A.Z
Machine Learning
Nov, 2023

lec 7 Kernel method

Abstract

Kernel method

1 Prelude

Some key charactristics:

Embedding: input space X are embedded into a feature space F via a feature map $\phi : X \mapsto F$

Linear models: in feature space, typically $\mathbf{w}^T\boldsymbol{\phi}(\mathbf{x})$, efficient to find the optimal model, convexoptimization

Kernel trick: inner products of feature vectors $\kappa(\mathbf{x}, \mathbf{z}) = \sum_j \phi_j(\mathbf{x})\phi_j(\mathbf{z})$ rather than the explicit features $\phi(\mathbf{x})$; side-step the efficiency problems of high-dimensionality

Regularized learning:To avoid overfitting, large feature weights are penalized

Many data analysis algorithms can be 'kernelized', i.e. transformed to an equivalent form by replacing object descriptions (feature vectors) by pairwise similarities (kernels)

Informally, a **kernel** is a function that calculates the similarity between two objects

Formally: a **kernel** function is an inner product (scalar product, dot product) in a feature space F, denoted by $\langle \cdot, \cdot \rangle_F$

For example:

Linear kernel: If $\mathbf{x} \in \mathbb{R}^n$ and the feature map $\phi(\mathbf{x}) = \mathbf{x}$ is the identity, then $F = \mathbb{R}^n$ and the resulting kernel

$$\kappa_{\text{lin}}(\mathbf{x}, \mathbf{z}) = \langle \phi(\mathbf{x}), \phi(\mathbf{z}) \rangle_F = \langle \mathbf{x}, \mathbf{z} \rangle_{\mathbb{R}^n}$$

is called the linear kernel

Linear kernel therefore corresponds to the dot product in $\mathbb{R}^n$

$$\kappa_{\text{lin}}\left(\mathbf{x},\mathbf{z}\right)=\sum_{j=1}^{n}x_j z_j=\mathbf{x}^T\mathbf{z}$$

Geometric interpretation of the linear kernel: cosine angle between two feature vectors

$$\cos\beta=\frac{\mathbf{x}^T\mathbf{z}}{\|\mathbf{x}\|_2\|\mathbf{z}\|_2}=\frac{\kappa_{\text{lin}}\left(\mathbf{x},\mathbf{z}\right)}{\sqrt{\kappa_{\text{lin}}\left(\mathbf{x},\mathbf{x}\right)}\sqrt{\kappa_{\text{lin}}\left(\mathbf{z},\mathbf{z}\right)}},$$

where

$$\|\mathbf{x}\|_2=\sqrt{\kappa_{\text{lin}}(\mathbf{x},\mathbf{x})}=\sqrt{\langle\mathbf{x},\mathbf{x}\rangle}=\sqrt{\sum_{j=1}^{n}x_j^2}$$

is the Euclidean norm.

1.1 Kernel vs. Euclidean distance

Assume two vectors $\mathbf{x},\mathbf{z}\in\mathbb{R}^n$ with unit length $\|\mathbf{x}\|_2=\|\mathbf{z}\|_2=1$ Kernel: $\kappa(\mathbf{x},\mathbf{z})=\mathbf{x}^T\mathbf{z}$

Euclidean Distance: $d(\mathbf{x},\mathbf{z})=\|\mathbf{x}-\mathbf{z}\|_2=\sqrt{\sum_{k=1}^{n}\left(x_k-z_k\right)^2}$

Expanding the squares and using unit length of the vectors we get:

$$\begin{aligned}&\frac{1}{2}d(\mathbf{x},\mathbf{z})^2=\frac{1}{2}\|\mathbf{x}-\mathbf{z}\|_2^2\\&=\frac{1}{2}(\mathbf{x}-\mathbf{z})^T(\mathbf{x}-\mathbf{z})\\&=\frac{1}{2}\left(\|\mathbf{x}\|_2^2-2\mathbf{x}^T\mathbf{z}+\|\mathbf{z}\|_2^2\right)\\&=1-\mathbf{x}^T\mathbf{z}\\&=1-\kappa(\mathbf{x},\mathbf{z})\end{aligned}$$

Formally the underlying space of a kernel is required to be a Hilbertspace

2 Kernel matrix

In kernel methods, a kernel matrix, also called the Gram matrix, an $m\times m$ matrix of pairwise similarity values is used:

$$\mathbf{K} = \begin{bmatrix} \kappa(\mathbf{x}_1, \mathbf{x}_1) & \kappa(\mathbf{x}_1, \mathbf{x}_2) & \dots & \kappa(\mathbf{x}_1, \mathbf{x}_m) \\ \kappa(\mathbf{x}_2, \mathbf{x}_1) & \kappa(\mathbf{x}_2, \mathbf{x}_2) & \dots & \kappa(\mathbf{x}_2, \mathbf{x}_m) \\ \vdots & \vdots & \ddots & \vdots \\ \kappa(\mathbf{x}_m, \mathbf{x}_1) & \kappa(\mathbf{x}_m, \mathbf{x}_2) & \dots & \kappa(\mathbf{x}_m, \mathbf{x}_m) \end{bmatrix}$$

Each entry is an inner product between two data points $\kappa(\mathbf{x}_i, \mathbf{x}_j) = \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle$, where $\phi : X \mapsto \mathcal{F}$ is a feature map

Since an inner product is symmetric, $\mathbf{K}$ is a symmetric matrix

A symmetric matrix $\mathbf{A} \in \mathbb{R}^{m \times m}$ is **positive semi-definite (PSD)** if for any vector $\mathbf{v} \in \mathbb{R}^m$, we have $\mathbf{v}^T \mathbf{A} \mathbf{v} \geq 0$

A symmetric PSD matrix has non-negative eigenvalues $\lambda_1 \geq \cdots \geq \lambda_m \geq 0$

The kernel matrix corresponding to the kernel function $\kappa(\mathbf{x}, \mathbf{z}) = \langle \phi(\mathbf{x}), \phi(\mathbf{z}) \rangle$ on a set of data points $\{\mathbf{x}_i\}_{i=1}^m$ is PSD:

$$\begin{aligned} \mathbf{v}^T \mathbf{K} \mathbf{v} &= \sum_{i,j=1}^{n} v_i \mathbf{K}_{ij} v_j = \sum_{i,j=1}^{m} v_i \langle \phi(x_i), \phi(x_j) \rangle v_j = \\ &= \left\langle \sum_{i=1}^{m} v_i \phi(x_i), \sum_{j=1}^{m} v_j \phi(x_j) \right\rangle = \left\| \sum_{i=1}^{m} v_i \phi(x_i) \right\|^2 \geq 0 \end{aligned}$$

Consider objective of the dual SVM optimization problem

$$\text{OBJ}(\boldsymbol{\alpha}) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j y_i y_j \kappa(\mathbf{x}_i, \mathbf{x}_j) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \boldsymbol{\alpha}^T \mathbf{H} \boldsymbol{\alpha}$$

where we denoted $\mathbf{H} = (y_i y_j \kappa(\mathbf{x}_i, \mathbf{x}_j))_{i,j=1}^m$

It is easy to verify that $\mathbf{H}$ is the Hessian matrix of second derivatives of the objective;

$$\mathbf{H} = \left(\frac{\partial^2 OBJ(\boldsymbol{\alpha})}{\partial \alpha_i \partial \alpha_j} \right)_{i,j=1}^{m}$$

If $\mathbf{H}$ is PSD OBJ $(\boldsymbol{\alpha})$ is concave ($-OBJ(\boldsymbol{\alpha})$ is convex), and has no non-optimal local maxima

However, $\mathbf{H}$ is PSD if and only if $\mathbf{K}$ is PSD

Thus, a PSD kernel matrix $\mathbf{K}$ ensures that we can find a global optimum by gradient descent approaches

So the local optimum of an SVM optimization problem is also the global optimum

2.1 Rademacher complexity

Assume a symmetric positive definite kernel $\kappa : X \times X \mapsto \mathbb{R}$ with associated feature map ϕ, and a sample S of size m, with the kernel matrix $\mathbf{K} = (\kappa(x_i, x_j))_{i,j=1}^m$, and $\kappa(\mathbf{x}_i, \mathbf{x}_i) \leq r^2$ for all $i = 1, \ldots, m$

Empirical Rademacher complexity of the hypothesis class containing support vector machines

$$\mathcal{H} = \{\mathbf{x} \mapsto \langle \mathbf{w}, \phi(\mathbf{x}) \rangle : \|\mathbf{w}\| \leq B\}$$

for some $B \geq 0$ satisfies (c.f. Mohri book for the proof)

$$\hat{\mathcal{R}}_S(\mathcal{H}) \leq \frac{B\sqrt{\text{trace}(\mathbf{K})}}{m}$$

The key quantities are

- the upper bound B of the norm of weight vector : relates to the margin

- the trace of the kernel matrix $\text{trace}(\mathbf{K}) = \sum_{i=1}^m \kappa(\mathbf{x}_i, \mathbf{x}_i) = \sum_{i=1}^m \|\phi(\mathbf{x}_i)\|^2 \leq mr^2$: relates to the norm of the data points

2.2 Generalization error bound

We can plug the above to a Rademacher complexity based generalization bound

$$\begin{aligned} R(h) &\leq \hat{R}(h) + \hat{\mathcal{R}}(\mathcal{H}) + 3\sqrt{\frac{\log \frac{2}{\delta}}{2m}} \\ &\leq \hat{R}(h) + \frac{B\sqrt{\text{trace}(\mathbf{K})}}{m} + 3\sqrt{\frac{\log \frac{2}{\delta}}{2m}} \end{aligned}$$

Evaluating this bound required observing

(1) the empirical risk of the hypothesis $\hat{R}(h)$ on training data,

(2) the norm of the weight vector $B = \|\mathbf{w}\| = \sqrt{\sum_{i,j=1}^m \alpha_i \alpha_j y_i y_j \kappa(\mathbf{x}_i, \mathbf{x}_j)}$, and

(3) the trace of the kernel matrix

3 Non-linear kernels

By defining kernels that are non-linear functions of the original feature vectors, a linear models (e.g. SVM classifier) can be turned into a non-linear model

However, the learning algorithm does not need to be changed, apart from plugging in the new kernel matrix

The most commonly used non-linear kernels:

- Polynomial kernel: $\kappa_{pol}(\mathbf{x}, \mathbf{z}) = \left(\mathbf{x}^T\mathbf{z} + c\right)^q$

- Gaussian (or radial basis function, RBF) kernel: $\kappa_{RBF}(\mathbf{x}, \mathbf{z}) = \exp\left(-\|\mathbf{x} - \mathbf{z}\|^2 / \left(2\sigma^2\right)\right)$

3.1 Polynomial kernal

Given inputs $\mathbf{x}, \mathbf{z} \in \mathbb{R}^d$, the polynomial kernel is given by

$$\kappa_{pol}(\mathbf{x}, \mathbf{z}) = \left(\mathbf{x}^T\mathbf{z} + c\right)^q$$

where:

- $\mathbf{x}^T\mathbf{z}$ is the dot product of vectors $\mathbf{x}$ and $\mathbf{z}$.

- c is a non-negative real constant, often referred to as the "softening" term.

- q is a positive integer that determines the degree of the polynomial.

The degree q of the polynomial kernel determines the complexity of the model. A higher degree allows the model to capture more complex relationships in the data. However, a very high degree can lead to overfitting

The constant term c allows for control over the influence of higher-degree versus lower-degree terms in the polynomial.

Integer $q > 0$ gives the degree of the polynomial kernel Real value $c \geq 0$ is a weighting factor for lower order polynomial terms

The underlying features are non-linear: monomial combinations $x_1 \cdot x_2 \cdots x_k$ of degree $k \leq q$ of the original features x_j

3.1.1 Polynomial kernel on 2D inputs

A linear model in the polynomial feature space corresponds to anon-linear modelin the original feature space

Consider two-dimensional inputs $\mathbf{x} = [x_1, x_2]^T \in \mathbb{R}^2$

The second degree polynomial kernel is given by $\kappa\left(\mathbf{x}, \mathbf{x}'\right) = \left(\mathbf{x}^T\mathbf{x}' + c\right)^2$

We can write it as a inner product in $\mathbb{R}^6$:

$$\begin{aligned}\kappa(\mathbf{x},\mathbf{x}') &= \left(\mathbf{x}^T\mathbf{x}'+c\right)^2 = \left(x_1x_1'+x_2x_2'+c\right)^2 = \\ &= x_1x_1'x_1x_1' + x_2x_2'x_2x_2' + c^2 + \\ &+ 2x_1x_1'x_2x_2' + 2cx_1x_1' + 2cx_2x_2' \\ &\begin{bmatrix} x_1^2 \\ x_2^2 \\ \sqrt{2}x_1x_2 \\ \sqrt{2c}x_1 \\ \sqrt{2c}x_2 \\ c \end{bmatrix}^T \begin{bmatrix} x_1'^2 \\ x_2'^2 \\ \sqrt{2}x_1'x_2' \\ \sqrt{2c}x_1' \\ \sqrt{2c}x_2' \\ c \end{bmatrix} \\ &= \phi(\mathbf{x})^T\phi(\mathbf{x}'),\end{aligned}$$

where $\phi(\mathbf{x}) = \left[x_1^2, x_2^2, \sqrt{2}x_1x_2, \sqrt{2c}x_1, \sqrt{2c}x_2, c\right]^T$

So

$$\mathbf{w}^T\phi(\mathbf{x}) = w_1x_1^2 + w_2x_2^2 + w_3\sqrt{2}x_1x_2 + w_4\sqrt{2c}x_1 + w_5\sqrt{2c}x_2 + w_6c = 0$$

is a hyperplane in new 6-dimensional feature space, while a second degree polynomial in the original inputs space

Using the dual representation $\mathbf{w} = \sum_{i=1}^m \alpha_i y_i \phi(\mathbf{x}_i)$, the polynomial kernel allows non-linear classification in the input space by

$$\mathbf{w}^T\phi(\mathbf{x}) = \sum_i \alpha_i y_i \left\langle \phi(\mathbf{x}_i), \phi(\mathbf{x}) \right\rangle = \sum_i \alpha_i y_i \kappa(\mathbf{x}_i, \mathbf{x})$$

Here, $\mathbf{w}^T\phi(\mathbf{x})$ is linear with respect to the transformed features $\phi(\mathbf{x})$.

3.1.2 Example: XOR with polynomial kernel

Consider the following simple example:

Input data points $\{(-1,-1),(-1,1),(1,-1),(1,1)\}$ and the label (red = 1, blue = -1) given by a XOR type function

$$y = x_1x_2 = \begin{cases} +1 & \text{if } x_1 = x_2 \\ -1 & \text{if } x_1 \neq x_2 \end{cases}$$

The classes are not linearly separable

However, map the data using the feature map $\phi(\mathbf{x}) = \left[x_1^2, x_2^2, \sqrt{2}x_1x_2, \sqrt{2c}x_1, \sqrt{2c}x_2, c\right]^T$ underlying the polynomial kernel function

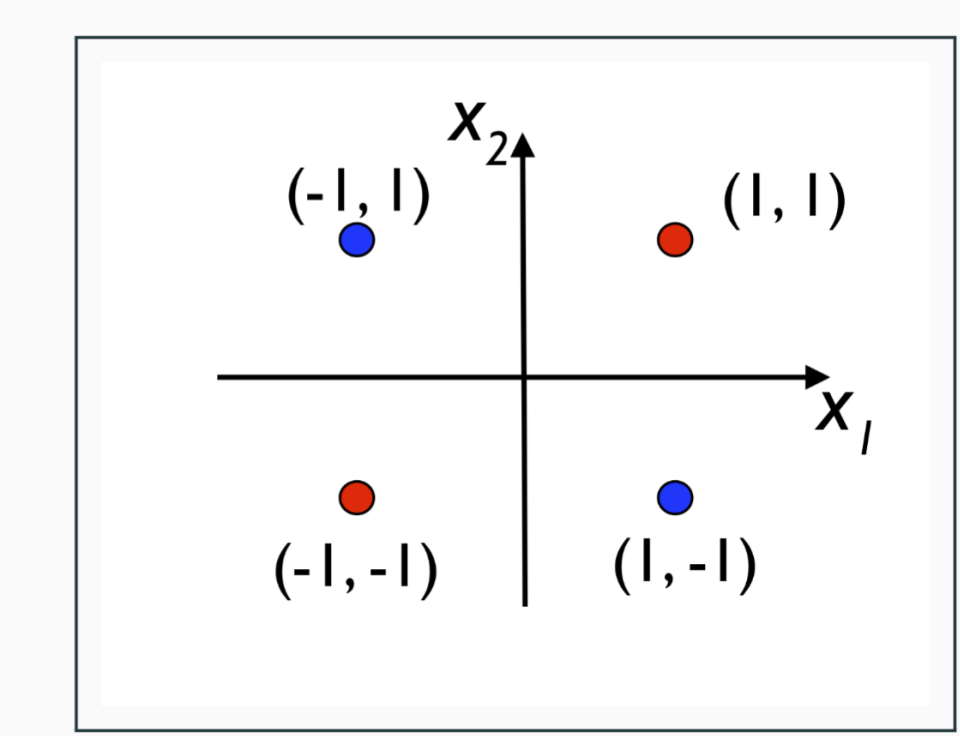

Figure 1: non-linearly separable

Now, the example data is linearly separable in the feature space, for example, choose $\alpha_i = 1/(4\sqrt{2})$, for all i :

$$\begin{aligned}\mathbf{w} &= \sum_i \alpha_i y_i \phi(\mathbf{x}_i) = \\ &= \left(\phi\left([-1,-1]^T\right) - \phi\left([-1,1]^T\right) - \phi\left([1,-1]^T\right) + \phi\left([1,1]^T\right)\right)/(4\sqrt{2}) \\ &= [0,0,1,0,0,0]^T\end{aligned}$$

We can consistently classify the example data by using the kernel function $\kappa(\mathbf{x}, \mathbf{x}') = \langle \phi(\mathbf{x}), \phi(\mathbf{x}') \rangle : h(\mathbf{x}) = \operatorname{sgn}\left(\sum_{i=1}^m \alpha_i y_i \kappa(\mathbf{x}_i, \mathbf{x})\right)$

3.1.3 Time complixity

The dimension of the polynomial feature space is $\begin{pmatrix} d+q \\ q \end{pmatrix} = O\left((d+q)^q\right)$ where d is the dimension of the input space X and q is the degree of the polynomial.

Explicitly maintaining the feature map $\phi(x)$ and the weight vector $\mathbf{w}$, and evaluating the model $\mathbf{w}^T\phi(\mathbf{x})$ takes $O\left(d^q\right)$ time and space

However, the polynomial kernel $\kappa(\mathbf{x}, \mathbf{x}') = \left(\mathbf{x}^T\mathbf{x}' + c\right)^q$ can be computed in time $O(d)$ in

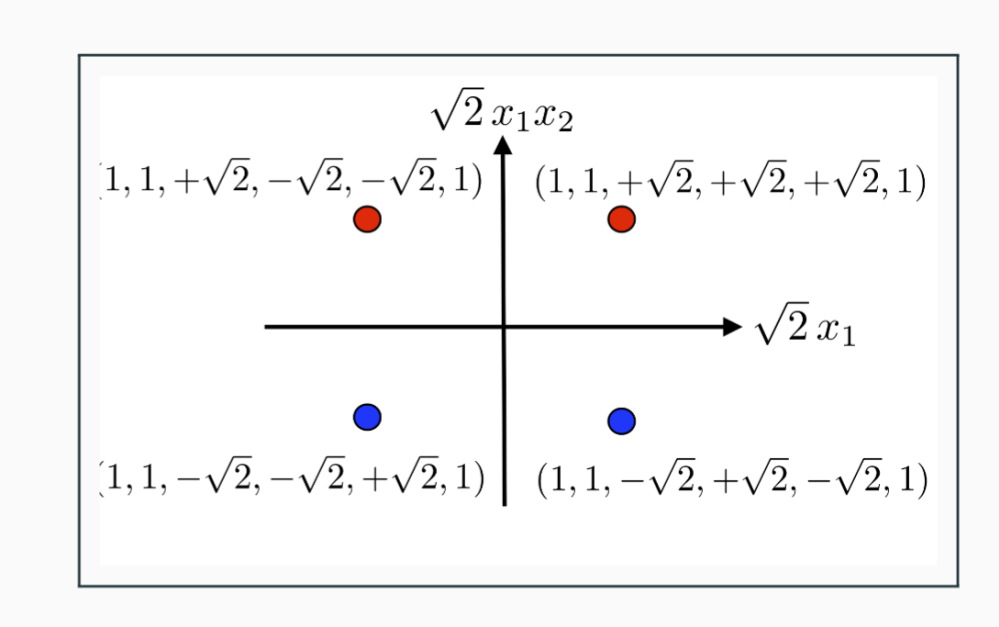

Figure 2: linearly separable

preprocessing, and evaluated in constant time

Evaluating the model using the dual representation $\sum_{i=1}^{m} \alpha_i y_i \kappa(\mathbf{x}_i, \mathbf{x})$ takes $O(m)$ time

Trade-off: No computational overhead from working in the high-dimensional feature space, but linear dependency in the size of training data

3.2 Gaussian kernel (Radial basis function kernel, RBF

Gaussian kernel between two inputs $\mathbf{x}, \mathbf{z} \in \mathbb{R}^d$ with bandwidth parameter $\sigma > 0$

$$\kappa_{RBF}(\mathbf{x}, \mathbf{z}) = \exp\left(-\|\mathbf{x} - \mathbf{z}\|^2 / \left(2\sigma^2\right)\right)$$

where $\|\mathbf{x} - \mathbf{z}\|^2$ is the squared Euclidean distance between the two input vectors $\mathbf{x}$ and $\mathbf{z}$, and σ is a positive parameter known as the bandwidth.

A larger value of σ results in a smoother kernel function, meaning the influence of a single training example reaches out further. This leads to a slower decay of the kernel value with respect to the distance between points. Consequently, the decision boundary tends to be more linear and smooth.

A smaller σ makes the kernel function decay more rapidly. This means that the influence of each training example is more localized. As a result, the decision boundary becomes more flexible and non-linear, allowing it to adapt more closely to the training data.

Gaussian kernel can be seen to correspond to an infinite dimensional polynomial kernel:

$$\kappa_{RBF}(\mathbf{x}, \mathbf{z}) = \exp\left(-\|\mathbf{x} - \mathbf{z}\|^2 / \left(2\sigma^2\right)\right) = \sum_{n=0}^{\infty} \frac{\left(\mathbf{x}^T \mathbf{z}\right)^n}{\sigma^{2n} n!}$$

This is due to its power series expansion, $e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!}$, which resembles an infinite sum of polynomial kernels of all degrees, weighted by $\frac{1}{\sigma^{2n} n!}$.

With a very small σ, there's a risk of overfitting, as the model may start to capture noise in the data. Conversely, with a very large σ, the model may become too simple and fail to capture important patterns.

4 Designing kernels

4.1 Approach I

Construct a feature map ϕ and think about efficient ways to compute the inner product $\langle \phi(x), \phi(x) \rangle$

- If $\phi(x)$ is very high-dimensional, computing the inner product element by element is slow, we don't want to do that

- For several cases, there are efficient algorithms to compute the kernel in low polynomial time, even with exponential or infinite dimension of ϕ

4.2 Approach II

Construct similarity measure and show that it qualifies as a kernel:

- Show that for any set of examples the matrix $K = \left(\kappa\left(\mathbf{x}_i, \mathbf{x}_j\right)\right)_{i,j=1}^{m}$ is positive semi-definite (PSD).

- In that case, there always is an underlying feature representation, for which the kernel represents the inner product

- Example: if you can show the matrix is a covariance matrix for some variates, you will know the matrix will be PSD.

4.3 Approach III

Convert a distance or a similarity into a kernel

- Take any distance $d(\mathbf{x}, \mathbf{z})$ or a similarity measure $s(\mathbf{x}, \mathbf{z})$ (that do not need to be a kernel)

- In addition a set of data points $Z = \{\mathbf{z}_j\}_{j=1}^{m}$ from the same domain is required (e.g. training data)

- Construct feature vector from distances (similarly for s):

$$\phi(x) = (d(\mathbf{x}, \mathbf{z}_1), d(\mathbf{x}, \mathbf{z}_2), \ldots, d(\mathbf{x}, \mathbf{z}_m))$$

- Compute linear kernel, also know as the empirical kernel map:

$$\kappa(\mathbf{x}, \mathbf{x}') = \langle \phi(\mathbf{x}), \phi(\mathbf{x}') \rangle$$

- This will always work technically, but requires that the data Z captures the essential patterns in the input space $\Longrightarrow$ need enough data

4.4 Approach IV

Making kernels from kernels

- Examples of elementary operations that give valid kernels when applied to kernels $\kappa_n, n = 1, 2, \ldots$

1. Sum: $\kappa(\mathbf{x}, \mathbf{z}) = \kappa_1(\mathbf{x}, \mathbf{z}) + \kappa_2(\mathbf{x}, \mathbf{z})$
2. Scaling with a positive scalar: $\kappa(\mathbf{x}, \mathbf{z}) = a\kappa_1(\mathbf{x}, \mathbf{z}), a > 0$
3. Itemwise product: $\kappa(\mathbf{x}, \mathbf{z}) = \kappa_1(\mathbf{x}, \mathbf{z})\kappa_2(\mathbf{x}, \mathbf{z})$
4. Normalization: $\kappa(\mathbf{x}, \mathbf{z}) = \frac{\kappa_1(\mathrm{x},\mathrm{z})}{\sqrt{\kappa_1(\mathrm{x},\mathrm{x})\kappa_1(\mathbf{z},\mathbf{z})}} = \left\langle \frac{\phi(\mathrm{x})}{\|\phi(\mathrm{x})\|}, \frac{\phi(\mathrm{z})}{\|\phi(\mathrm{z})\|} \right\rangle$
5. Pointwise limit: $\kappa(\mathbf{x}, \mathbf{z}) = \lim_{n\to\infty} \kappa_n(\mathbf{x}, \mathbf{z})$
6. Composition with a power series of radius of convergence ρ : $\kappa(\mathbf{x}, \mathbf{z}) = \sum_{n=0}^{\infty} a_n \kappa(\mathbf{x}, \mathbf{z})^n$, with $a_n \geq 0$ for all n, and $|\kappa(\mathbf{x}, \mathbf{z})| < \rho$

- The operations can be combined to construct arbitrarily complex kernels, e.g. polynomial kernels and Gaussian kernels can be derived this way (see details in the Mohri book ch. 6)

5 Kernels for structured data

In many applications the data does not come in the form of numerical vectors or data matrices, but from a general set of objects X

We can use an algorithm that directly computes the kernel values $\kappa(x_i, x_j)$ for any pair or objects x_i, x_j

The commonly seen form for kernels for two structured objects x_i and x_j

$$\kappa(x_i, x_j) = \sum_{s \in S} \phi_s(x_i)\, \phi_s(x_j),$$

where S is the set of substructures of interest and $\phi_s(x)$ is either

- An indicator function: $\phi_s(x) = 1$ if and only if x contains s

- A count: x contains $\phi_s(x)$ instances of s

In many cases, the set S has exponential size in the size of the objects and the feature vectors $\phi(x)$ are high dimensional and sparse

Efficient algorithms exist for computing the kernels $\kappa(x_i, x_j)$ directly from the structured data, skipping writing down the feature vectors $\phi(x)$

5.1 Example : String kernels

Lodhi et al. "Text classification usingstring kernels." Journal of Machine Learning Research 2.Feb (2002):419-444.32

5.2 Example : Graph kernels

Vishwanathan et al. "Graph kernels."Journal of Machine Learning Research 11 (2010): 1201-1242.33