$
$$(\boldsymbol{u_1 v_1} + \boldsymbol{u_2 v_2})^2 = < \vec{\boldsymbol{u}} \cdot \vec{\boldsymbol{v}} >^2$$

Kernel Function = F (scalar products)

- **Kernel function in terms of:**
  - the original inner/scalar/dot products
  - without explicit knowledge of new coordinates!

E.g.: $K(u, v) = (u, v)^2$

- **No need to know the coordinates of vectors *u* and *v***
  - only their similarity in terms of inner product=scalar product = dot product.

- **It is very handy when**
  - the objects are strings, graphs, images, texts
  - for which it is easier to define similarity metric
  - rather than coordinates of their vector representation

Implicit Mapping (Kernel) vs. Explicit Mapping

Explicit mapping (not desirable):

$\boldsymbol{\Phi}: \mathbb{R}^{\boldsymbol{d}} \rightarrow \mathbb{R}^{\boldsymbol{m}} = \boldsymbol{H}(\boldsymbol{m} > \boldsymbol{d})$ – feature space, Hilbert space (generalization of a Euclidian space), $m = \infty$ possible such that:

$$\forall\, \vec{\boldsymbol{u}}, \vec{\boldsymbol{v}} \in \mathbb{R}^{\boldsymbol{d}}\colon (\vec{\boldsymbol{u}}, \vec{\boldsymbol{v}}) \approx (\overrightarrow{\boldsymbol{\Phi}(\boldsymbol{u})}, \boldsymbol{\Phi}(\vec{\boldsymbol{v}}))$$

inner products/similarity in transformed space are similar to inner products in original space

Implicit representation:

$$K(u, v) = \big(\boldsymbol{\Phi}(u), \boldsymbol{\Phi}(v)\big) = F((u, v))$$

is the inner product in a Hilbert space, H, i.e.,
no need to know coordinates: $\Phi(u) \in \mathbb{R}^m = H$

Examples: Kernel Functions

$K(u,v) = (u,v)^2$ - vanilladot kernel

$K(u,v) = (u,v)^2$ or

$K(u,v) = (scale * (u,v) + offset)^{degree}$ - polynomial kernel

$K(u,v) = e^{\frac{|u-v|^2}{\sigma}}$ - Radial Basis Function **(RBF)**

($\mathbb{R}^m = H$, $m = \infty$ infinite Hilbert space)

Inner/dot/scalar product

Euclidean distance:

$$\boxed{|u-v|^2 = (u-v, u-v) = (u,u) - 2(u,v) + (v,v)}$$

Formal Definition of a Kernel

$\boldsymbol{K(u,v)}$ **is a kernel if**

(1) Symmetric: $K(u,v) = K(v,u)$ (*)

(2) Positive semi-definite:

$\forall\, c_1, c_2, \dots, c_n \in \mathbb{R}$ **and a set of points**

$\forall\, u_1, u_2, \dots, u_n \in \mathbb{R}^d$

$\sum_{i=1}^{n} \sum_{j=1}^{n} c_i c_j K(u_i, u_j) \geq 0$ (**)

- valid similarity measure

In terms of matrix notation (**) can be written as:

$C^T \cdot M_K \cdot C \geq 0$, where

kernel matrix

$$C = \begin{bmatrix} c_1 \\ c_2 \\ \dots \\ c_n \end{bmatrix} \text{ and } \quad M_K = \begin{bmatrix} K(u_1,u_1) & K(u_1,u_2) & \dots & K(u_1,u_n) \\ K(u_2,u_1) & K(u_2,u_2) & \dots & K(u_2,u_n) \\ \dots & \dots & & \dots \\ K(u_n,u_1) & K(u_n,u_2) & \dots & K(u_n,u_n) \end{bmatrix}$$

Euclidean Distance as Dot Product

Let $X = [x_1, x_2, \dots, x_n] \in \mathbb{R}^{n \times d}$ be a set of points in **d**-dimensional space.

It is a **vector** data. Let (x_i, x_j) be the dot product between two vectors.
Note: that dot (inner/scalar) product is a similarity measure

Distance (Euclidean) between x_i and x_j can be calculated in terms of dot/inner products:

$$\boxed{\begin{aligned} d^2(\boldsymbol{x}_i, \boldsymbol{x}_j) &= \left|\left|\boldsymbol{x}_i - \boldsymbol{x}_j\right|\right|^2 = (\boldsymbol{x}_i - \boldsymbol{x}_j, \boldsymbol{x}_i - \boldsymbol{x}_j) \\ &= (\boldsymbol{x}_i, \boldsymbol{x}_i) - 2(\boldsymbol{x}_i, \boldsymbol{x}_j) + (\boldsymbol{x}_j, \boldsymbol{x}_j) \end{aligned}}$$

Kernel Trick and Kernelization

Distance (Euclidian) between x_i and x_j:

$$\boxed{d^2(x_i, x_j) = (x_i, x_i) - 2(x_i, x_j) + (x_j, x_j)}$$

What happens if we replace dot product by a kernel evaluation?

$x_i \to \Phi(x_i)$ and $x_j \to \Phi(x_j)$ in Hilbert space (Generalized Euclidean space)

$$\boxed{d^2\left(\Phi(x_i), \Phi(x_j)\right) = K(x_i, x_i) - 2K(x_i, x_j) + K(x_j, x_j)}$$

$$d^2\left(\Phi(x_i), \Phi(x_j)\right) = \left|\left|\Phi(x_i) - \Phi(x_j)\right|\right|^2 =$$

$$= \left(\Phi(x_i) - \Phi(x_j), \Phi(x_i) - \Phi(x_j)\right) =$$

$$= \left(\Phi(x_i), \Phi(x_i)\right) - 2\left(\Phi(x_i), \Phi(x_j)\right) + \left(\Phi(x_j), \Phi(x_j)\right) =$$

$$= K(x_i, x_i) - 2K(x_i, x_j) + K(x_j, x_j)$$

$$d^2(x_i, x_j) = \left|\left|x_i - x_j\right|\right|^2 = (x_i - x_j, x_i - x_j)$$

$$= (x_i, x_i) - 2(x_i, x_j) + (x_j, x_j)$$

Ex #2: Euclidean Distance as Dot Products

Example: $\boldsymbol{\Phi}: \mathbb{R}^2 \rightarrow \mathbb{R}^3$

$$\boldsymbol{p} = (\boldsymbol{x_1}, \boldsymbol{x_2}) \in \mathbb{R}^2 \rightarrow \boldsymbol{p_{new}} = \left(\boldsymbol{x_1^2}, \sqrt{\boldsymbol{2}}\boldsymbol{x_1}\boldsymbol{x_2}, \boldsymbol{x_2^2}\right) \in \mathbb{R}^3$$

$$\boldsymbol{u} = (\boldsymbol{1}, -\boldsymbol{2})$$

$$\boldsymbol{v} = (\boldsymbol{2}, \boldsymbol{3})$$

$$\boldsymbol{d^2}(\boldsymbol{u}, \boldsymbol{v}) =$$

$$\boldsymbol{u_{new}} = \left(\boldsymbol{1}, -\boldsymbol{2}\sqrt{\boldsymbol{2}}, \boldsymbol{4}\right) \in \mathbb{R}^3$$

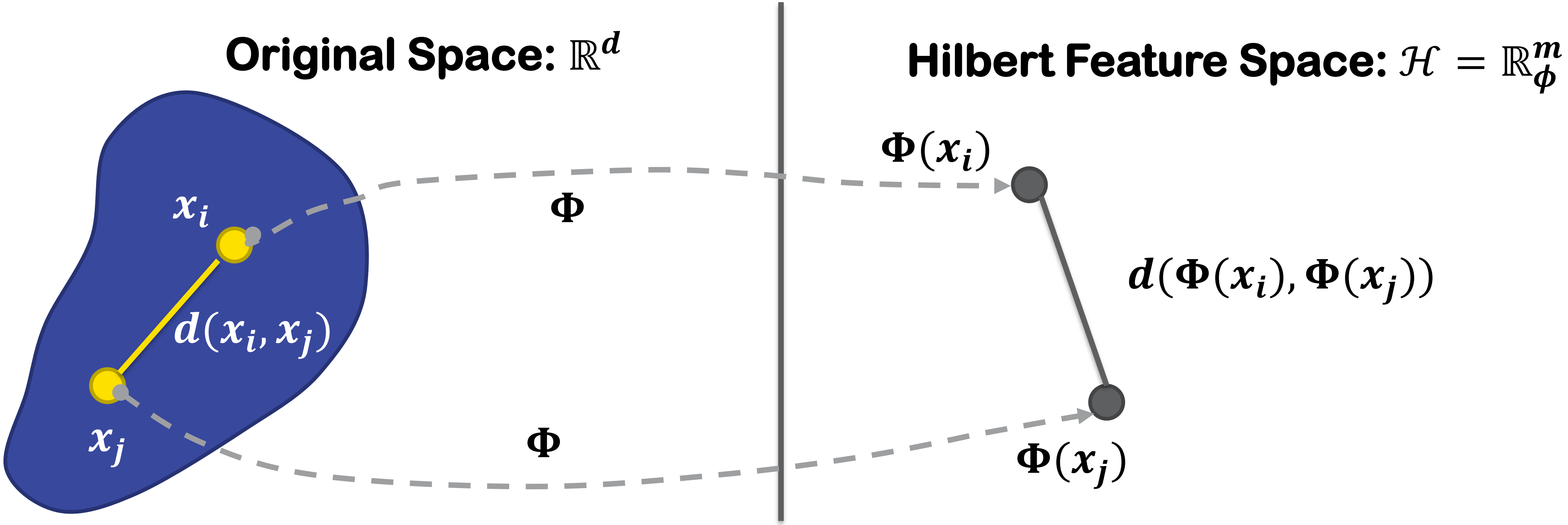
$$\boldsymbol{v_{new}} = \left(\boldsymbol{4}, \boldsymbol{6}\sqrt{\boldsymbol{2}}, \boldsymbol{9}\right) \in \mathbb{R}^3$$

$$\boldsymbol{d^2}(\boldsymbol{u_{new}}, \boldsymbol{v_{new}}) =$$

Euclidean Distance in H

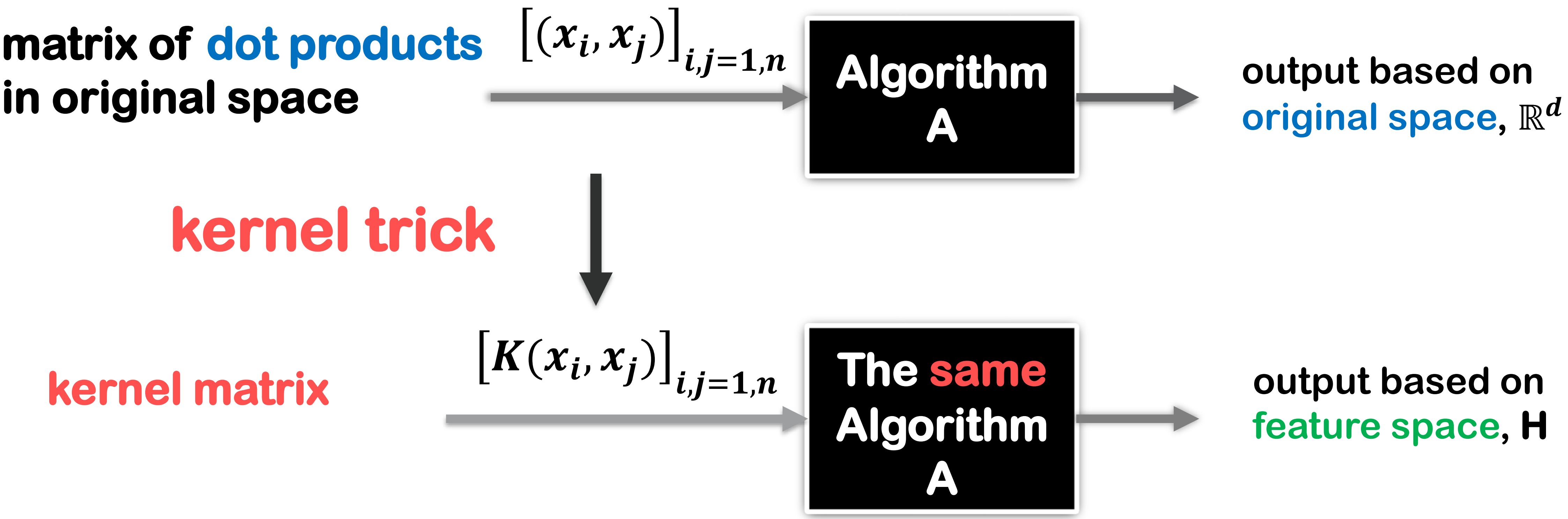
$$d^2\left(\Phi(x_i), \Phi(x_j)\right) = K(x_i, x_i) - 2K(x_i, x_j) + K(x_j, x_j)$$

b/s by def. of a kernel $K(u, v) = (\Phi(u), \Phi(v))$



Kernelization

If $K(\boldsymbol{u},\boldsymbol{v}) = \boldsymbol{F}((\boldsymbol{u},\boldsymbol{v}))$ (e.g., $K(u,v) = (u,v)^2$),
then we can compute distances (similarities) in the transformed (Hilbert) space
(a) only using dot products of original data
(b) without any explicit knowledge of vector coordinates in the feature space H



Kernel Trick and Kernelization

- **Thus, the kernel trick will**
  - replace dot products over vector data in original space $\mathbb{R}^d$
  - with their kernel evaluations $(K(u, v) = \boldsymbol{F}((u, v)))$
    - expressed as symmetric, semi-definite F() over (u, v)
- **and will apply the same algorithm to the kernel matrix**

- **Basically, the kernel trick will**
  - generalize linear methods (e.g., PCA) to non-linear methods (e.g., kernel PCA)
  - by simply replacing classic dot products by a kernel
- **This transformation is called kernelization**

Examples: Kernel Tricks

dot products

$[(x_i, x_j)]_{i,j}$

Linear method (e.g. PCA)

$[K(x_i, x_j)]_{i,j}$

Non-linear method (e.g. kPCA)

Kernelization

---

vector data

$X \in \mathbb{R}^{n \times d}$

Linear method (e.g. k-means)

$[K(x_i, x_j)]_{i,j}$

Non-linear method (e.g. kernel k-means)

D_Φ distance in H

Kernelization

Kernels and Kernel Tricks

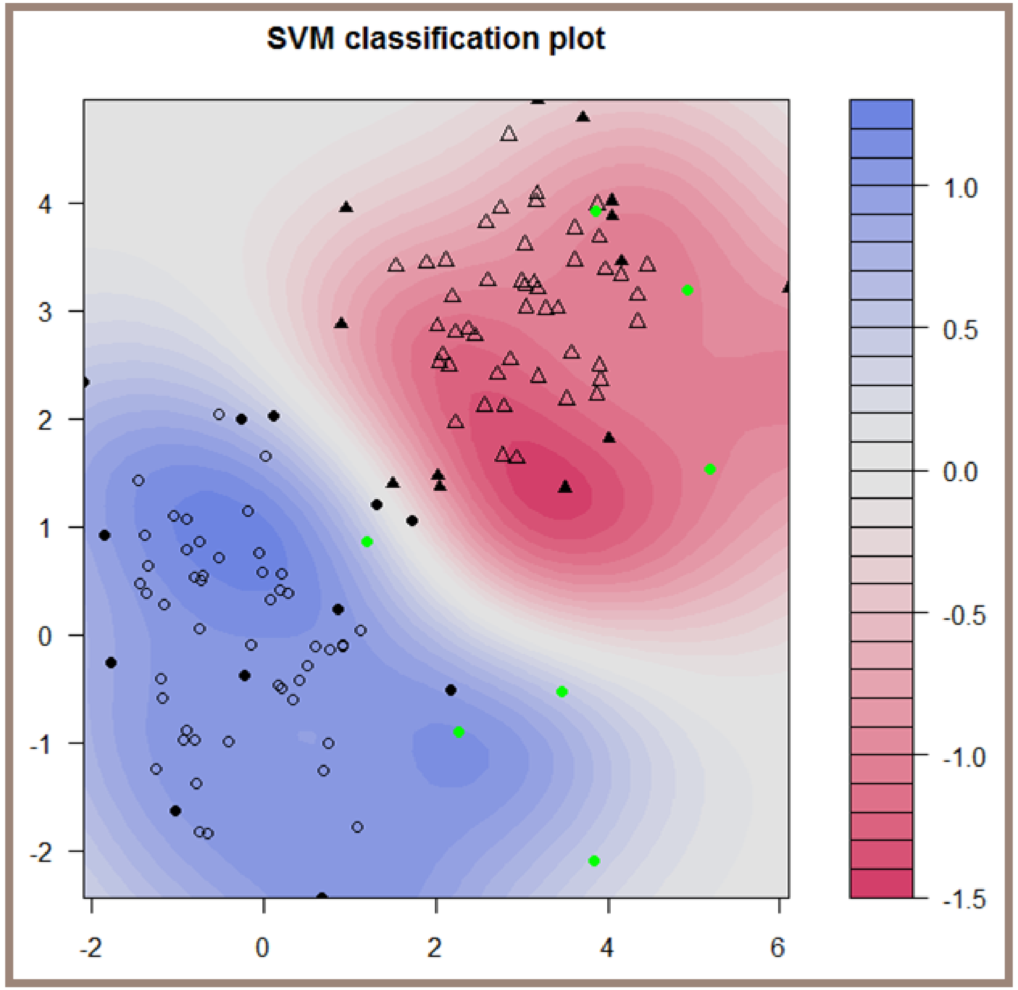
CLASSIFICATION

Classification with Kernel SVM

- **SVM Optimization problem:**
  - The data points only appear in the inner products
- **As long as we can calculate the inner product in the feature space:**
  - we do not need any explicit mapping
- **Common geometric operations:**
  - angles (cosine angle)
  - distances
- **They can be expressed by inner products**
- **Define the kernel function as the function of inner products**
  - this give the implicit mapping

$$
\begin{aligned}
\max_{\alpha} \quad & \mathcal{J}(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j (\mathbf{x}_i^T \mathbf{x}_j) \\
\text{s.t.} \quad & 0 \le \alpha_i \le C, \quad i = 1, \ldots, m \\
& \sum_{i=1}^{m} \alpha_i y_i = 0.
\end{aligned}
$$

Example: Linear Separable SVM

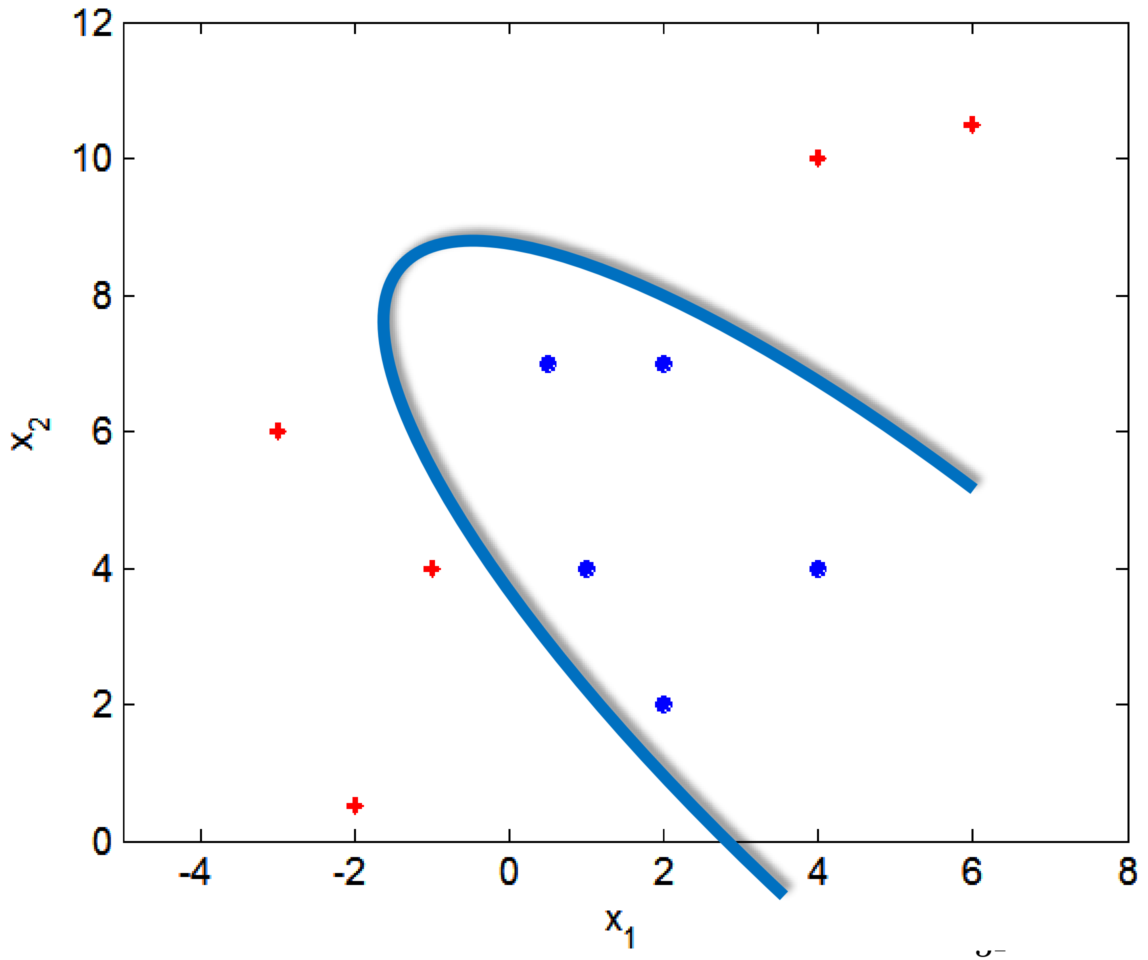


Kernels

NON-LINEAR PROBLEMS

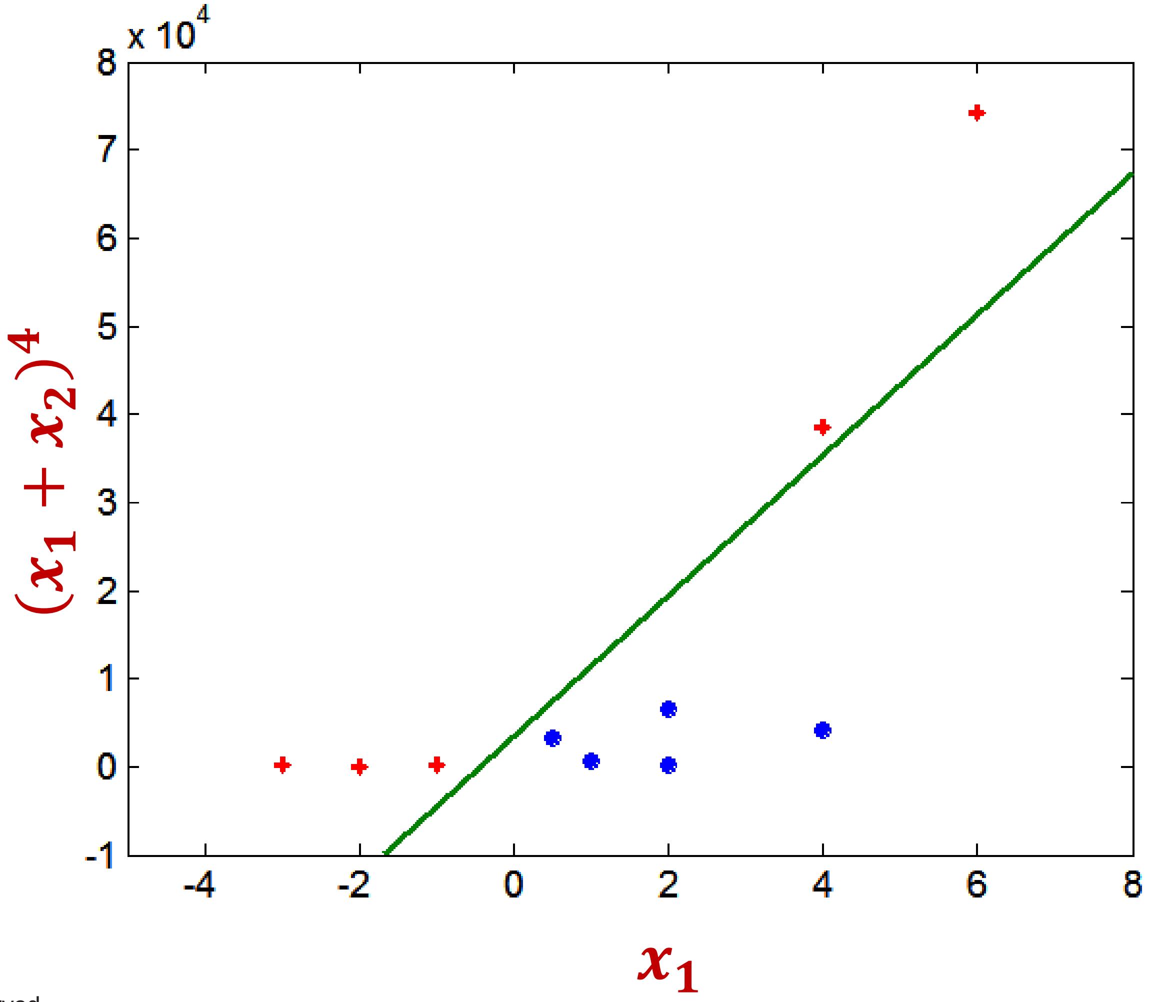
Nonlinear Support Vector Machines

- **What if the decision boundary is not-linear?**



Non-linear Support Vector Machines

Transformed data into higher dimensional space using Kernel functions



Example: Nonlinear SVM

```
1.  library ("kernlab")
2.  help (ksvm)
3.  x = rbind(matrix(rnorm(120),,2),matrix(rnorm(60,mean=3),,2),
                  matrix(rnorm(60, mean=-3), , 2))
4.  y = matrix(c(rep(1,60), rep(-1,60)))
5.  cl = c (rep("purple",60), rep("red",60))
6.  plot (x, col=cl, pch=19)

7.  # play with the cost of constraints
8.  model = ksvm (x, y, type="C-svc", C=0)
9.  model; error (model)
10. model = ksvm (x, y, type="C-svc", C=1)
11. model

12. # play with kernel functions
13. model = ksvm (x, y, type="C-svc", kernel = "polydot")
14. model
15. model = ksvm (x, y, type="C-svc", kernel = "rbfdot")
16. model
```

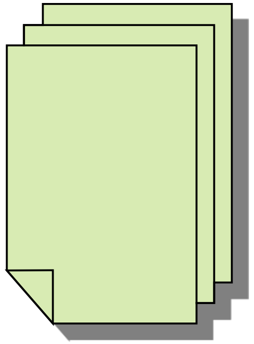
Kernels

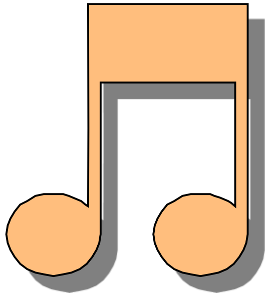
COMPLEX NON-VECTOR DATA

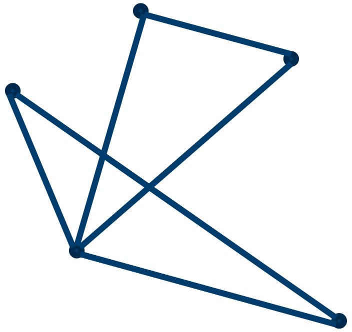
Kernels are ubiquitous

- **There exist kernels for a variety of data formats**
  - vectors
  - graphs
  - music
  - strings
  - text
  - and so on

- **There can be multiple kernels for each data modality:**
  - Choosing the right one is an art!







Summary: Kernels: Strengths & Weaknesses

- **Pros:**
  - Generalizes many algorithms that work solely in linear space (non-linear in input space □ linear in feature space)
  - Very simple:
    - does not involve non-linear optimization
    - essentially, linear algebra
- **Cons:**
  - Time-consuming for large datasets because of the kernel function computation
  - Kernel matrix is of size $n \times n$ where n is the number of observations, that becomes a quadratic cost, prohibitive for large values of n