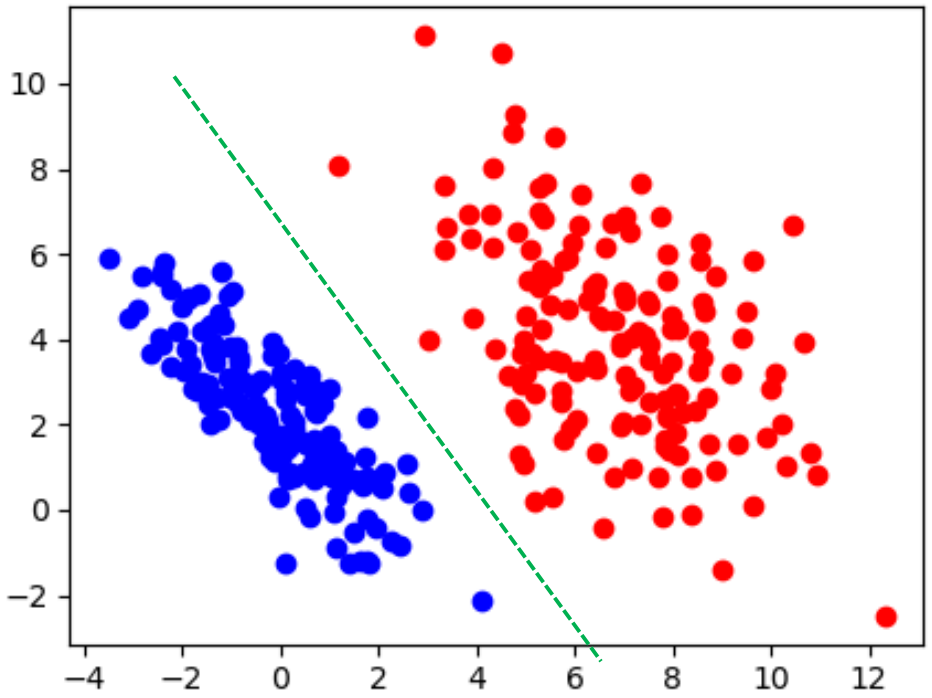
Introduction to Kernel Trick

Hanwool Jeong

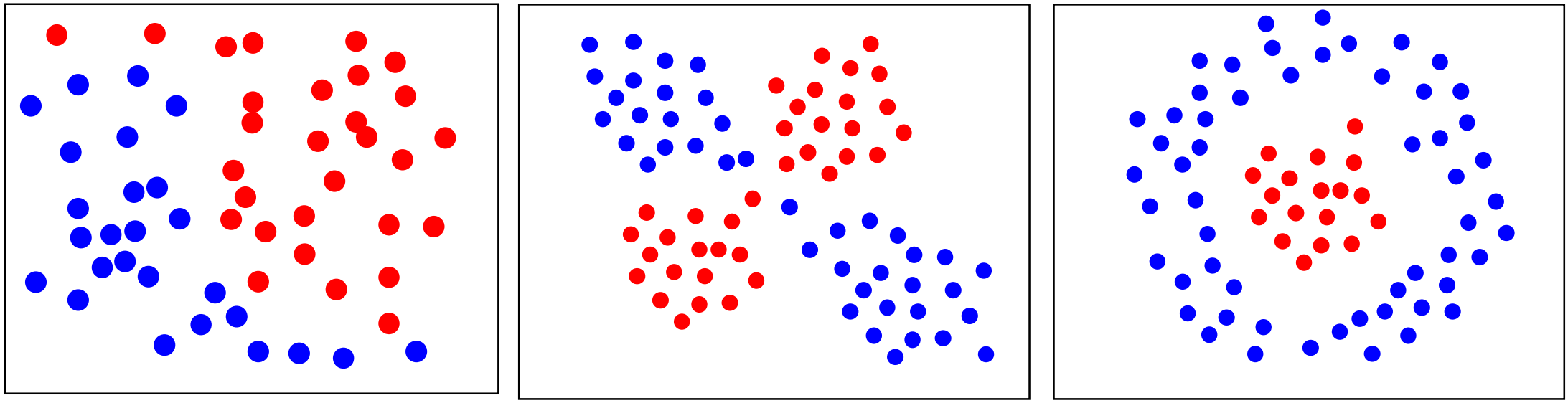
hwjeong@kw.ac.kr

Classification Data

- They are linearly separable, so we can adopt linear boundary:

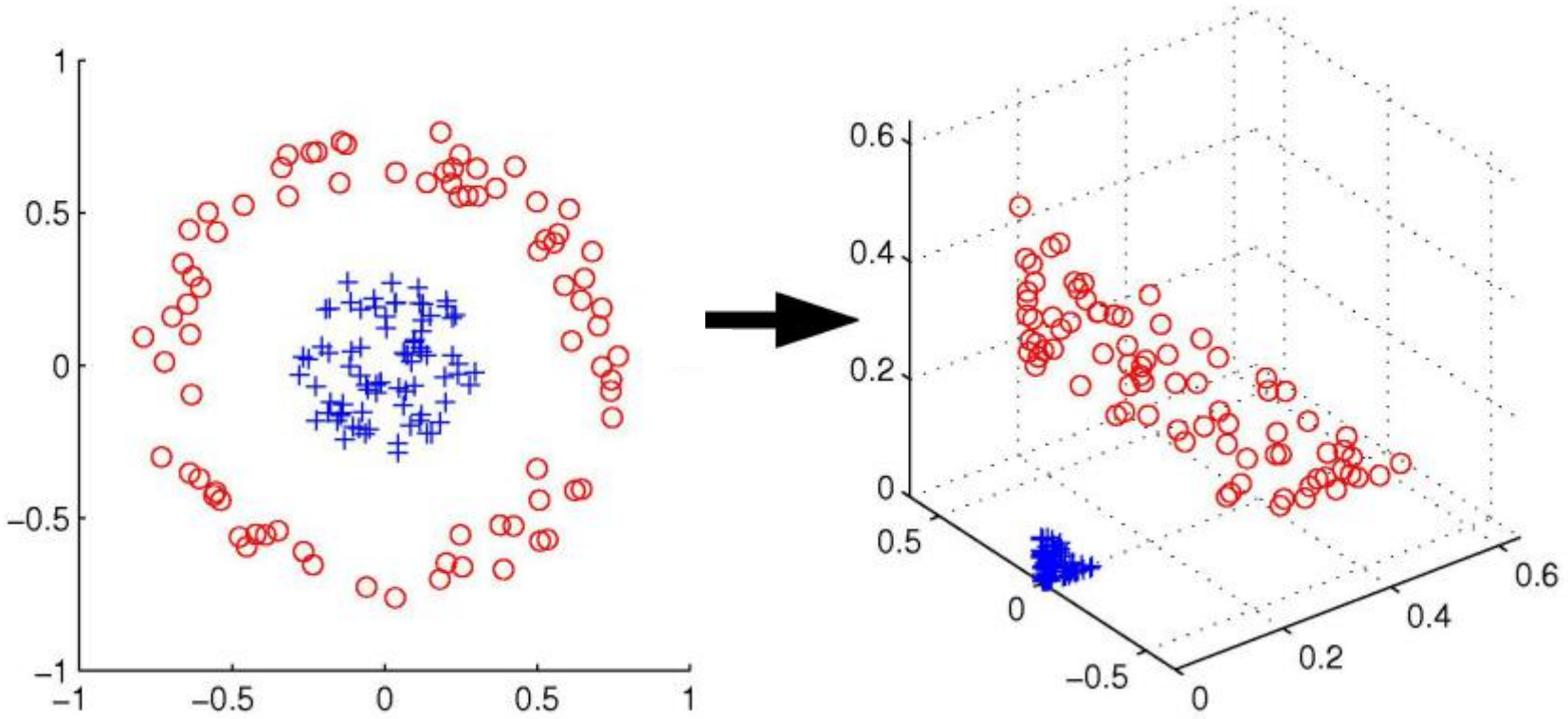


How About This Data?



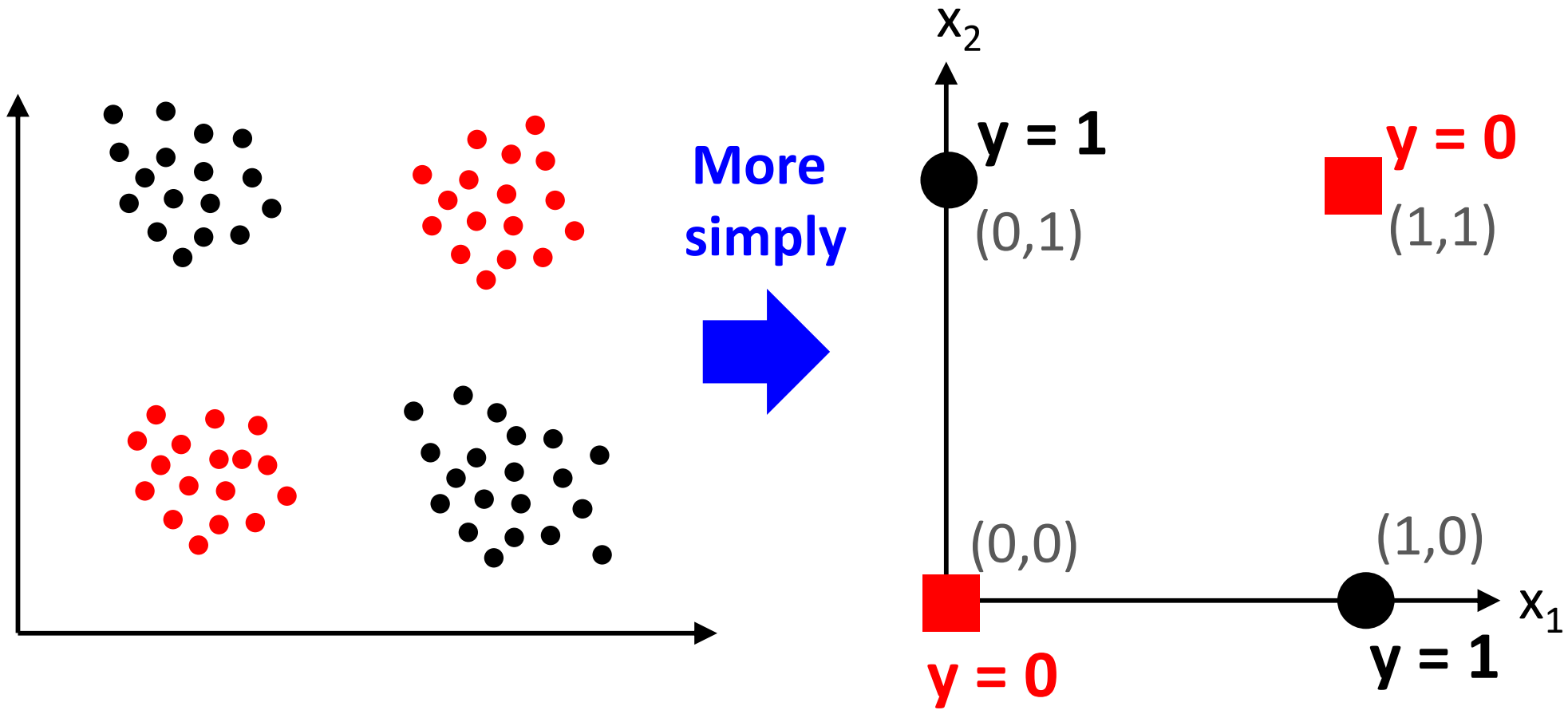
Transforming Data

- Make it linearly separable by transformation of $\Phi(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}) = \begin{bmatrix} {x_1}^2 \\ {x_2}^2 \\ \sqrt{2}x_1x_2 \end{bmatrix}$



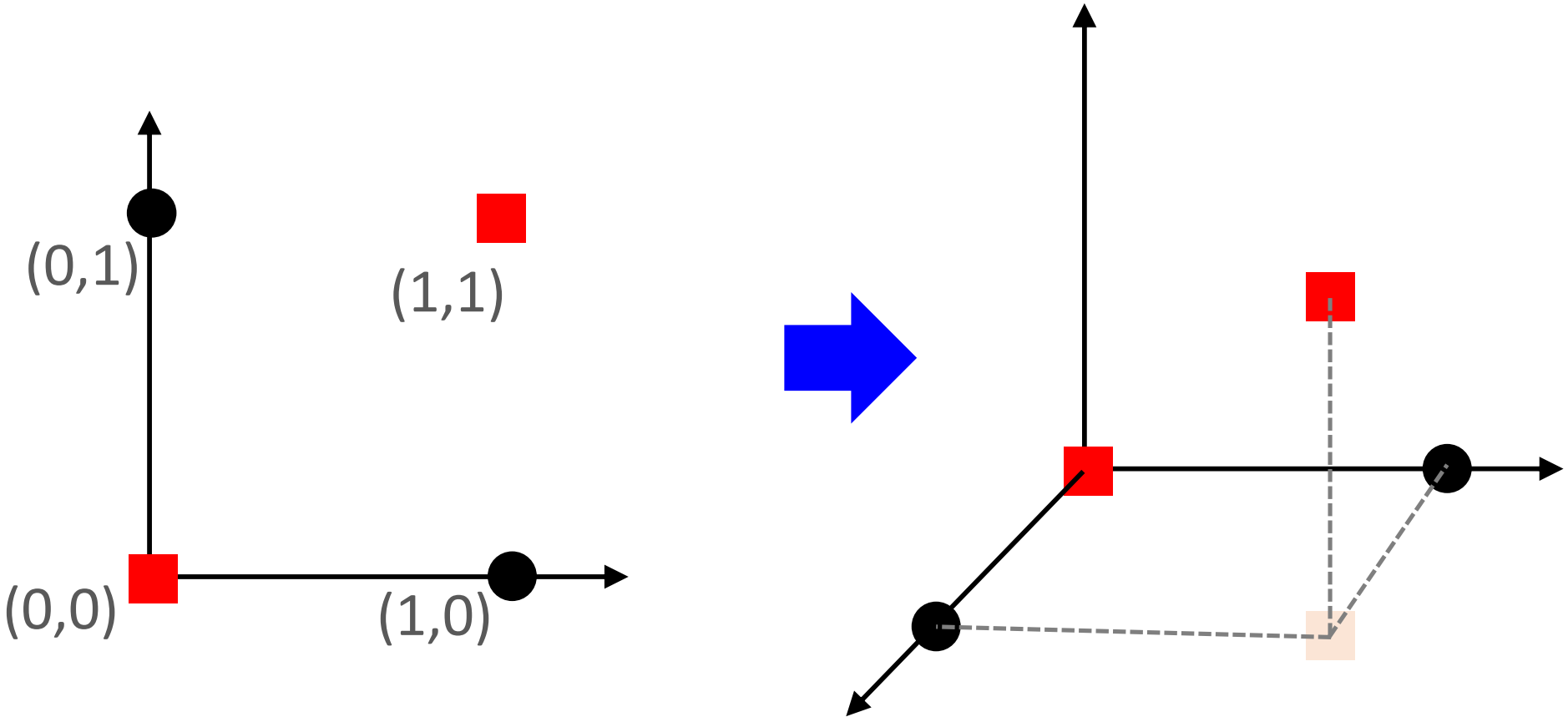
Can you Classify This?

- We call this XOR classification (Why?)
  ➔ It is not linearly distinguishable



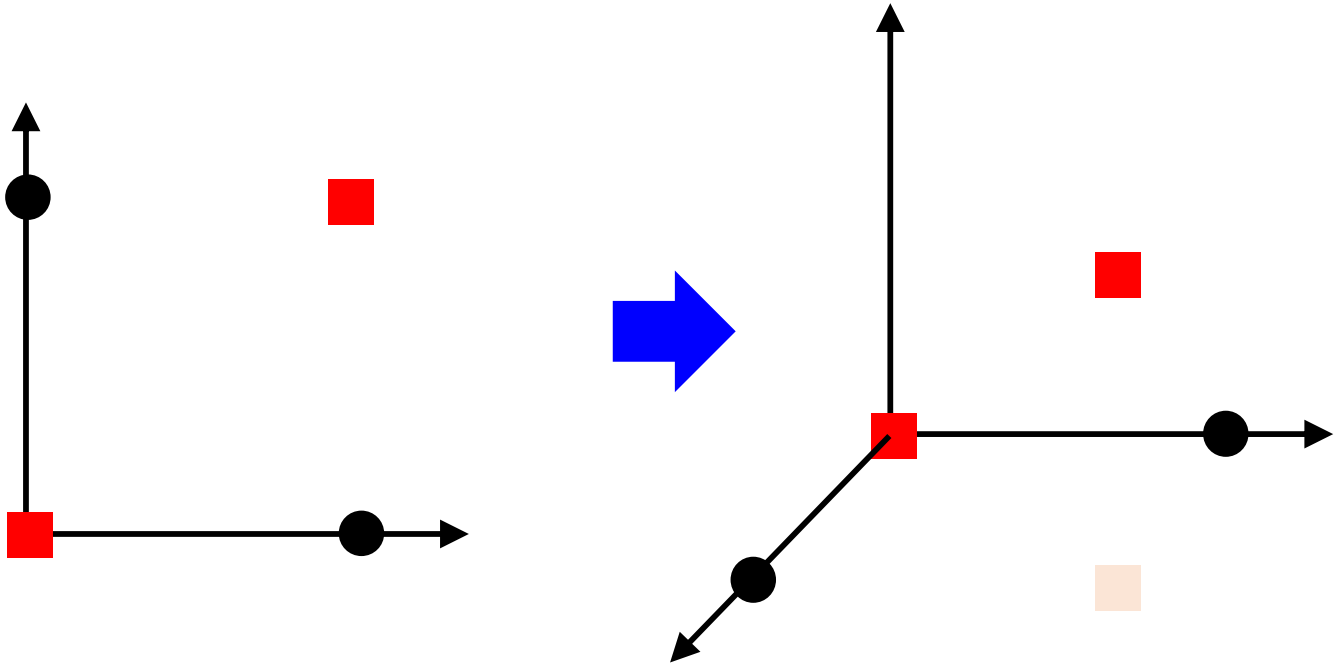
Transforming Data; Increasing Dimension

- Becomes linearly separable



Transformation Example

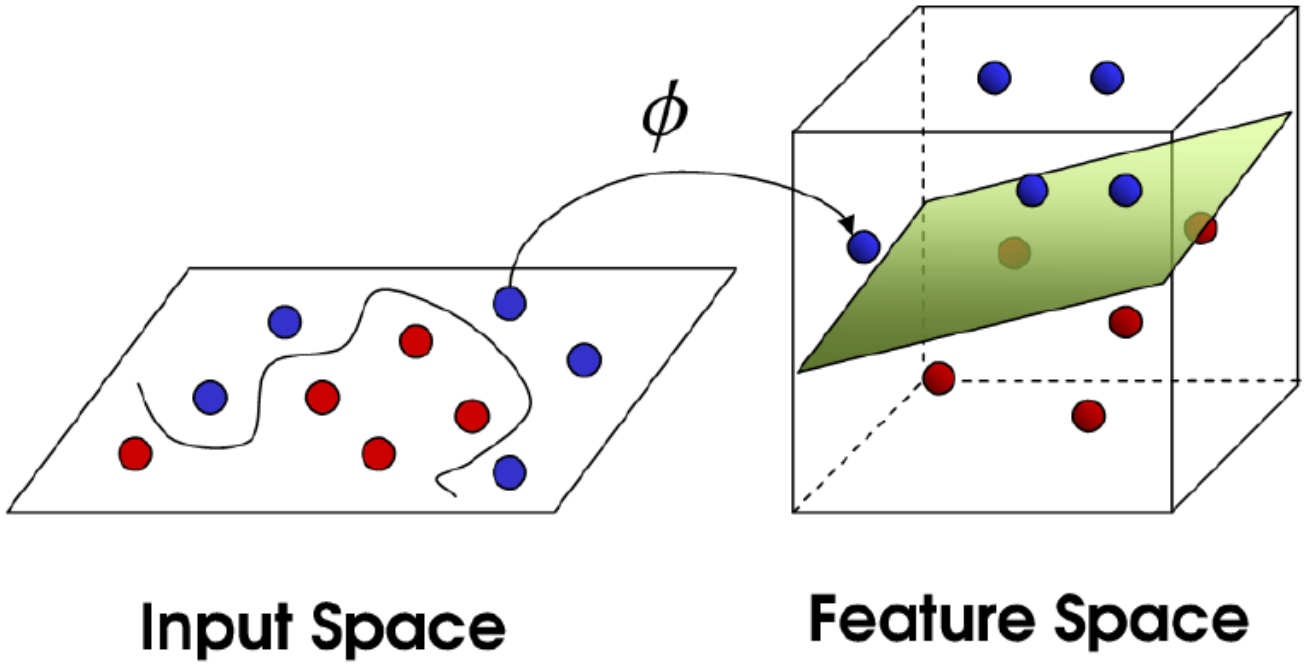
- $\Phi(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}) = \begin{bmatrix} x_1 \\ x_2 \\ x_1 x_2 \end{bmatrix}$



Linearizing Decision Boundary

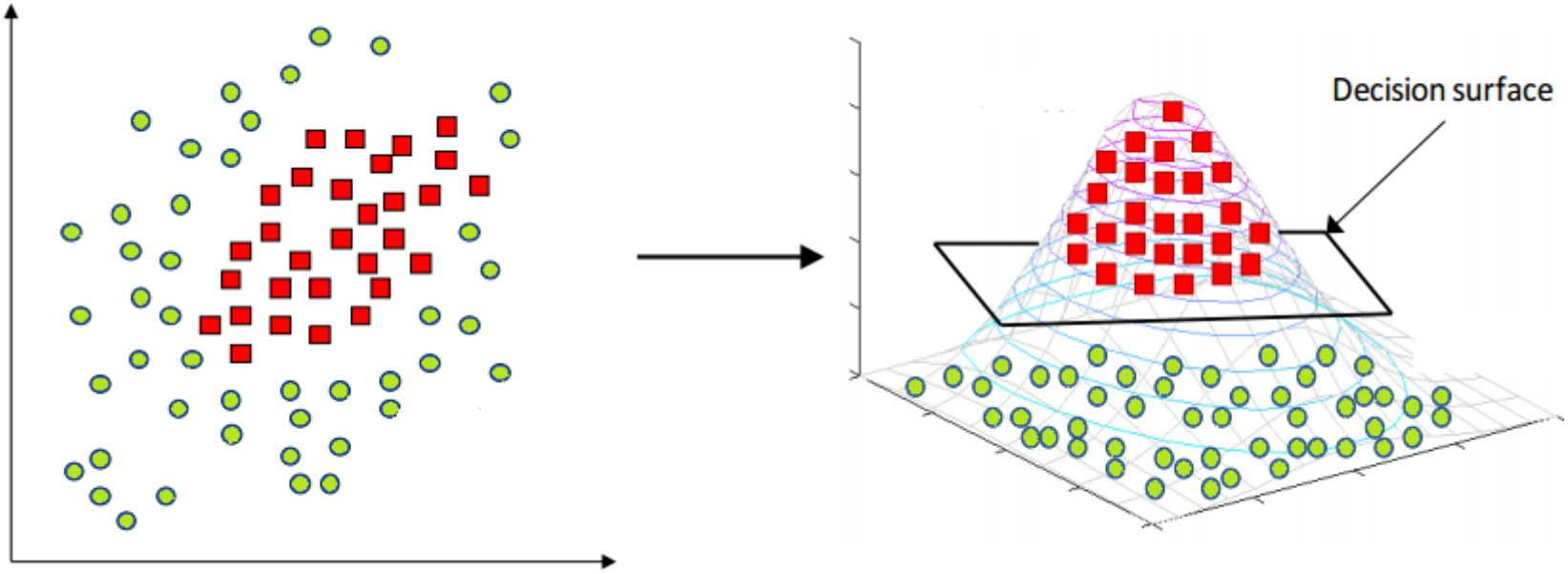
- With proper transformation Φ,

✓ **Change the environment**
✓ In changed world, we do the
1) training (e.g., MLE, optimization)
And **2) prediction**.



Or,

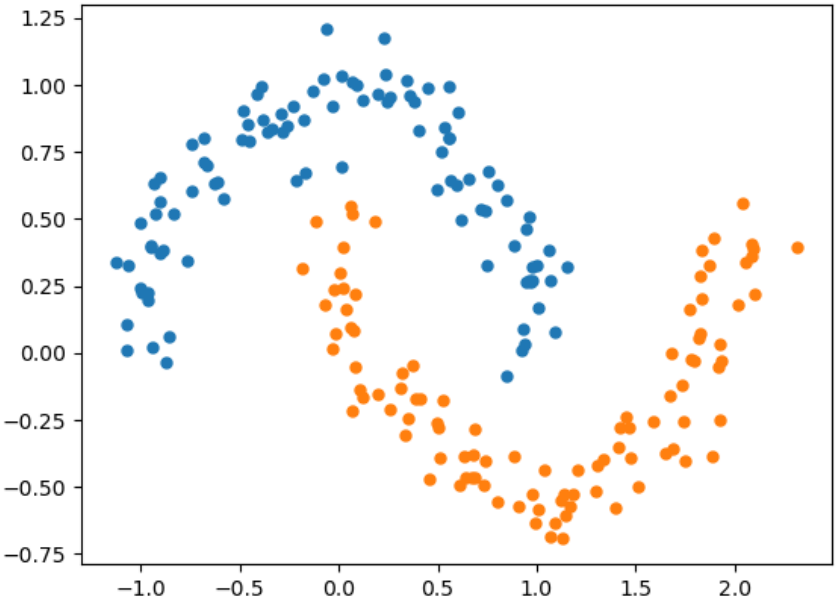
- With proper transformation Φ to D-dimensional vector, the data becomes linearly separable in "**higher**" Q dimension.
- Usually, "**much higher**" dimension. That is, Q>> D

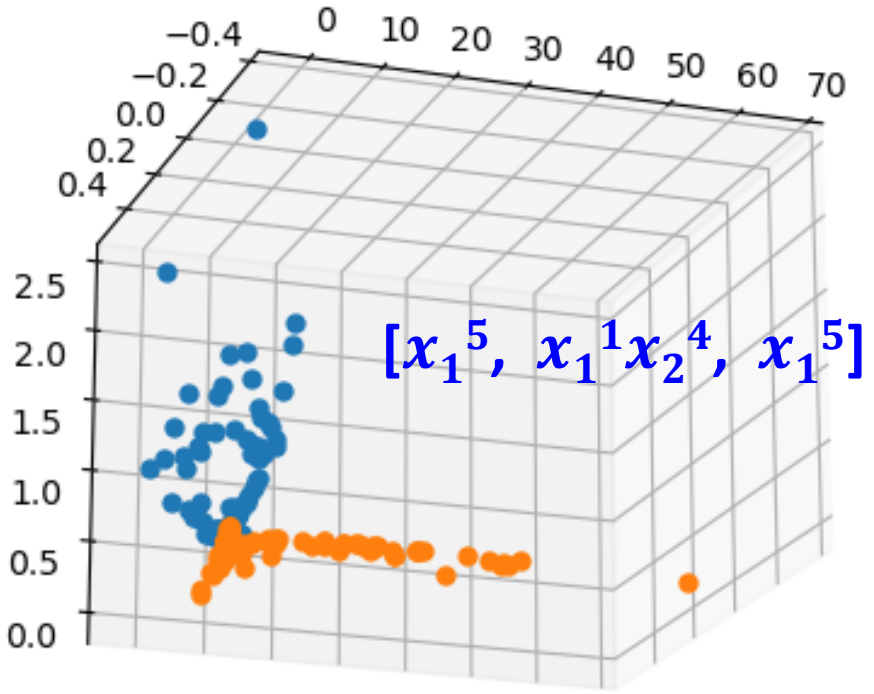


Generally, More Likely to be Linearly Separable in Higher Dimension

- w/o Thinking, $\Phi([x_1, x_2]^T) = [x_1^5, x_1^4x_2^1, x_1^3x_2^2, x_1^2x_2^3, x_1^1x_2^4, x_2^5]$

$[x_1, x_2]$





However,
Higher Dimension Increases Complexity

- Efficiency/Complexity .. ➔ What do these terms mean in ML?
- We start from that we want to use “linear boundary” for the efficiency and reducing the computational complexity.
- To stick to use “linear boundary”, we make the data separable by exploiting high dimension.
- Is it reasonable? It increases the computational cost as well.
- During the optimization (MLE) or prediction, we require a number of vector-matrix calculation.

Motivation for Kernel Trick

- As explained, Q >> D, thus the calculation of the following is highly complex, increasing computational costs.

$$\Phi(\boldsymbol{x})\bullet\Phi(\boldsymbol{x'}) = \Phi(\boldsymbol{x})^T\Phi(\boldsymbol{x'})$$

- But how about there is a function we can calculate the above in the original dimension space, R^D. That is,

$$K(\boldsymbol{x}, \boldsymbol{x'}) = \Phi(\boldsymbol{x})\bullet\Phi(\boldsymbol{x'}) = \Phi(\boldsymbol{x})^T\Phi(\boldsymbol{x'})$$

ex) $3|x+x'|^2+4$

$(x \bullet x'+3)^5$

- Then, we can calculate the inner product in R^Q (representing similarity in R^Q) in the space of R^D, with reduced complexity.
- The above K(x, x') is called the **kernel** function, and this technique is called **kernel trick**.

Example

- Given **x** and **y** in R^3,

$$\mathbf{x} = (x_1, x_2, x_3)^T$$
$$\mathbf{y} = (y_1, y_2, y_3)^T$$

- And the transformation to R^9 is defined as

$$\phi(\mathbf{x}) = (x_1^2, x_1x_2, x_1x_3, x_2x_1, x_2^2, x_2x_3, x_3x_1, x_3x_2, x_3^2)^T$$

- Inner product

$$\phi(\mathbf{x})^T\phi(\mathbf{y}) = \sum_{i,j=1}^{3} x_i x_j y_i y_j$$

- We can use kernel function as follows.

$$k(\mathbf{x}, \mathbf{y}) = (\mathbf{x}^T\mathbf{y})^2$$

Kernel Function Defined by Mercer Mercer's Theorem

- If there exists a function Φ mapping $\boldsymbol{x} \in \mathrm{R}^{\mathrm{D}}$ to R^{Q} such that

$$\kappa(\mathbf{x}, \mathbf{x}') = \boldsymbol{\phi}(\mathbf{x})^T \boldsymbol{\phi}(\mathbf{x}')$$

- Then, we call this (Mercer) kernel function
- Typically, it is symmetric and used for representing similarity, but these are not required condition.

Various Kernel Function

- Radial basis function or **RBF** kernel or Gaussian kernel

$$\kappa(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{||\mathbf{x} - \mathbf{x}'||^2}{2\sigma^2}\right)$$

- Polynomial kernel

$$\kappa(\mathbf{x}, \mathbf{x}') = (\gamma \mathbf{x}^T \mathbf{x}' + r)^M$$

- Sigmoid kernel or tanh kernel

$$\kappa(\mathbf{x}, \mathbf{x}') = \tanh(\gamma \mathbf{x}^T \mathbf{x}' + r)$$

Why RBL is Effective

- Using Taylor expansion,

$$e^{\frac{-1}{2\sigma^2}(x_i - x_j)^2} = e^{\frac{-x_i^2 - x_j^2}{2\sigma^2}} \left(1 + \frac{2x_i x_j}{1!} + \frac{(2x_i x_j)^2}{2!} + \ldots\right)$$

$$= e^{\frac{-x_i^2 - x_j^2}{2\sigma^2}} \left(1 \cdot 1 + \sqrt{\frac{2}{1!}} x_i \cdot \sqrt{\frac{2}{1!}} x_j + \sqrt{\frac{(2)^2}{2!}} (x_i)^2 \cdot \sqrt{\frac{(2)^2}{2!}} (x_j)^2 + \ldots\right)$$

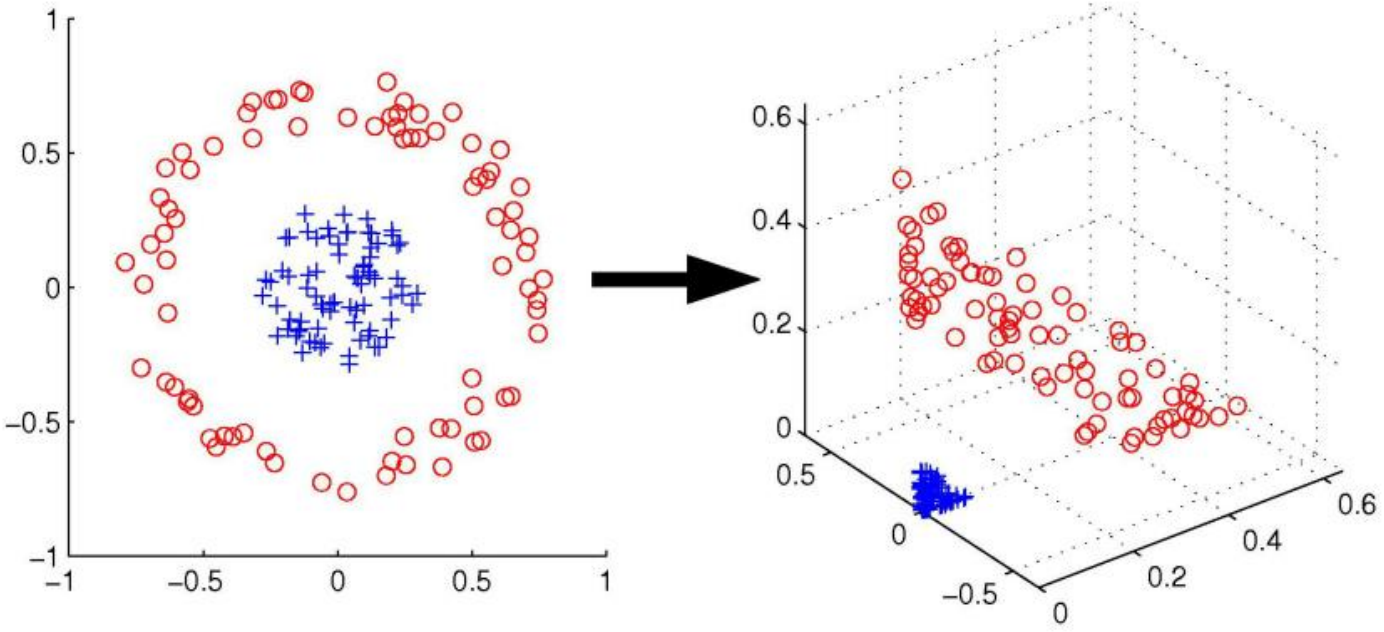
$$= \phi(x_i)^T \phi(x_j)$$

$$\text{where, } \phi(x) = e^{\frac{-x^2}{2\sigma^2}} \left(1, \sqrt{\frac{2}{1!}} x, \sqrt{\frac{2^2}{2!}} x^2, \ldots\right)$$

Revisit This!

- Now we can use k-means clustering or k-NN classification by kernelized k-means or kernelized k-NN

  Calculation complexity may not be exceedingly increased with trick!



Revisit Ridge Regression

- Linear regression

$$\hat{y} = h_w(\boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{x} \text{ : Hyperplane}$$

$$\hat{\mathbf{w}}_{OLS} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

- To prevent overfitting

$$\hat{\mathbf{w}}_{ridge} = (\lambda\mathbf{I}_D + \mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

which is the result of minimizing

$$J(\mathbf{w}) = (\mathbf{y} - \mathbf{X}\mathbf{w})^T(\mathbf{y} - \mathbf{X}\mathbf{w}) + \lambda||\mathbf{w}||^2$$

Dual Problem

- Multiplying $(X^TX + \lambda I_D)$ both side,

$$\mathbf{w} = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I}_D)^{-1}\mathbf{X}^T\mathbf{y} \rightarrow \mathbf{X^TXw} + \lambda\mathbf{w} = \mathbf{X^Ty}$$

$$\lambda\mathbf{w} = \mathbf{X^ty} - \mathbf{X^TXw} = \mathbf{X^T}(\mathbf{y} - \mathbf{Xw}) \rightarrow \mathbf{w} = \mathbf{X^T}(1/\lambda)\,(\mathbf{y} - \mathbf{Xw})$$

- Let the latter part $(1/\lambda)\,(\mathbf{y} - \mathbf{Xw}) = \boldsymbol{\alpha}$,

$$(\mathbf{y} - \mathbf{Xw}) = \lambda\boldsymbol{\alpha} \rightarrow \mathbf{y} - \mathbf{XX^T}\boldsymbol{\alpha} = \boldsymbol{\lambda\alpha} \rightarrow \mathbf{y} = (\lambda\mathbf{I_N} + \mathbf{XX^T})\boldsymbol{\alpha}$$

$$\boldsymbol{\alpha} = (\boldsymbol{\lambda}\mathbf{I}_N + \mathbf{XX^T})^{-1}\mathbf{y}$$

- Defining $XX^T = G$, gram matrix, which consists of only $\mathbf{x}_i \cdot \mathbf{x}_j$

$$\boldsymbol{\alpha} = (\lambda\mathbf{I_N} + \mathbf{G})^{-1}\mathbf{y} \rightarrow \mathbf{w} = \mathbf{X^T}\boldsymbol{\alpha} = \mathbf{X^T}(\lambda\mathbf{I_N} + \mathbf{G})^{-1}\mathbf{y}$$

- Compare the two terms for **w**

$$w = (\mathbf{X^TX} + \lambda\mathbf{I}_D)^{-1}X^Ty = \mathbf{X^T}(\lambda\mathbf{I}_N + \mathbf{G})^{-1}\mathbf{y}$$

Kernel Trick for Ridge Regression

- Compare the two terms for $\mathbf{w}$

$$\mathbf{w} = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I}_D)^{-1}\mathbf{X}^T\mathbf{y} \qquad \text{vs.} \qquad \mathbf{w} = \mathbf{X}^T(\lambda\mathbf{I}_N + \mathbf{G})^{-1}\mathbf{y}$$

- Imagine that the kernel trick is applied for MLE.
- **Note that when G is derived, we do not need to know mapping function Φ, but only the kernel function.**

General Step for Applying Kernel Trick

- Suppose that we do not want to linear model for various ML algorithms.
- However, the data does not show linear characteristics.
- First, we linearized the data into higher dimension.
- Then, we run the algorithm in that transformed higher dimension (training or prediction) w/ reduced calculation complexity thru kernel trick.
- Often, we do not need to know mapping function but only needs the kernel function.

Checkpoints

- ✓Why do we apply kernel trick?
- ✓What is the kernel function?
- ✓What is the general step for applying kernel function to ML algorithm?
- ✓Coming up next : Lagrange multiplier