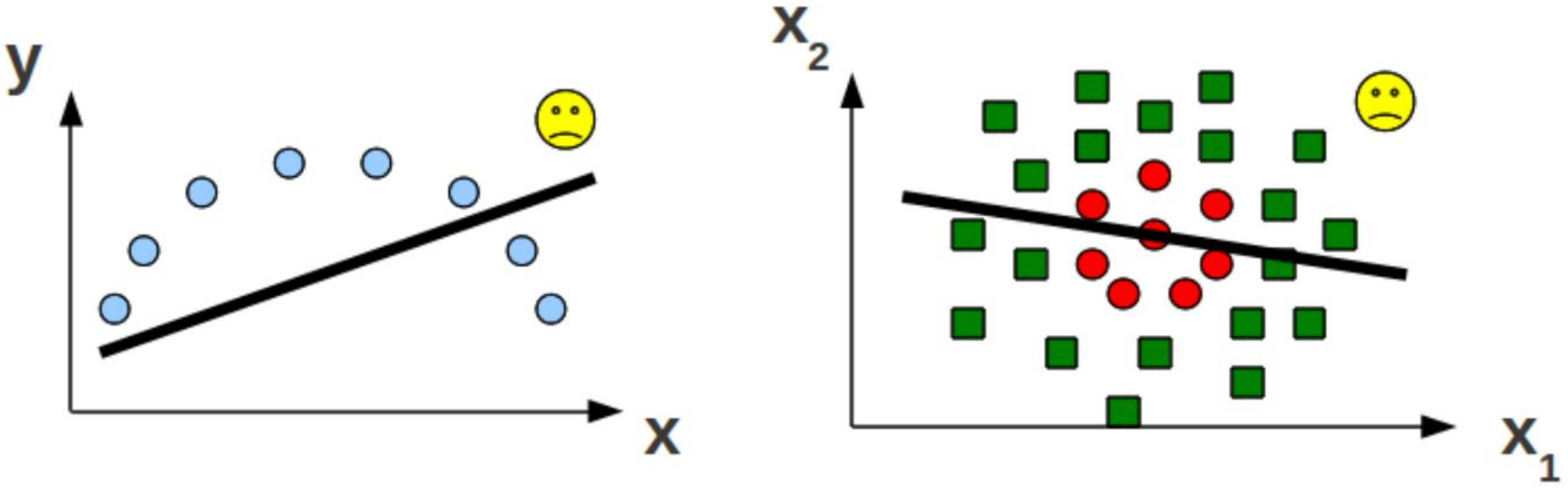
Turning Linear Models into Nonlinear Models using Kernel Methods

CS771: Introduction to Machine Learning

Piyush Rai

Linear Models

- Nice and interpretable but can't learn "difficult" nonlinear patterns



- So, are linear models useless for such problems?

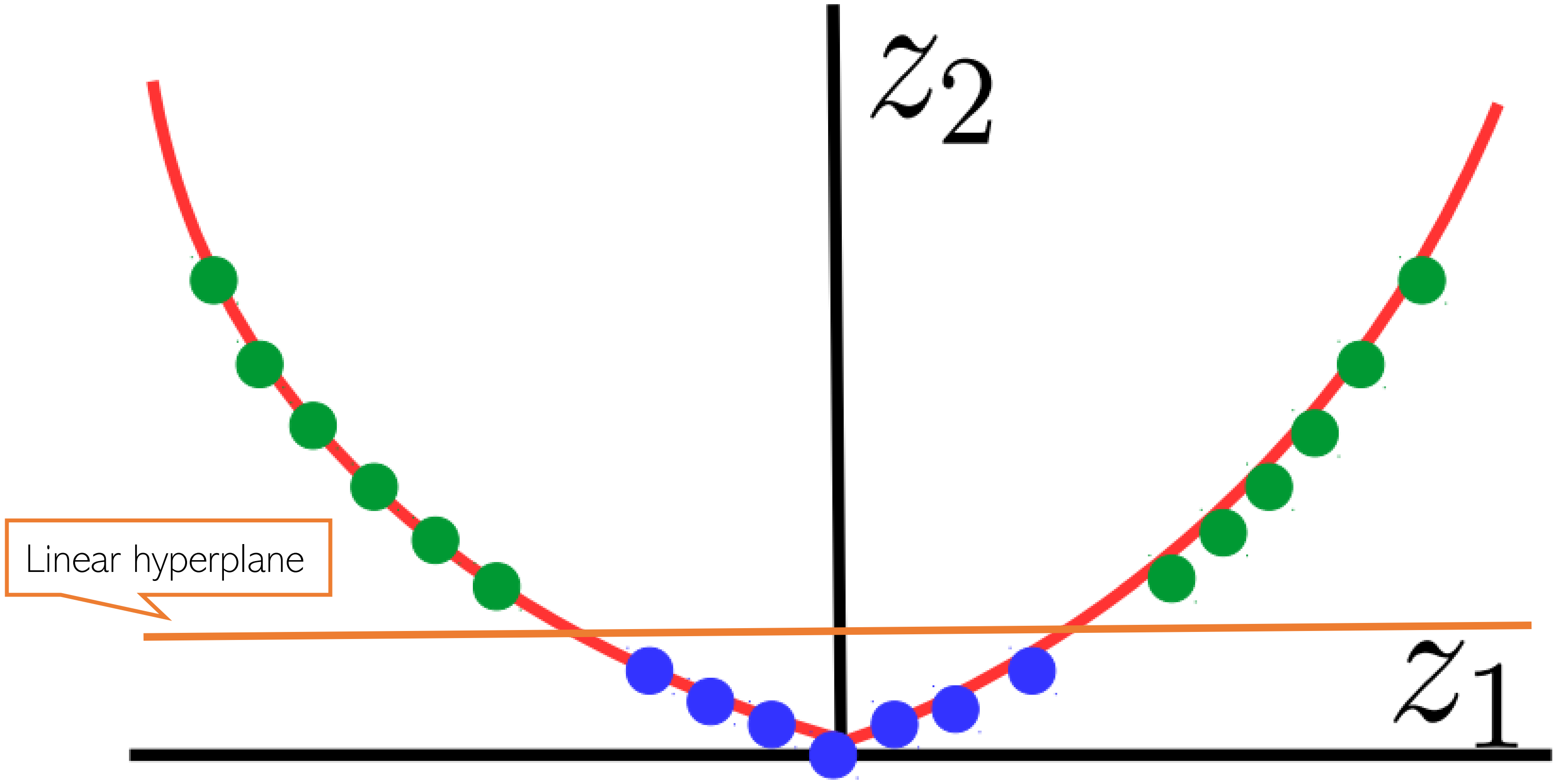
Linear Models for Nonlinear Problems

- Consider the following one-dimensional inputs from two classes

- Can't separate using a linear hyperplane

Linear Models for Nonlinear Problems

- Consider mapping each $\boldsymbol{x}$ to two-dimensions as $\boldsymbol{x} \to \boldsymbol{z} = [z_1, z_2] = [x, x^2]$



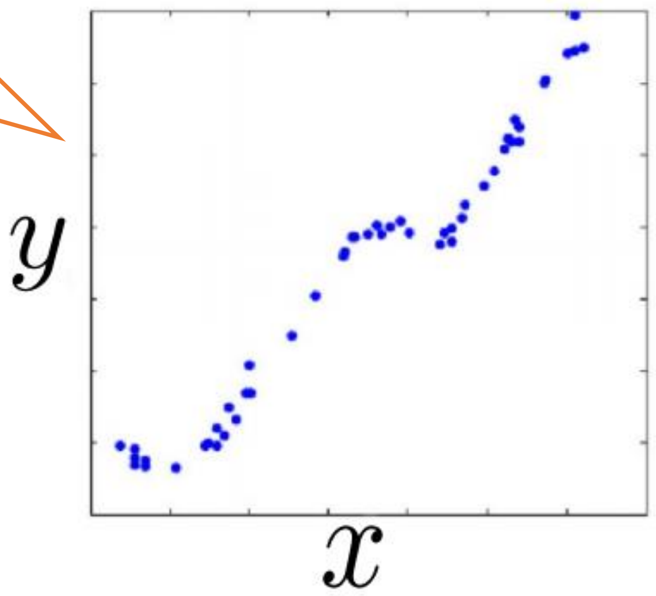
- Classes are now linearly separable in the two-dimensional space

Linear Models for Nonlinear Problems

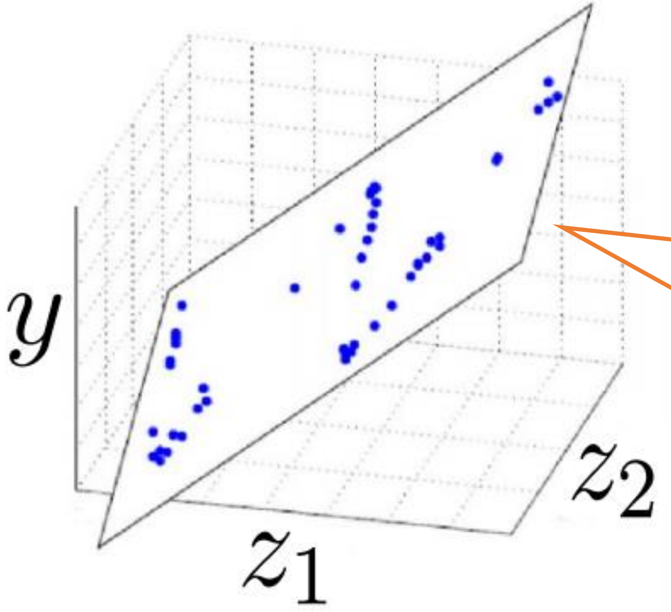
- The same idea can be applied for nonlinear regression as well

Not a linear relationship between inputs (x) and outputs (y)

A linear regression model will not work well



$$x \rightarrow \boldsymbol{z} = [z_1, z_2] = [x, \cos(x)]$$



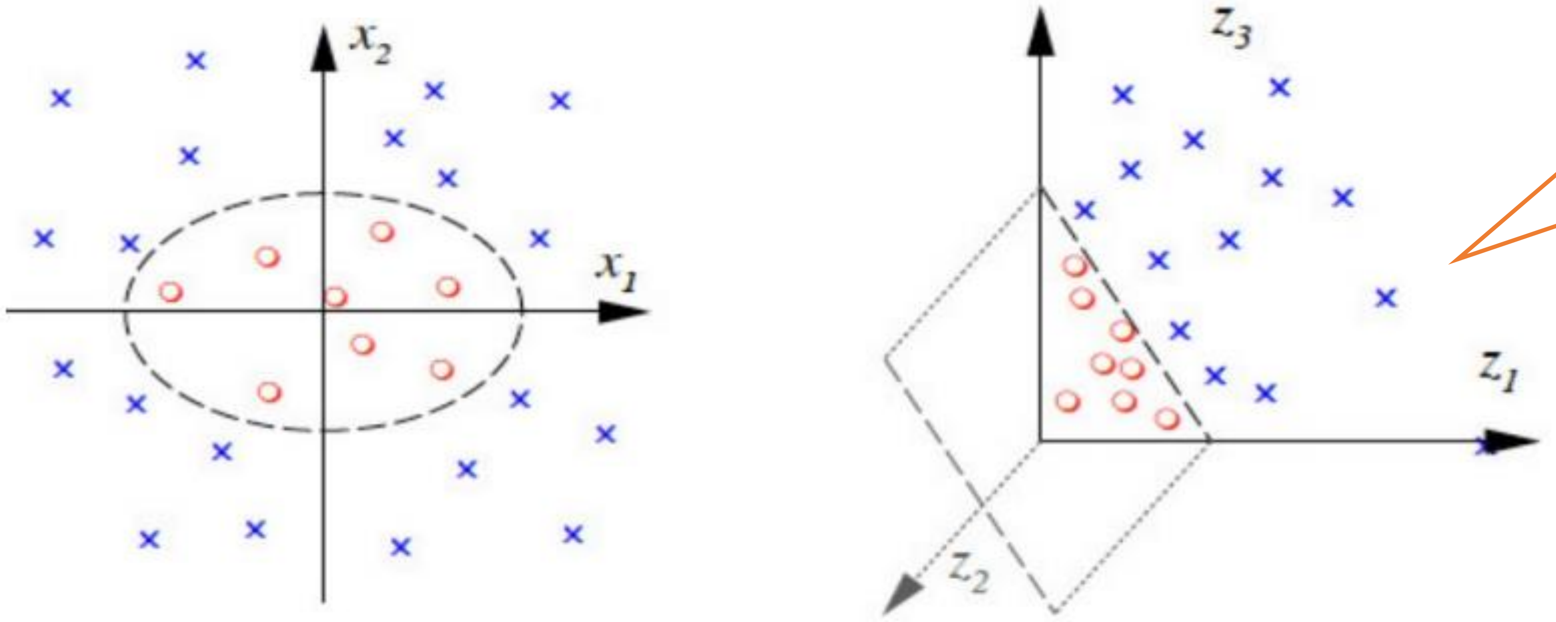
A linear regression model will work well with this new two-dim representation of the original one-dim inputs

Linear Models for Nonlinear Problems

- Can assume a feature mapping $\boldsymbol{\phi}$ that maps/transforms the inputs to a "nice" space

$$\phi : \mathbb{R}^2 \rightarrow \mathbb{R}^3$$

$$(x_1, x_2) \mapsto (z_1, z_2, z_3) := (x_1^2, \sqrt(2)x_1x_2, x_2^2)$$

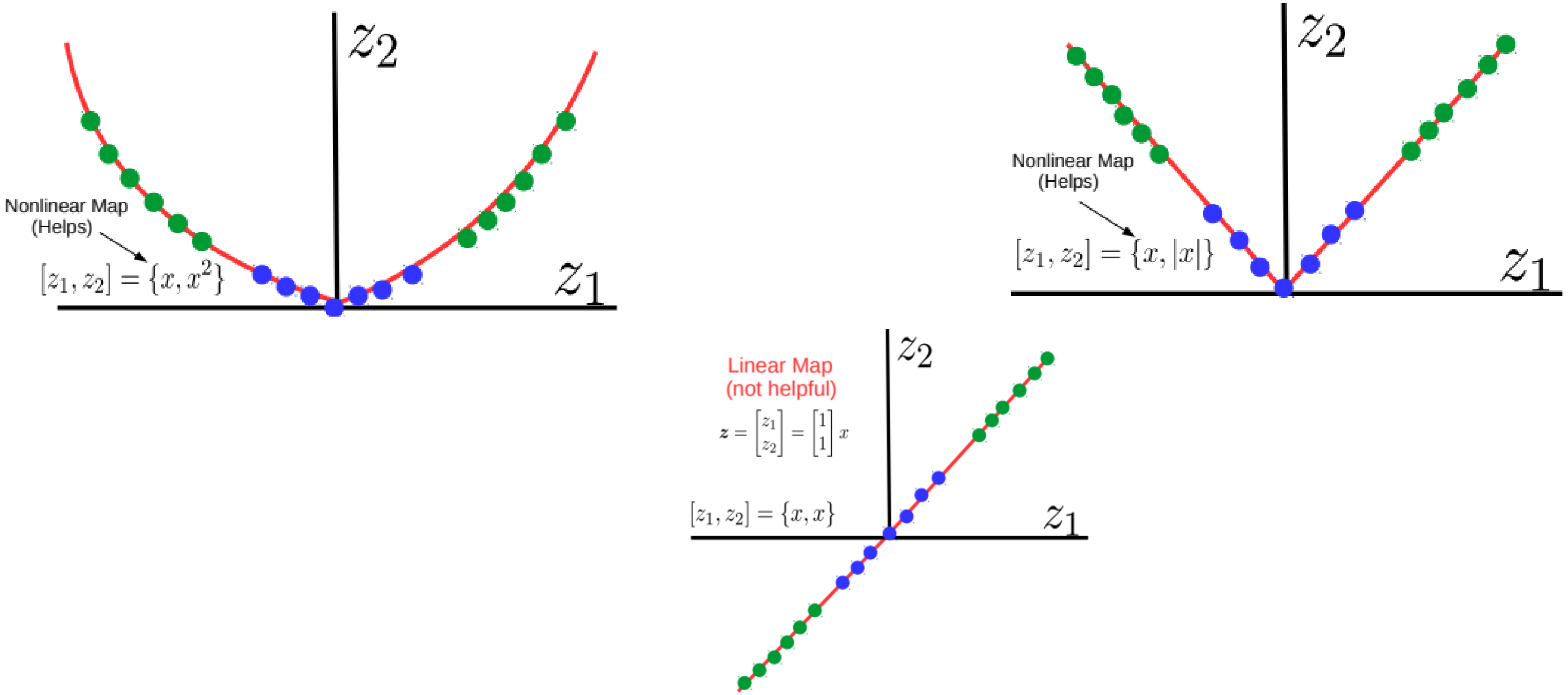


The linear model in the new feature space corresponds to a nonlinear model in the original feature space

- .. and then happily apply a linear model in the new space!

Not Every Mapping is Helpful

- Not every higher-dim mapping helps in learning nonlinear patterns
- Must be a nonlinear mapping
- For the nonlin classfn problem we saw earlier, consider some possible mappings

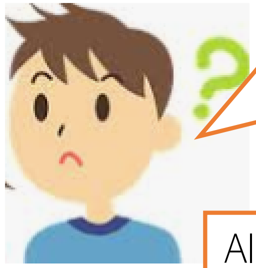


How to get these “good” (nonlinear) mappings?

- Can try to learn the mapping from the data itself (e.g., using deep learning - later)
- Can use pre-defined “good” mappings (e.g., defined by kernel functions - today’s topic)

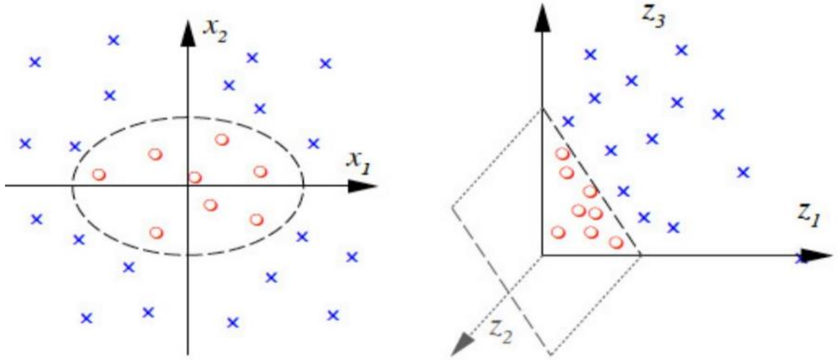
$$\phi : \mathbb{R}^2 \rightarrow \mathbb{R}^3$$

$$(x_1, x_2) \mapsto (z_1, z_2, z_3) := (x_1^2, \sqrt(2)x_1x_2, x_2^2)$$



Even if I knew a good mapping, it seems I need to apply it for every input. Won’t this be computationally expensive?

Also, the number of features will increase? Will it not slow down the learning algorithm?



Thankfully, using kernels, you don’t need to compute these mappings explicitly

The kernel will define an “implicit” feature mapping

Important: The idea can be applied to any ML algo in which training and test stage only require computing pairwise similarities b/w inputs

In a high-dim space implicitly defined by an underlying mapping $\boldsymbol{\phi}$ associated this this kernel function $\boldsymbol{k}(.,.)$

- Kernel: A function $\boldsymbol{k}(.,.)$ that gives dot product similarity b/w two inputs, say $\boldsymbol{x}_n$ and $\boldsymbol{x}_m$

Important: As we will see, computing $\boldsymbol{k}(.,.)$ does not require computing the mapping $\boldsymbol{\phi}$

$$k(\boldsymbol{x}_n, \boldsymbol{x}_m) = \phi(\boldsymbol{x}_n)^\top \phi(\boldsymbol{x}_m)$$

Kernels as (Implicit) Feature Maps

- Consider two inputs (in the same two-dim feature space): $\boldsymbol{x} = [x_1, x_2], \boldsymbol{z} = [z_1, z_2]$
- Suppose we have a function $k(.,.)$ which takes two inputs $\boldsymbol{x}$ and $\boldsymbol{z}$ and computes

Called the “kernel function”

$$k(\boldsymbol{x}, \boldsymbol{z}) = (\boldsymbol{x}^\top \boldsymbol{z})^2$$

Can think of this as a notion of similarity b/w $\boldsymbol{x}$ and $\boldsymbol{z}$

This is not a dot/inner product similarity but similarity using a more general function of $\boldsymbol{x}$ and $\boldsymbol{z}$ (square of dot product)

$$= (x_1 z_1 + x_2 z_2)^2$$

$$= x_1^2 z_1^2 + x_2^2 z_2^2 + 2 x_1 x_2 z_1 z_2$$

$$= (x_1^2, \sqrt{2} x_1 x_2, x_2^2)^\top (z_1^2, \sqrt{2} z_1 z_2, z_2^2)$$

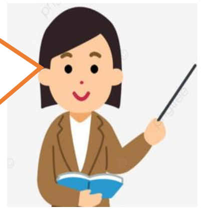
$$= \phi(\boldsymbol{x})^\top \phi(\boldsymbol{z})$$

Didn’t need to compute $\phi(x)$ explicitly. Just using the definition of the kernel $k(x, z) = (x^\top z)^2$ implicitly gave us this mapping for each input

Thus kernel function $k(\boldsymbol{x}, \boldsymbol{z}) = (\boldsymbol{x}^\top \boldsymbol{z})^2$ implicitly defined a feature mapping ϕ such that for $\boldsymbol{x} = [x_1, x_2]$, $\phi(\boldsymbol{x}) = (x_1^2, \sqrt{2} x_1 x_2, x_2^2)$

Dot product similarity in the new feature space defined by the mapping ϕ

Remember that a kernel does two things: Maps the data implicitly into a new feature space (feature transformation) and computes pairwise similarity between any two inputs under the new feature representation



- Also didn’t have to compute $\phi(\boldsymbol{x})^\top \phi(\boldsymbol{z})$. Defn $k(\boldsymbol{x}, \boldsymbol{z}) = (\boldsymbol{x}^\top \boldsymbol{z})^2$ gives that

Kernel Functions

As we saw, kernel function $\boldsymbol{k}(\boldsymbol{x},\boldsymbol{z}) = (\boldsymbol{x}^\top \boldsymbol{z})^2$ implicitly defines a feature mapping $\boldsymbol{\phi}$ such that for a two-dim $\boldsymbol{x} = [x_1, x_2]$, $\boldsymbol{\phi}(\boldsymbol{x}) = (x_1^2, \sqrt{2}x_1x_2, x_2^2)$

- Every kernel function $\boldsymbol{k}$ implicitly defines a feature mapping $\boldsymbol{\phi}$
- $\boldsymbol{\phi}$ takes input $\boldsymbol{x} \in \mathcal{X}$ (e.g., $\mathbb{R}^D$) and maps it to a new "feature space" $\mathcal{F}$
- The kernel function $\boldsymbol{k}$ can be seen as taking two points as inputs and computing their inner-product based similarity in the $\mathcal{F}$ space

For some kernels, as we will see shortly, $\boldsymbol{\phi}(x)$ (and thus the new feature space $\mathcal{F}$) can be very high-dimensional or even be infinite dimensional (but we don't need to compute it anyway, so it is not an issue)

$$\phi : \mathcal{X} \to \mathcal{F}$$

$$k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}, \quad k(\boldsymbol{x},\boldsymbol{z}) = \phi(\boldsymbol{x})^\top \phi(\boldsymbol{z})$$

- $\mathcal{F}$ needs to be a vector space with a dot product defined on it (a.k.a. a Hilbert space)
- Is any function $\boldsymbol{k}(\boldsymbol{x},\boldsymbol{z}) = \boldsymbol{\phi}(\boldsymbol{x})^\top \boldsymbol{\phi}(\boldsymbol{z})$ for some $\boldsymbol{\phi}$ a kernel function?
  - No. The function $\boldsymbol{k}$ must satisfy Mercer's Condition

Kernel Functions

- For $k(.,.)$ to be a kernel function
  - k must define a dot product for some Hilbert Space
  - Above is true if k is symmetric and positive semi-definite (p.s.d.) function (though there are exceptions; there are also "indefinite" kernels)

$$k(\boldsymbol{x},\boldsymbol{z}) = k(\boldsymbol{z},\boldsymbol{x})$$

$$\iint f(\boldsymbol{x})k(\boldsymbol{x},\boldsymbol{z})f(\boldsymbol{z})d\boldsymbol{x}d\boldsymbol{z} \geq 0$$

For all "square integrable" functions f (such functions satisfy $\int f(\boldsymbol{x})^2 d\boldsymbol{x} < \infty$

Loosely speaking a PSD function here means that if we evaluation this function for N inputs (N^2pairs) then the $N \times N$ matrix will be PSD (also called a kernel matrix)

  - The above condition is essentially known as Mercer's Condition

Can easily verify that the Mercer's Condition holds

- Let k_1, k_2 be two kernel functions then the following are as well
  - $k(\boldsymbol{x},\boldsymbol{z}) = k_1(\boldsymbol{x},\boldsymbol{z}) + k_2(\boldsymbol{x},\boldsymbol{z})$: simple sum
  - $k(\boldsymbol{x},\boldsymbol{z}) = \alpha k_1(\boldsymbol{x},\boldsymbol{z})$: scalar product
  - $k(\boldsymbol{x},\boldsymbol{z}) = k_1(\boldsymbol{x},\boldsymbol{z})k_2(\boldsymbol{x},\boldsymbol{z})$: direct product of two kernels

Can also combine these rules and the resulting function will also be a kernel function

Some Pre-defined Kernel Functions

Several other kernels proposed for non-vector data, such as trees, strings, etc

Remember that kernels are a notion of similarity between pairs of inputs

Kernels can have a pre-defined form or can be learned from data (a bit advanced for this course)

- Linear kernel: $k(\boldsymbol{x}, \boldsymbol{z}) = \boldsymbol{x}^\top \boldsymbol{z}$
- Quadratic Kernel: $k(\boldsymbol{x}, \boldsymbol{z}) = (\boldsymbol{x}^\top \boldsymbol{z})^2$ or $k(\boldsymbol{x}, \boldsymbol{z}) = (1 + \boldsymbol{x}^\top \boldsymbol{z})^2$
- Polynomial Kernel (of degree d): $k(\boldsymbol{x}, \boldsymbol{z}) = (\boldsymbol{x}^\top \boldsymbol{z})^d$ or $k(\boldsymbol{x}, \boldsymbol{z}) = (1 + \boldsymbol{x}^\top \boldsymbol{z})^d$
- Radial Basis Function (RBF) or "Gaussian" Kernel: $k(\boldsymbol{x}, \boldsymbol{z}) = \exp[-\gamma \|\boldsymbol{x} - \boldsymbol{z}\|^2]$
  - Gaussian kernel gives a similarity score between 0 and 1
  - $\gamma > 0$ is a hyperparameter (called the kernel bandwidth parameter)
  - The RBF kernel corresponds to an infinite dim. feature space $\mathcal{F}$ (i.e., you can't actually write down or store the map $\phi(\boldsymbol{x})$ explicitly – but we don't need to do that anyway ☺)
  - Also called "stationary kernel": only depends on the distance between $\boldsymbol{x}$ and $\boldsymbol{z}$ (translating both by the same amount won't change the value of $k(\boldsymbol{x}, \boldsymbol{z})$)
- Kernel hyperparameters (e.g., d, γ) can be set via cross-validation

Controls how the distance between two inputs should be converted into a similarity

RBF Kernel = Infinite Dimensional Mapping

- We saw that the RBF/Gaussian kernel is defined as $\boldsymbol{k}(\boldsymbol{x},\boldsymbol{z}) = \exp[-\gamma\, \|\boldsymbol{x} - \boldsymbol{z}\|^2]$
- Using this kernel corresponds to mapping data to infinite dimensional space

$$k(x,z) = \exp[-(x-z)^2] \qquad \text{(assuming } \gamma = 1 \text{ and } x \text{ and } z \text{ to be scalars)}$$

$$= \exp(-x^2)\exp(-z^2)\exp(2xz)$$

$$= \exp(-x^2)\exp(-z^2)\sum_{k=1}^{\infty}\frac{2^k x^k z^k}{k!}$$

$$= \phi(x)^\top \phi(z)$$

Thus an infinite-dim vector (ignoring the constants coming from the 2^k and $k!$ terms

- Here $\boldsymbol{\phi}(\boldsymbol{x}) = [\exp(-x^2)x^1, \exp(-x^2)x^2, \exp(-x^2)x^3, \ldots, \exp(-x^2)x^\infty]$
- But again, note that we never need to compute $\boldsymbol{\phi}(\boldsymbol{x})$ to compute $\boldsymbol{k}(\boldsymbol{x},\boldsymbol{z})$
  - $\boldsymbol{k}(\boldsymbol{x},\boldsymbol{z})$ is easily computable from its definition itself ($\exp[-(x-z)^2]$ in this case)

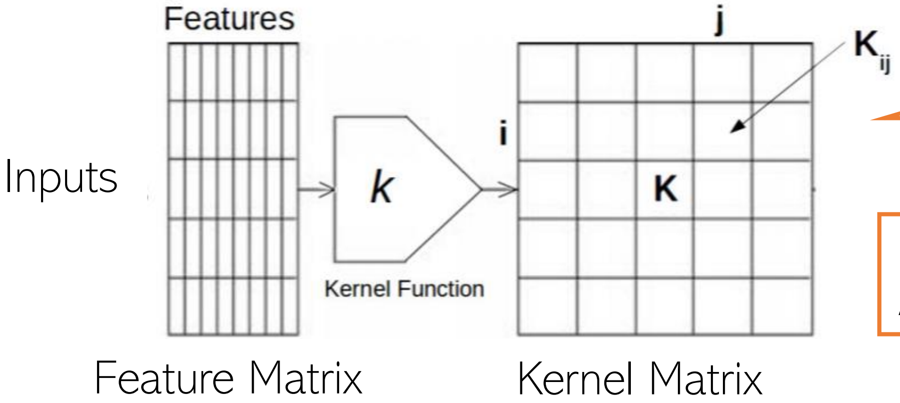
Kernel Matrix

- Kernel based ML algos work with kernel matrices rather than feature vectors
- Given $\boldsymbol{N}$ inputs, the kernel function $\boldsymbol{k}$ can be used to construct a Kernel Matrix $\boldsymbol{K}$
- The kernel matrix $\boldsymbol{K}$ is of size $\boldsymbol{N} \times \boldsymbol{N}$ with each entry defined as

$$K_{ij} = k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \phi(\boldsymbol{x}_i)^\top \phi(\boldsymbol{x}_j)$$

Note again that we don't need to compute ϕ and this dot product explicitly

- K_{ij} : Similarity between the i^{th} and j^{th} inputs in the kernel induced feature space ϕ



K is a symmetric and positive semi-definite matrix

$z^\top K z \geq 0 \; \forall z \in \mathbb{R}^N$

Also, all eigenvalues of K are non-negative

Coming up next..

- Applying kernel methods for SVM and ridge regression