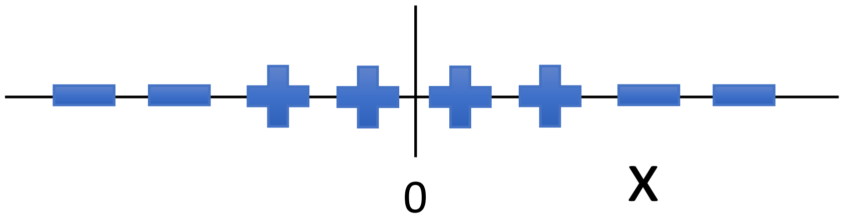
CS M146: Introduction to Machine Learning
Kernels

Aditya Grover

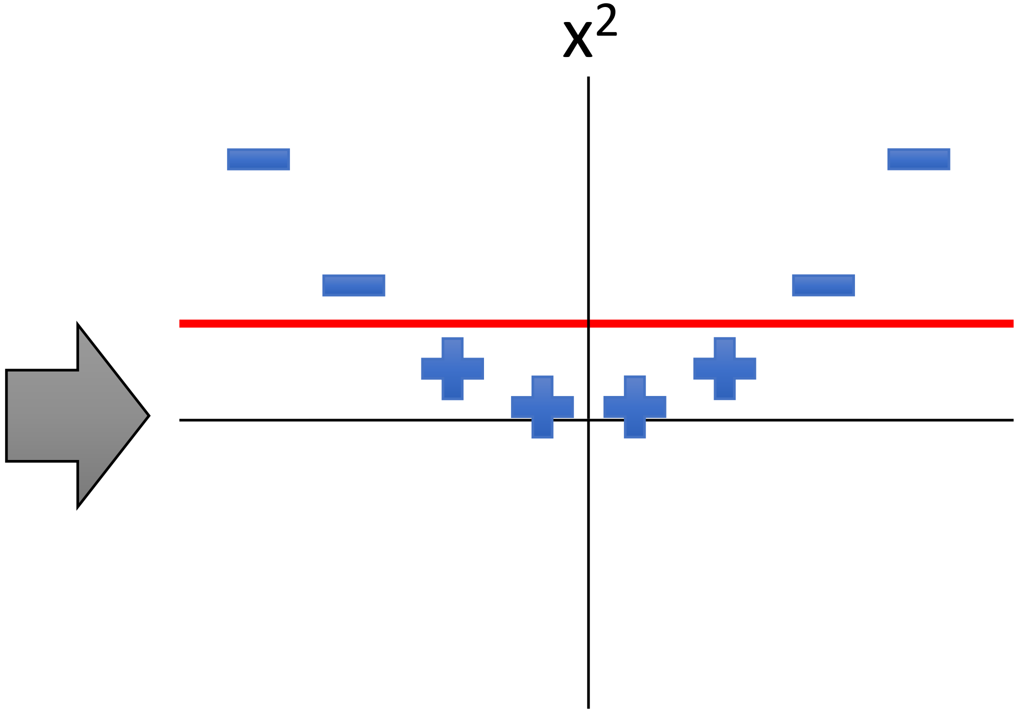
https://aditya-grover.github.io/ @adityagrover_

Revisiting Basis Functions

Can we separate the + and – examples using a linear decision boundary?

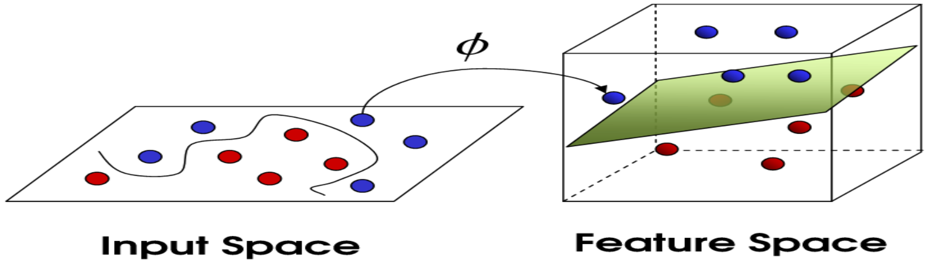


No



Yes

Mapping into a New Feature Space



- **Feature map/basis function:** Any function $\phi: \mathcal{X} \rightarrow \widehat{\mathcal{X}}$ that maps input attributes to a feature space
- For example, with $\boldsymbol{x} \in \mathbb{R}^2$, we can define:

$$\phi(\boldsymbol{x}) = \phi([x_1, x_2]) = [1, x_1, x_2, x_1^2, x_2^2, x_1 x_2]$$

  - Here, $\mathcal{X} \equiv \mathbb{R}^2$ and $\widehat{\mathcal{X}} \equiv \mathbb{R}^6$
  - For simplicity of notation, assume there is no bias
- **Key advantage:** Can still use a hypothesis class linear in $\boldsymbol{\theta}$

$$h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \boldsymbol{\theta}^T \phi(\boldsymbol{x})$$

Image from http://web.engr.oregonstate.edu/ ~afern/classes/cs534/

Learning With Feature Maps

- Gradient descent update rule

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \frac{1}{n} \sum_{i=1}^{n} \left(\boldsymbol{\theta}^T \phi(\boldsymbol{x}^{(i)}) - y^{(i)} \right) \phi(\boldsymbol{x}^{(i)})$$

- Each update requires computing $\boldsymbol{\theta}^T \phi(\boldsymbol{x}^{(i)})$
  - What if $\phi(\boldsymbol{x})$ is very high dimensional?
- E.g., consider all polynomial features with degree ≤3

$$\phi(\boldsymbol{x}) = \phi([x_1, x_2]) = [1, x_1, x_2, x_1^2, x_2^2, x_1 x_2, x_1^3, x_2^3, x_1^2 x_2, x_1 x_2^2]$$

  - **PRO:** Very expressive feature space, $\widehat{\mathcal{X}} \equiv \mathbb{R}^{10}$
  - **CON 1:** Prone to overfitting (need to use regularization!)
  - **CON 2:** High compute/memory requirements for gradient descent

Revisiting Regularized Linear Models

- Hypothesis for linear models

$$h_{\boldsymbol{\theta}}(\boldsymbol{x}) = g(\boldsymbol{\theta}^T \boldsymbol{x})$$

- ℓ_2 regularized loss for linear models:

$$J(\boldsymbol{\theta}) = \frac{1}{n}\sum_{i=1}^{n} L(\boldsymbol{\theta}^T \boldsymbol{x}^{(i)}, y^{(i)}) + \lambda \|\boldsymbol{\theta}_{1:d}\|_2^2$$

where

- **Linear Regression:** $g(z) = z$ and $L\left(\boldsymbol{\theta}^T \boldsymbol{x}^{(i)}, y^{(i)}\right) = \frac{1}{2}\left(y^{(i)} - \boldsymbol{\theta}^T \boldsymbol{x}^{(i)}\right)^2$
- **Perceptron:** $g(z) = \mathrm{sign}(z)$ and $L\left(\boldsymbol{\theta}^T \boldsymbol{x}^{(i)}, y^{(i)}\right) = \max(0, -y^{(i)}\boldsymbol{\theta}^T \boldsymbol{x}^{(i)})$
- **Logistic Regression:** $g(z) = \mathrm{sigmoid}(z)$ and $L\left(\boldsymbol{\theta}^T \boldsymbol{x}^{(i)}, y^{(i)}\right) = -y^{(i)}\log\left(\mathrm{sigmoid}\left(\boldsymbol{\theta}^T \boldsymbol{x}^{(i)}\right)\right) - \left(1 - y^{(i)}\right)\log\left(1 - \mathrm{sigmoid}\left(\boldsymbol{\theta}^T \boldsymbol{x}^{(i)}\right)\right)$

Linear Models With Feature Maps

- Hypothesis for linear models **with feature maps**:

$$h_{\boldsymbol{\theta}}(\boldsymbol{x}) = g(\boldsymbol{\theta}^T \phi(\boldsymbol{x}))$$

- ℓ_2 regularized loss for linear models **with feature maps**:

$$J(\boldsymbol{\theta}) = \frac{1}{n} \sum_{i=1}^{n} L(\boldsymbol{\theta}^T \phi(\boldsymbol{x}^{(i)}), y^{(i)}) + \lambda ||\boldsymbol{\theta}_{1:d}||_2^2$$

Can train and evaluate linear models with expressive feature maps efficiently using **Representor Theorem**

Representor Theorem

- **Representor Theorem** (special case): For any $\lambda > 0$, there exists some real-valued coefficients $\beta_i \in \mathbb{R}$ such that the minimizer of the regularized empirical risk $J(\boldsymbol{\theta})$ can be expressed as:

$$\widehat{\boldsymbol{\theta}} = \sum_{i=1}^{n} \beta_i \phi(\boldsymbol{x}^{(i)}) \tag{1}$$

- **Proof:** Since $\widehat{\boldsymbol{\theta}}$ is a a stationary point,

$$\nabla_{\boldsymbol{\theta}} J\left(\widehat{\boldsymbol{\theta}}\right) = 0 \tag{2}$$

Since $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \frac{1}{n} \sum_{i=1}^{n} L'(\boldsymbol{\theta}^T \phi(\boldsymbol{x}^{(i)}), y^{(i)}) \phi(\boldsymbol{x}^{(i)}) + 2\lambda\boldsymbol{\theta}$ (3)

Combining (2) and (3), we get $\widehat{\boldsymbol{\theta}} = -\frac{1}{2\lambda n} \sum_{i=1}^{n} L'(\boldsymbol{\theta}^T \phi(\boldsymbol{x}^{(i)}), y^{(i)}) \phi(\boldsymbol{x}^{(i)})$ (4)

Letting $\beta_i = -\frac{1}{2\lambda n} L'(\boldsymbol{\theta}^T \phi(\boldsymbol{x}^{(i)}), y^{(i)})$ in (4) gives (1) and finishes the proof

Reparametrized Risk

- Key immediate benefit of Representor Theorem is that it allows us to **reparameterize** the (regularized) empirical l
- Empirical risk for linear models with ℓ_2 regularization:

$$J(\boldsymbol{\theta}) = \frac{1}{n}\sum_{i=1}^{n} L(\boldsymbol{\theta}^T \phi(\boldsymbol{x}^{(i)}), y^{(i)}) + \lambda\|\boldsymbol{\theta}\|_2^2 \qquad (1)$$

- By Representor Theorem, we can plug $\widehat{\boldsymbol{\theta}} = \sum_{i=1}^{n} \beta_i \phi(\boldsymbol{x}^{(i)})$ in (1) to obtain an equivalent reparameterized objective $\tilde{J}$ w.r.t. $\boldsymbol{\beta}$

$$\tilde{J}(\boldsymbol{\beta}) = \frac{1}{n}\sum_{i=1}^{n} L\left(\left(\sum_{j=1}^{n} \beta_j \phi(\boldsymbol{x}^{(j)})\right)^T \phi(\boldsymbol{x}^{(i)}), y^{(i)}\right) + \lambda \left\|\sum_{i=1}^{n} \beta_i \phi(\boldsymbol{x}^{(i)})\right\|_2^2$$

$$= \frac{1}{n}\sum_{i=1}^{n} L\left(\sum_{j=1}^{n} \beta_j \phi(\boldsymbol{x}^{(j)})^T \phi(\boldsymbol{x}^{(i)}), y^{(i)}\right) + \lambda \sum_{i=1}^{n}\sum_{j=1}^{n} \beta_i \beta_j \phi(\boldsymbol{x}^{(j)})^T \phi(\boldsymbol{x}^{(i)})$$

Reparametrized Risk

- $\tilde{J}(\boldsymbol{\beta}) = \frac{1}{n}\sum_{i=1}^{n} L(\sum_{j=1}^{n} \beta_j \phi\left(\boldsymbol{x}^{(j)}\right)^T \phi(\boldsymbol{x}^{(i)}), y^{(i)}) + \lambda \sum_{i=1}^{n} \sum_{j=1}^{n} \beta_i \beta_j \phi\left(\boldsymbol{x}^{(j)}\right)^T \phi(\boldsymbol{x}^{(i)})$
- Define a **kernel function** $k\left(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)}\right) = \phi\left(\boldsymbol{x}^{(j)}\right)^T \phi(\boldsymbol{x}^{(i)})$

$$\tilde{J}(\boldsymbol{\beta}) = \frac{1}{n}\sum_{i=1}^{n} L(\sum_{j=1}^{n} \beta_j k\left(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)}\right), y^{(i)}) + \lambda \sum_{i=1}^{n} \sum_{j=1}^{n} \beta_i \beta_j k\left(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)}\right)$$

- **Notes:**
  - $k\left(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)}\right)$ is fixed during training and can be precomputed for all i, j
  - During training, we only need to optimize over $\boldsymbol{\beta}$. In contrast, earlier we needed to compute $\boldsymbol{\theta}^T \phi(\boldsymbol{x}^{(i)})$ at every iteration during training
  - Is that it? No, more savings in line via the **kernel trick**!

Kernel Trick

- Consider the **kernel function** $k(\boldsymbol{u}, \boldsymbol{v}) = \phi(\boldsymbol{u})^T \phi(\boldsymbol{v})$
  - Which is easier to compute: $\phi(\boldsymbol{u})$ or $k(\boldsymbol{u}, \boldsymbol{v})$?
  - Naively, for $\phi: \mathbb{R}^d \to \mathbb{R}^k$, both require $O(k)$ time.
- **Kernel trick:** Sometimes computing $k(\boldsymbol{u}, \boldsymbol{v})$ is much more efficient
- E.g., $\phi(\boldsymbol{x}) = \phi([x_1, x_2]) = [1, x_1, x_2, x_1^2, x_2^2, x_1 x_2, x_1^3, x_2^3, x_1^2 x_2, x_1 x_2^2]$

$$k(\boldsymbol{u}, \boldsymbol{v}) = \phi(\boldsymbol{u})^T \phi(\boldsymbol{v})$$

$$= 1 + \sum_{a=1}^{2} u_a v_a + \sum_{a,b \in \{1,2\}} u_a u_b v_a v_b + \sum_{a,b,c \in \{1,2\}} u_a u_b u_c v_a v_b v_c$$

$$= 1 + \textstyle\sum_{a=1}^{2} u_a v_a + \left(\sum_{a=1}^{2} u_a v_a\right)^2 + \left(\sum_{a=1}^{2} u_a v_a\right)^3$$

$$= 1 + \boldsymbol{u}^T \boldsymbol{v} + (\boldsymbol{u}^T \boldsymbol{v})^2 + (\boldsymbol{u}^T \boldsymbol{v})^3$$

- Computing $\boldsymbol{u}^T \boldsymbol{v}$ (and consequently $k(\boldsymbol{u}, \boldsymbol{v})$) only requires $O(d)$ time.

Kernel Trick

- Consider the **kernel function** $k(\boldsymbol{u}, \boldsymbol{v}) = \phi(\boldsymbol{u})^T \phi(\boldsymbol{v})$
  - Which is easier to compute: $\phi(\boldsymbol{u})$ or $k(\boldsymbol{u}, \boldsymbol{v})$?
  - Naively, for $\phi: \mathbb{R}^d \to \mathbb{R}^k$, both require $O(k)$ time.
- **Kernel trick:** Sometimes computing $k(\boldsymbol{u}, \boldsymbol{v})$ is much more efficient
- E.g., $\phi(\boldsymbol{x}) = \phi([x_1, x_2]) = [1, x_1, x_2, x_1^2, x_2^2, x_1 x_2, x_1^3, x_2^3, x_1^2 x_2, x_1 x_2^2]$

$$k(\boldsymbol{u}, \boldsymbol{v}) = \phi(\boldsymbol{u})^T \phi(\boldsymbol{v})$$

$$= 1 + \sum_{a=1}^{2} u_a v_a + \sum_{a,b \in \{1,2\}} u_a u_b v_a v_b + \sum_{a,b,c \in \{1,2\}} u_a u_b u_c v_a v_b v_c$$

$$= 1 + \textstyle\sum_{a=1}^{2} u_a v_a + \left(\sum_{a=1}^{2} u_a v_a\right)^2 + \left(\sum_{a=1}^{2} u_a v_a\right)^3$$

$$= 1 + \boldsymbol{u}^T \boldsymbol{v} + (\boldsymbol{u}^T \boldsymbol{v})^2 + (\boldsymbol{u}^T \boldsymbol{v})^3$$

- Computing $\boldsymbol{u}^T \boldsymbol{v}$ (and consequently $k(\boldsymbol{u}, \boldsymbol{v})$) only requires $O(d)$ time.

Kernelized Linear Models

- Kernel Trick: "Only compute what you need”

- We can apply the kernel trick to train and evaluate linear models with basis functions/feature maps → **Kernalized Linear Models**

- **Definition:** Given a kernel function k and a set of points $S = \{\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)} \ldots, \boldsymbol{x}^{(m)}\}$, we can define a **kernel matrix** $K \in \mathbb{R}^{m \times n}$ as:

$$K_{ij} = k\left(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)}\right)$$

Kernelized Linear Models

- **Step 1** (Pre-processing): Pre-compute Kernel Matrix $K \in \mathbb{R}^{n \times n}$ for n training points
- **Step 2** (Kernalized Training): Update $\boldsymbol{\beta}$ by gradient descent over reparameterized risk

$$\boldsymbol{\beta}^T K \boldsymbol{\beta}$$

$$\tilde{J}(\boldsymbol{\beta}) = \frac{1}{n}\sum_{i=1}^{n} L(\sum_{j=1}^{n} \beta_j K_{ij}, y^{(i)}) + \lambda \sum_{i=1}^{n}\sum_{j=1}^{n} \beta_i \beta_j K_{ij}$$

- **Step 3** (Kernalized Evaluation): Make predictions on a test point $\boldsymbol{x}$ by computing its **weighted** similarity with all training points :

$$h_{\boldsymbol{\theta}}(\boldsymbol{x}) = g\left(\boldsymbol{\theta}^T \phi(\boldsymbol{x})\right) = g\left(\sum_{i=1}^{n} \beta_i k\left(\boldsymbol{x}^{(i)}, \boldsymbol{x}\right)\right)$$

Example: Kernalized Ridge Regression

- Standard objective

$$J(\boldsymbol{\theta}) = \frac{1}{2n}\sum_{i=1}^{n}\left(y^{(i)} - \boldsymbol{\theta}^T\phi(\boldsymbol{x}^{(i)})\right)^2 + \lambda\|\boldsymbol{\theta}\|_2^2$$

- Gradient Update

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha\left(\frac{1}{n}\sum_{i=1}^{n}\left(\boldsymbol{\theta}^T\phi(\boldsymbol{x}^{(i)}) - y^{(i)}\right)\phi(\boldsymbol{x}^{(i)}) + \lambda\boldsymbol{\theta}\right)$$

- Kernalized objective

$$\tilde{J}(\boldsymbol{\beta}) = \frac{1}{2n}\sum_{i=1}^{n}\left(y^{(i)} - \sum_{j=1}^{n}\beta_j K_{ij}\right)^2 + \lambda\beta^T K\beta$$

- Kernalized Gradient Update

$$\boldsymbol{\beta} \leftarrow \boldsymbol{\beta} - \alpha\left(\frac{1}{n}\sum_{i=1}^{n}\left(\sum_{j=1}^{n}\beta_j K_{ij} - y^{(i)}\right) K^{(i)} + \lambda\sum_{i=1}^{n}\beta_i K^{(i)}\right)$$

where $K^{(i)} = K_{i,:}^T$ (transpose of i-th row of K)

Valid Kernels

- Given a feature map ϕ, we can define a kernel function $k(\boldsymbol{u}, \boldsymbol{v}) = \phi(\boldsymbol{u})^T \phi(\boldsymbol{v})$
- Intuitively, a kernel function defines a notion of similarity between datapoints
  - E.g., $K_{ij} = k\left(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)}\right)$ is the similarity between $\boldsymbol{x}^{(i)}$ and $\boldsymbol{x}^{(j)}$
- There are many possible notions of similarity
  - Euclidean distance, cosine similarity
  - Make your own similarity kernel! E.g., $\log \|\boldsymbol{x}^{(i)}\|_2 - \log \|\boldsymbol{x}^{(j)}\|_2$
- Given an arbitrary kernel function $k(\boldsymbol{u}, \boldsymbol{v})$, does it imply a feature map ϕ?

Valid Kernels

- **Definition:** A kernel is **valid** if it can be induced via a feature map ϕ
- How to determine if k is a valid kernel?
- **Method 1** (First Principles): To prove k is valid, show that there exists a function $\phi: \mathcal{X} \to \widehat{\mathcal{X}}$

  $$k(\boldsymbol{u}, \boldsymbol{v}) = \phi(\boldsymbol{u})^T \phi(\boldsymbol{v}) \text{ for } \textbf{all } \boldsymbol{u}, \boldsymbol{v} \in \mathcal{X}$$

  - Non-trivial to apply in practice!
- **Method 2:** Use **Mercer's Theorem**
- **Method 3:** Use kernel composition rules

Mercer's Theorem

- **Mercer's Theorem**: A kernel function k is valid **if and only if** over for *any* set of $m > 0$ points, the corresponding kernel matrix K is **symmetric** and **positive semi-definite (PSD)**
- *Note:* The m points are not necessarily the training set

- **Symmetric:** $K_{ij} = K_{ji}$ for all i, j
  - Equivalently, $K = K^T$
- **Positive semi-definite:** For all $\boldsymbol{z} \in \mathbb{R}^m$, $\boldsymbol{z}^T K \boldsymbol{z} \geq 0$
  - Equivalently, all eigenvalues of K are real and positive

Example

- Prove $k(\boldsymbol{u}, \boldsymbol{v}) = (\boldsymbol{u}^T \boldsymbol{v})^2$ is a valid kernel using Mercer's theorem
- **Symmetricity:**
  - $k(\boldsymbol{v}, \boldsymbol{u}) = (\boldsymbol{v}^T \boldsymbol{u})^2 = (\boldsymbol{u}^T \boldsymbol{v})^2 = k(\boldsymbol{u}, \boldsymbol{v})$
  - $K_{ij} = k\left(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)}\right) = K_{ji}$
- **PSD:** For any $\boldsymbol{z} \in \mathbb{R}^m$, we have:

$$
\begin{aligned}
\boldsymbol{z}^T K \boldsymbol{z} &= \textstyle\sum_{i=1}^m \sum_{j=1}^n z_i z_j K_{ij} \\
&= \textstyle\sum_{i=1}^m \sum_{j=1}^n z_i z_j (\boldsymbol{x}^{(i)\boldsymbol{T}} \boldsymbol{x}^{(j)})^2 \\
&= \textstyle\sum_{i=1}^m \sum_{j=1}^n [(\sqrt{z_i}\boldsymbol{x}^{(i)})^T (\sqrt{z_j}\boldsymbol{x}^{(j)})]^2 \\
&\geq 0
\end{aligned}
$$

Kernel Composition Rules

- Valid kernels can be **composed** with each other using many common algebraic operations to give valid kernels
- **Composition Rules:** If $k_1(\boldsymbol{u}, \boldsymbol{v})$ and $k_2(\boldsymbol{u}, \boldsymbol{v})$ are valid kernels, then the following are also valid kernels:
  - $k_1(\boldsymbol{u}, \boldsymbol{v}) + k_2(\boldsymbol{u}, \boldsymbol{v})$ is a valid kernel
  - $k_1(\boldsymbol{u}, \boldsymbol{v}) k_2(\boldsymbol{u}, \boldsymbol{v})$ is a valid kernel
  - $c k_1(\boldsymbol{u}, \boldsymbol{v})$ for any positive real $c > 0$
  - $g(\boldsymbol{u}) k_1(\boldsymbol{u}, \boldsymbol{v}) g(\boldsymbol{v})$ for any real-valued function g
  - $\exp k_1(\boldsymbol{u}, \boldsymbol{v})$

Summary

Kernels

Motivation: Featurize inputs to arbitrary high dimensions

Representer Theorem and Kernel Trick

Ensures we can compute loss and gradients efficiently using kernels

Identifying Valid Kernels

First Principles, Mercer's Theorem, Kernel Compositions