732A99/TDDE01/732A68 Machine Learning

Lecture 3a Block 1: Kernel Methods

Jose M. Peña
IDA, Linköping University, Sweden

Contents and Literature

- Content
  - **Nonparametric** Kernel Methods
    - Histogram, Moving Window, and Kernel Classification
    - Histogram, Moving Window, and Kernel Regression
  - Kernel Trick
  - **Parametric** Kernel Methods
    - Kernel Ridge Regression
  - Summary
- Literature
  - Lindholm, A., Wahlström, N., Lindsten, F. and Schön, T. B. *Machine Learning - A First Course for Engineers and Scientists*. Cambridge University Press, 2022. Chapter 8-8.2.
  - Additional Literature
    - Hastie, T., Tibshirani, R. and Friedman, J. *The Elements of Statistical Learning*. Springer, 2009. Chapter 6.
    - Bishop, C. M. *Pattern Recognition and Machine Learning*. Springer, 2006. Sections 2.5 and 6.1-6.2.
    - Devroye, L., Györfi, L. and Lugosi, G. *A Probabilistic Theory of Pattern Recognition*. Springer, 1996. Sections 6.4 and 10.0.

Histogram Classification

- Consider binary ($y \in \{0, 1\}$) classification with input space $\mathbb{R}^p$.

[1] There seems to be some errors in the figure.

Histogram Classification

- Consider binary ($y \in \{0, 1\}$) classification with input space $\mathbb{R}^p$.
- The best classifier under the 0-1 loss function is $\widehat{y}(\boldsymbol{x}_*) = \arg\max_y p(y|\boldsymbol{x}_*)$.

[1] There seems to be some errors in the figure.

Histogram Classification

- Consider binary ($y \in \{0, 1\}$) classification with input space $\mathbb{R}^p$.
- The best classifier under the 0-1 loss function is $\widehat{y}(\boldsymbol{x}_*) = \arg\max_y p(y|\boldsymbol{x}_*)$.
- Since $\boldsymbol{x}_*$ may not appear in the training data $\{\boldsymbol{x}_i, y_i\}_{i=1}^n$ available, then

[1] There seems to be some errors in the figure.

Histogram Classification

- Consider binary ($y \in \{0, 1\}$) classification with input space $\mathbb{R}^p$.
- The best classifier under the 0-1 loss function is $\widehat{y}(\boldsymbol{x}_*) = \arg\max_y p(y|\boldsymbol{x}_*)$.
- Since $\boldsymbol{x}_*$ may not appear in the training data $\{\boldsymbol{x}_i, y_i\}_{i=1}^n$ available, then
  - divide the input space into p-dimensional cubes of side h, and
  - classify according to majority vote in the cube $C(\boldsymbol{x}_*, h)$ that contains $\boldsymbol{x}_*$.[1]

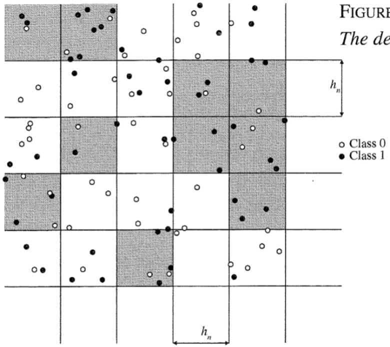


FIGURE 6.1. *A cubic histogram rule: The decision is* 1 *in the shaded area.*

[1]There seems to be some errors in the figure.

Histogram Classification

- Consider binary ($y \in \{0, 1\}$) classification with input space $\mathbb{R}^p$.
- The best classifier under the 0-1 loss function is $\widehat{y}(\boldsymbol{x}_*) = \arg\max_y p(y|\boldsymbol{x}_*)$.
- Since $\boldsymbol{x}_*$ may not appear in the training data $\{\boldsymbol{x}_i, y_i\}_{i=1}^n$ available, then
  - divide the input space into p-dimensional cubes of side h, and
  - classify according to majority vote in the cube $C(\boldsymbol{x}_*, h)$ that contains $\boldsymbol{x}_*$.[1]

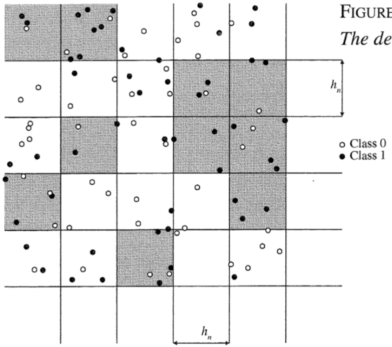


FIGURE 6.1. *A cubic histogram rule: The decision is* 1 *in the shaded area.*

- In other words,

$$\widehat{y}_C(\boldsymbol{x}_*) = \begin{cases} 0 & \text{if } \sum_{i=1}^n \mathbf{1}_{\{y_i=1, \boldsymbol{x}_i \in C(\boldsymbol{x}_*, h)\}} \le \sum_{i=1}^n \mathbf{1}_{\{y_i=0, \boldsymbol{x}_i \in C(\boldsymbol{x}_*, h)\}} \\ 1 & \text{otherwise} \end{cases}$$

[1]There seems to be some errors in the figure.

Moving Window Classification

- The predictions of the histogram rule are less accurate at the corners of the cube, because the corners may be far from the points in the cube. Then,

Moving Window Classification

- The predictions of the histogram rule are less accurate at the corners of the cube, because the corners may be far from the points in the cube. Then,
  - consider the points within a certain distance to the point to classify, and
  - classify the point according to majority vote.

Moving Window Classification

- The predictions of the histogram rule are less accurate at the corners of the cube, because the corners may be far from the points in the cube. Then,
  - consider the points within a certain distance to the point to classify, and
  - classify the point according to majority vote.
- In other words,

$$\widehat{y}_S(\boldsymbol{x}_*) = \begin{cases} 0 & \text{if } \sum_{i=1}^n \mathbf{1}_{\{y_i=1, \boldsymbol{x}_i \in S(\boldsymbol{x}_*, h)\}} \le \sum_{i=1}^n \mathbf{1}_{\{y_i=0, \boldsymbol{x}_i \in S(\boldsymbol{x}_*, h)\}} \\ 1 & \text{otherwise} \end{cases}$$

where $S(\boldsymbol{x}_*, h)$ is a p-dimensional closed ball of radius h centered at $\boldsymbol{x}_*$, or equivalently

$$\widehat{y}_S(\boldsymbol{x}_*) = \begin{cases} 0 & \text{if } \sum_{i=1}^n \mathbf{1}_{\{y_i=1, \boldsymbol{x}_* \in S(\boldsymbol{x}_i, h)\}} \le \sum_{i=1}^n \mathbf{1}_{\{y_i=0, \boldsymbol{x}_* \in S(\boldsymbol{x}_i, h)\}} \\ 1 & \text{otherwise} \end{cases}$$

Moving Window Classification

- The predictions of the histogram rule are less accurate at the corners of the cube, because the corners may be far from the points in the cube. Then,
  - consider the points within a certain distance to the point to classify, and
  - classify the point according to majority vote.
- In other words,

$$\widehat{y}_S(\boldsymbol{x}_*) = \begin{cases} 0 & \text{if } \sum_{i=1}^n \mathbf{1}_{\{y_i=1, \boldsymbol{x}_i \in S(\boldsymbol{x}_*, h)\}} \le \sum_{i=1}^n \mathbf{1}_{\{y_i=0, \boldsymbol{x}_i \in S(\boldsymbol{x}_*, h)\}} \\ 1 & \text{otherwise} \end{cases}$$

where $S(\boldsymbol{x}_*, h)$ is a p-dimensional closed ball of radius h centered at $\boldsymbol{x}_*$, or equivalently

$$\widehat{y}_S(\boldsymbol{x}_*) = \begin{cases} 0 & \text{if } \sum_{i=1}^n \mathbf{1}_{\{y_i=1, \boldsymbol{x}_* \in S(\boldsymbol{x}_i, h)\}} \le \sum_{i=1}^n \mathbf{1}_{\{y_i=0, \boldsymbol{x}_* \in S(\boldsymbol{x}_i, h)\}} \\ 1 & \text{otherwise} \end{cases}$$

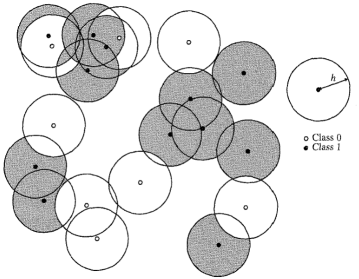


FIGURE 10.1. *The moving window rule in $\mathcal{R}^2$. The decision is 1 in the shaded area.*

Kernel Classification

- The moving window rule gives equal weight to all the points in the ball, which may be counterintuitive. Then,

Kernel Classification

- The moving window rule gives equal weight to all the points in the ball, which may be counterintuitive. Then,

$$\widehat{y}_k(\boldsymbol{x}_*) = \begin{cases} 0 & \text{if } \sum_{i=1}^{n} \mathbf{1}_{\{y_i=1\}} k\left(\frac{\boldsymbol{x}_* - \boldsymbol{x}_i}{h}\right) \leq \sum_{i=1}^{n} \mathbf{1}_{\{y_i=0\}} k\left(\frac{\boldsymbol{x}_* - \boldsymbol{x}_i}{h}\right) \\ 1 & \text{otherwise} \end{cases}$$

  where $k : \mathbb{R}^p \to \mathbb{R}$ is a kernel function, which is usually non-negative and monotone decreasing along rays starting from the origin. The parameter h is called smoothing factor or width. Intuitively, **the kernel function determines how similar any two data points are**.

Kernel Classification

- The moving window rule gives equal weight to all the points in the ball, which may be counterintuitive. Then,

$$\widehat{y}_k(\boldsymbol{x}_*) = \begin{cases} 0 & \text{if } \sum_{i=1}^n \mathbf{1}_{\{y_i=1\}} k\left(\frac{\boldsymbol{x}_* - \boldsymbol{x}_i}{h}\right) \leq \sum_{i=1}^n \mathbf{1}_{\{y_i=0\}} k\left(\frac{\boldsymbol{x}_* - \boldsymbol{x}_i}{h}\right) \\ 1 & \text{otherwise} \end{cases}$$

where $k : \mathbb{R}^p \to \mathbb{R}$ is a kernel function, which is usually non-negative and monotone decreasing along rays starting from the origin. The parameter h is called smoothing factor or width. Intuitively, **the kernel function determines how similar any two data points are**.

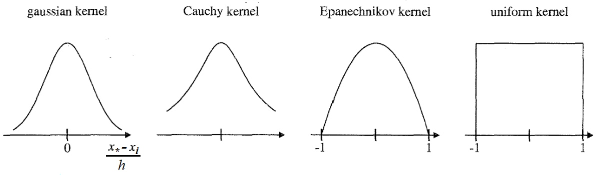


FIGURE 10.3. *Various kernels on $\mathcal{R}$.*

Kernel Classification

- The moving window rule gives equal weight to all the points in the ball, which may be counterintuitive. Then,

$$\widehat{y}_k(\boldsymbol{x}_*) = \begin{cases} 0 & \text{if } \sum_{i=1}^n \mathbf{1}_{\{y_i=1\}} k\left(\frac{\boldsymbol{x}_* - \boldsymbol{x}_i}{h}\right) \le \sum_{i=1}^n \mathbf{1}_{\{y_i=0\}} k\left(\frac{\boldsymbol{x}_* - \boldsymbol{x}_i}{h}\right) \\ 1 & \text{otherwise} \end{cases}$$

where $k : \mathbb{R}^p \to \mathbb{R}$ is a kernel function, which is usually non-negative and monotone decreasing along rays starting from the origin. The parameter h is called smoothing factor or width. Intuitively, **the kernel function determines how similar any two data points are**.

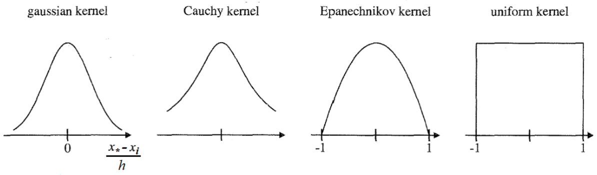
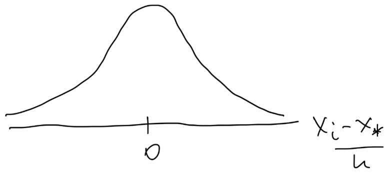
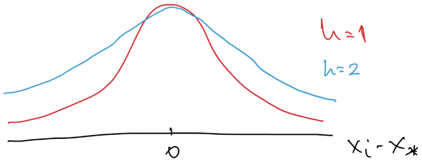


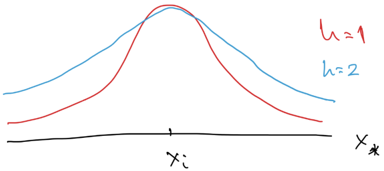
FIGURE 10.3. *Various kernels on $\mathcal{R}$.*

- Gaussian kernel: $k(u) = exp(-||u||^2)$ where $||\cdot||$ is the Euclidean norm.
- Cauchy kernel: $k(u) = 1/(1 + ||u||^{D+1})$
- Epanechnikov kernel: $k(u) = (1 - ||u||^2)\mathbf{1}_{\{||u|| \le 1\}}$
- Moving window kernel (a.k.a. uniform kernel): $k(u) = \mathbf{1}_{\{||u|| \le 1\}}$

Kernel Classification







Kernel Classification

$$\widehat{y}_k(\boldsymbol{x}_*) = \begin{cases} 0 & \text{if } \sum_{i=1}^n \mathbf{1}_{\{y_i=1\}} k\left(\frac{\boldsymbol{x}_* - \boldsymbol{x}_i}{h}\right) \leq \sum_{i=1}^n \mathbf{1}_{\{y_i=0\}} k\left(\frac{\boldsymbol{x}_* - \boldsymbol{x}_i}{h}\right) \\ 1 & \text{otherwise} \end{cases}$$

or equivalently

$$\widehat{y}_k(\boldsymbol{x}_*) = \begin{cases} 0 & \text{if } \sum_{i=1}^n \mathbf{1}_{\{y_i=1\}} k\left(\frac{\boldsymbol{x}_* - \boldsymbol{x}_i}{h}\right) - \sum_{i=1}^n \mathbf{1}_{\{y_i=0\}} k\left(\frac{\boldsymbol{x}_* - \boldsymbol{x}_i}{h}\right) \leq 0 \\ 1 & \text{otherwise} \end{cases}$$

or equivalently

$$\widehat{y}_k(\boldsymbol{x}_*) = \begin{cases} 0 & \text{if } \sum_{i=1}^n (2y_i - 1) k\left(\frac{\boldsymbol{x}_* - \boldsymbol{x}_i}{h}\right) \leq 0 \\ 1 & \text{otherwise} \end{cases}$$

Kernel Classification

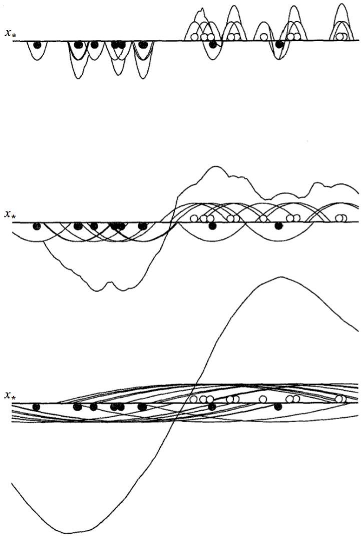


FIGURE 10.2. *Kernel rule on the real line. The figure shows* $\sum_{i=1}^{n}(2y_i - 1)\, k((x_* - x_i)/h)$ *for* $n = 20$, $k(u) = (1 - u^2)I_{\{|u| \le 1\}}$ *(the Epanechnikov kernel), and three smoothing factors h. One definitely undersmooths and one oversmooths.*

Histogram, Moving Window, and Kernel Regression

- Consider regressing an unidimensional continuous random variable on a p-dimensional continuous random variable.

Histogram, Moving Window, and Kernel Regression

- Consider regressing an unidimensional continuous random variable on a p-dimensional continuous random variable.
- The best regression function under the squared error loss function is $\widehat{y}(\boldsymbol{x}_*) = \mathbb{E}_Y[y|\boldsymbol{x}_*]$.

Histogram, Moving Window, and Kernel Regression

- Consider regressing an unidimensional continuous random variable on a p-dimensional continuous random variable.
- The best regression function under the squared error loss function is $\widehat{y}(\boldsymbol{x}_*) = \mathbb{E}_Y[y|\boldsymbol{x}_*]$.
- Since $\boldsymbol{x}_*$ may not appear in the training data $\{\boldsymbol{x}_i, y_i\}_{i=1}^n$ available, then we average over the points in $C(\boldsymbol{x}_*, h)$ or $S(\boldsymbol{x}_*, h)$, or kernel-weighted average over all the points.

Histogram, Moving Window, and Kernel Regression

- Consider regressing an unidimensional continuous random variable on a p-dimensional continuous random variable.
- The best regression function under the squared error loss function is $\widehat{y}(\boldsymbol{x}_*) = \mathbb{E}_Y[y|\boldsymbol{x}_*]$.
- Since $\boldsymbol{x}_*$ may not appear in the training data $\{\boldsymbol{x}_i, y_i\}_{i=1}^n$ available, then we average over the points in $C(\boldsymbol{x}_*, h)$ or $S(\boldsymbol{x}_*, h)$, or kernel-weighted average over all the points.
- In other words,

$$\widehat{y}_C(\boldsymbol{x}_*) = \frac{\sum_{\boldsymbol{x}_i \in C(\boldsymbol{x}_*, h)} y_i}{|\{\boldsymbol{x}_i \in C(\boldsymbol{x}_*, h)\}|}$$

or

$$\widehat{y}_S(\boldsymbol{x}_*) = \frac{\sum_{\boldsymbol{x}_i \in S(\boldsymbol{x}_*, h)} y_i}{|\{\boldsymbol{x}_i \in S(\boldsymbol{x}_*, h)\}|}$$

or

$$\widehat{y}_k(\boldsymbol{x}_*) = \frac{\sum_{i=1}^n k\left(\frac{\boldsymbol{x}_* - \boldsymbol{x}_i}{h}\right) y_i}{\sum_{i=1}^n k\left(\frac{\boldsymbol{x}_* - \boldsymbol{x}_i}{h}\right)}$$

Kernel Trick

- The kernel function $k\left(\frac{\boldsymbol{x}-\boldsymbol{x}'}{h}\right)$ only depends on the difference between the data points $\boldsymbol{x}$ and $\boldsymbol{x}'$ and, thus, it is invariant to data point translations and, thus, it can be generalized as $k(\boldsymbol{x}, \boldsymbol{x}')$. For instance,
  - Polynomial kernel: $k(\boldsymbol{x}, \boldsymbol{x}') = (\boldsymbol{x}^T\boldsymbol{x}' + c)^{d-1}$
  - Gaussian kernel: $k(\boldsymbol{x}, \boldsymbol{x}') = exp(-||\boldsymbol{x} - \boldsymbol{x}'||^2/2\ell^2)$

Kernel Trick

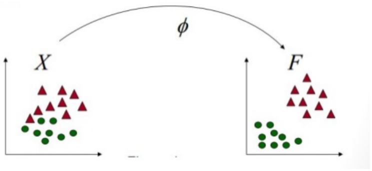
- The kernel function $k\left(\frac{\boldsymbol{x}-\boldsymbol{x}'}{h}\right)$ only depends on the difference between the data points $\boldsymbol{x}$ and $\boldsymbol{x}'$ and, thus, it is invariant to data point translations and, thus, it can be generalized as $k(\boldsymbol{x},\boldsymbol{x}')$. For instance,
  - Polynomial kernel: $k(\boldsymbol{x},\boldsymbol{x}') = (\boldsymbol{x}^T\boldsymbol{x}' + c)^{d-1}$
  - Gaussian kernel: $k(\boldsymbol{x},\boldsymbol{x}') = exp(-||\boldsymbol{x}-\boldsymbol{x}'||^2/2\ell^2)$
- If the matrix
$$\begin{pmatrix} k(\boldsymbol{x}_1,\boldsymbol{x}_1) & \dots & k(\boldsymbol{x}_1,\boldsymbol{x}_n) \\ \vdots & \dots & \vdots \\ k(\boldsymbol{x}_n,\boldsymbol{x}_1) & \dots & k(\boldsymbol{x}_n,\boldsymbol{x}_n) \end{pmatrix}$$
is symmetric and positive semi-definite for all training datasets $\{\boldsymbol{x}_i\}_{i=1}^n$, then $k(\boldsymbol{x},\boldsymbol{x}') = \boldsymbol{\phi}(\boldsymbol{x})^T\boldsymbol{\phi}(\boldsymbol{x}')$ where $\boldsymbol{\phi}(\cdot)$ is a **mapping from the input space to the feature space**. The converse is also true.

Kernel Trick

- The kernel function $k\left(\frac{\boldsymbol{x}-\boldsymbol{x}'}{h}\right)$ only depends on the difference between the data points $\boldsymbol{x}$ and $\boldsymbol{x}'$ and, thus, it is invariant to data point translations and, thus, it can be generalized as $k(\boldsymbol{x},\boldsymbol{x}')$. For instance,
  - Polynomial kernel: $k(\boldsymbol{x},\boldsymbol{x}') = (\boldsymbol{x}^T\boldsymbol{x}' + c)^{d-1}$
  - Gaussian kernel: $k(\boldsymbol{x},\boldsymbol{x}') = exp(-\|\boldsymbol{x}-\boldsymbol{x}'\|^2/2\ell^2)$
- If the matrix

$$\begin{pmatrix} k(\boldsymbol{x}_1,\boldsymbol{x}_1) & \dots & k(\boldsymbol{x}_1,\boldsymbol{x}_n) \\ \vdots & \dots & \vdots \\ k(\boldsymbol{x}_n,\boldsymbol{x}_1) & \dots & k(\boldsymbol{x}_n,\boldsymbol{x}_n) \end{pmatrix}$$

  is symmetric and positive semi-definite for all training datasets $\{\boldsymbol{x}_i\}_{i=1}^n$, then $k(\boldsymbol{x},\boldsymbol{x}') = \boldsymbol{\phi}(\boldsymbol{x})^T\boldsymbol{\phi}(\boldsymbol{x}')$ where $\boldsymbol{\phi}(\cdot)$ is a **mapping from the input space to the feature space**. The converse is also true.

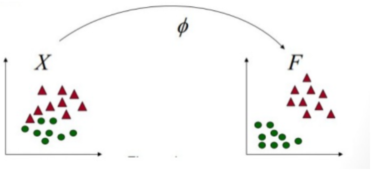


Kernel Trick

- The kernel function $k\left(\frac{\boldsymbol{x}-\boldsymbol{x}'}{h}\right)$ only depends on the difference between the data points $\boldsymbol{x}$ and $\boldsymbol{x}'$ and, thus, it is invariant to data point translations and, thus, it can be generalized as $k(\boldsymbol{x}, \boldsymbol{x}')$. For instance,
  - Polynomial kernel: $k(\boldsymbol{x}, \boldsymbol{x}') = (\boldsymbol{x}^T\boldsymbol{x}' + c)^{d-1}$
  - Gaussian kernel: $k(\boldsymbol{x}, \boldsymbol{x}') = exp(-\|\boldsymbol{x} - \boldsymbol{x}'\|^2/2\ell^2)$
- If the matrix

$$\begin{pmatrix} k(\boldsymbol{x}_1, \boldsymbol{x}_1) & \dots & k(\boldsymbol{x}_1, \boldsymbol{x}_n) \\ \vdots & \dots & \vdots \\ k(\boldsymbol{x}_n, \boldsymbol{x}_1) & \dots & k(\boldsymbol{x}_n, \boldsymbol{x}_n) \end{pmatrix}$$

is symmetric and positive semi-definite for all training datasets $\{\boldsymbol{x}_i\}_{i=1}^n$, then $k(\boldsymbol{x}, \boldsymbol{x}') = \boldsymbol{\phi}(\boldsymbol{x})^T\boldsymbol{\phi}(\boldsymbol{x}')$ where $\boldsymbol{\phi}(\cdot)$ is a **mapping from the input space to the feature space**. The converse is also true.



- The feature space may be non-linear and even infinite dimensional. For instance,

$$\boldsymbol{\phi}(x) = (1, \sqrt{3}x, \sqrt{3}x^2, x^3)^T$$

for the polynomial kernel with $p = 1$, $d = 4$, and $c = 1$.

Kernel Trick

- Consider again moving window classification or regression.

Kernel Trick

- Consider again moving window classification or regression.
- Note that $\boldsymbol{x} \in S(\boldsymbol{x}_*, h)$ if and only if $\|\boldsymbol{x} - \boldsymbol{x}_*\| \leq h$.

Kernel Trick

- Consider again moving window classification or regression.
- Note that $\boldsymbol{x} \in S(\boldsymbol{x}_*, h)$ if and only if $||\boldsymbol{x} - \boldsymbol{x}_*|| \leq h$.
- The Euclidean distance in the input space is

$$||\boldsymbol{x} - \boldsymbol{x}_*|| = \sqrt{(\boldsymbol{x} - \boldsymbol{x}_*)^T(\boldsymbol{x} - \boldsymbol{x}_*)} = \sqrt{\boldsymbol{x}^T\boldsymbol{x} + \boldsymbol{x}_*^T\boldsymbol{x}_* - 2\boldsymbol{x}^T\boldsymbol{x}_*}$$

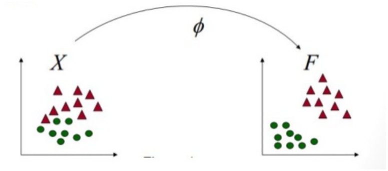
Kernel Trick

- Consider again moving window classification or regression.
- Note that $\boldsymbol{x} \in S(\boldsymbol{x}_*, h)$ if and only if $||\boldsymbol{x} - \boldsymbol{x}_*|| \leq h$.
- The Euclidean distance in the input space is

$$||\boldsymbol{x} - \boldsymbol{x}_*|| = \sqrt{(\boldsymbol{x} - \boldsymbol{x}_*)^T(\boldsymbol{x} - \boldsymbol{x}_*)} = \sqrt{\boldsymbol{x}^T\boldsymbol{x} + \boldsymbol{x}_*^T\boldsymbol{x}_* - 2\boldsymbol{x}^T\boldsymbol{x}_*}$$

- The Euclidean distance in the (**hopefully more convenient**) feature space

$$\begin{aligned} ||\phi(\boldsymbol{x}) - \phi(\boldsymbol{x}_*)|| &= \sqrt{\phi(\boldsymbol{x}^T)\phi(\boldsymbol{x}) + \phi(\boldsymbol{x}_*^T)\phi(\boldsymbol{x}_*) - 2\phi(\boldsymbol{x}^T)\phi(\boldsymbol{x}_*)} \\ &= \sqrt{k(\boldsymbol{x},\boldsymbol{x}) + k(\boldsymbol{x}_*,\boldsymbol{x}_*) - 2k(\boldsymbol{x},\boldsymbol{x}_*)} \end{aligned}$$



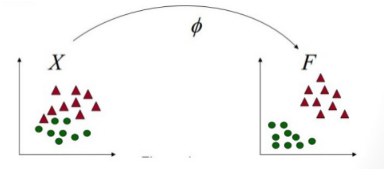
Kernel Trick

- Consider again moving window classification or regression.
- Note that $\boldsymbol{x} \in S(\boldsymbol{x}_*, h)$ if and only if $\|\boldsymbol{x} - \boldsymbol{x}_*\| \leq h$.
- The Euclidean distance in the input space is

$$\|\boldsymbol{x} - \boldsymbol{x}_*\| = \sqrt{(\boldsymbol{x} - \boldsymbol{x}_*)^T(\boldsymbol{x} - \boldsymbol{x}_*)} = \sqrt{\boldsymbol{x}^T\boldsymbol{x} + \boldsymbol{x}_*^T\boldsymbol{x}_* - 2\boldsymbol{x}^T\boldsymbol{x}_*}$$

- The Euclidean distance in the (**hopefully more convenient**) feature space

$$\begin{aligned} \|\boldsymbol{\phi}(\boldsymbol{x}) - \boldsymbol{\phi}(\boldsymbol{x}_*)\| &= \sqrt{\boldsymbol{\phi}(\boldsymbol{x}^T)\boldsymbol{\phi}(\boldsymbol{x}) + \boldsymbol{\phi}(\boldsymbol{x}_*^T)\boldsymbol{\phi}(\boldsymbol{x}_*) - 2\boldsymbol{\phi}(\boldsymbol{x}^T)\boldsymbol{\phi}(\boldsymbol{x}_*)} \\ &= \sqrt{k(\boldsymbol{x},\boldsymbol{x}) + k(\boldsymbol{x}_*,\boldsymbol{x}_*) - 2k(\boldsymbol{x},\boldsymbol{x}_*)} \end{aligned}$$



- For many (but not all) kernels, it holds that $k(\boldsymbol{x},\boldsymbol{x}) = k(\boldsymbol{x}_*,\boldsymbol{x}_*) =$ constant and, thus, the kernel specifies how close two points are **in the feature space**.

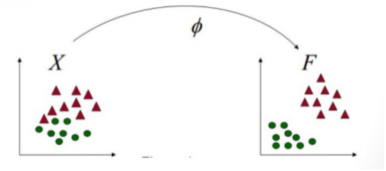
Kernel Trick

- Consider again moving window classification or regression.
- Note that $\boldsymbol{x} \in S(\boldsymbol{x}_*, h)$ if and only if $\|\boldsymbol{x} - \boldsymbol{x}_*\| \le h$.
- The Euclidean distance in the input space is

$$\|\boldsymbol{x} - \boldsymbol{x}_*\| = \sqrt{(\boldsymbol{x} - \boldsymbol{x}_*)^T(\boldsymbol{x} - \boldsymbol{x}_*)} = \sqrt{\boldsymbol{x}^T\boldsymbol{x} + \boldsymbol{x}_*^T\boldsymbol{x}_* - 2\boldsymbol{x}^T\boldsymbol{x}_*}$$

- The Euclidean distance in the (**hopefully more convenient**) feature space

$$\begin{aligned}\|\phi(\boldsymbol{x}) - \phi(\boldsymbol{x}_*)\| &= \sqrt{\phi(\boldsymbol{x}^T)\phi(\boldsymbol{x}) + \phi(\boldsymbol{x}_*^T)\phi(\boldsymbol{x}_*) - 2\phi(\boldsymbol{x}^T)\phi(\boldsymbol{x}_*)} \\ &= \sqrt{k(\boldsymbol{x},\boldsymbol{x}) + k(\boldsymbol{x}_*,\boldsymbol{x}_*) - 2k(\boldsymbol{x},\boldsymbol{x}_*)}\end{aligned}$$



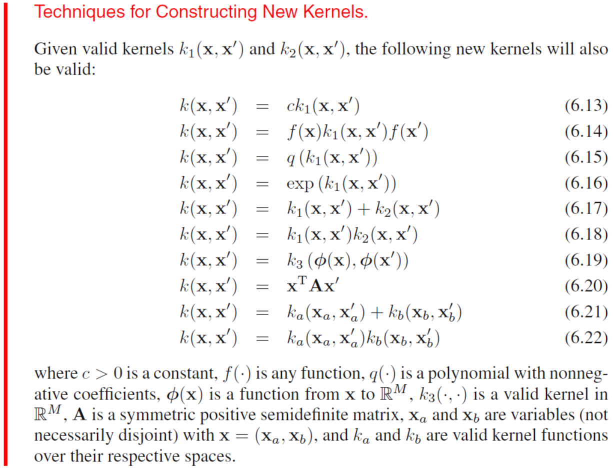
- For many (but not all) kernels, it holds that $k(\boldsymbol{x},\boldsymbol{x}) = k(\boldsymbol{x}_*,\boldsymbol{x}_*) =$ constant and, thus, the kernel specifies how close two points are **in the feature space**.
- Note that we **do not** compute $\phi(\boldsymbol{x})$ or $\phi(\boldsymbol{x}_*)$. Since the feature space only enters the distance via inner products, we design/select the inner product instead of the feature space. This is the **kernel trick**.

Kernel Trick

- Two alternatives for building $k(\boldsymbol{x}, \boldsymbol{x}')$:
  - Choose a convenient $\boldsymbol{\phi}(\boldsymbol{x})$ and let $k(\boldsymbol{x}, \boldsymbol{x}') = \boldsymbol{\phi}(\boldsymbol{x})^T \boldsymbol{\phi}(\boldsymbol{x}')$.
  - Build it from existing kernel functions as follows.

Kernel Trick

- Two alternatives for building $k(\boldsymbol{x}, \boldsymbol{x}')$:
  - Choose a convenient $\boldsymbol{\phi}(\boldsymbol{x})$ and let $k(\boldsymbol{x}, \boldsymbol{x}') = \boldsymbol{\phi}(\boldsymbol{x})^T \boldsymbol{\phi}(\boldsymbol{x}')$.
  - Build it from existing kernel functions as follows.



Techniques for Constructing New Kernels.

Given valid kernels $k_1(\mathbf{x}, \mathbf{x}')$ and $k_2(\mathbf{x}, \mathbf{x}')$, the following new kernels will also be valid:

$$
\begin{aligned}
k(\mathbf{x}, \mathbf{x}') &= c k_1(\mathbf{x}, \mathbf{x}') && (6.13)\\
k(\mathbf{x}, \mathbf{x}') &= f(\mathbf{x}) k_1(\mathbf{x}, \mathbf{x}') f(\mathbf{x}') && (6.14)\\
k(\mathbf{x}, \mathbf{x}') &= q\left(k_1(\mathbf{x}, \mathbf{x}')\right) && (6.15)\\
k(\mathbf{x}, \mathbf{x}') &= \exp\left(k_1(\mathbf{x}, \mathbf{x}')\right) && (6.16)\\
k(\mathbf{x}, \mathbf{x}') &= k_1(\mathbf{x}, \mathbf{x}') + k_2(\mathbf{x}, \mathbf{x}') && (6.17)\\
k(\mathbf{x}, \mathbf{x}') &= k_1(\mathbf{x}, \mathbf{x}') k_2(\mathbf{x}, \mathbf{x}') && (6.18)\\
k(\mathbf{x}, \mathbf{x}') &= k_3\left(\boldsymbol{\phi}(\mathbf{x}), \boldsymbol{\phi}(\mathbf{x}')\right) && (6.19)\\
k(\mathbf{x}, \mathbf{x}') &= \mathbf{x}^{\mathrm{T}} \mathbf{A} \mathbf{x}' && (6.20)\\
k(\mathbf{x}, \mathbf{x}') &= k_a(\mathbf{x}_a, \mathbf{x}_a') + k_b(\mathbf{x}_b, \mathbf{x}_b') && (6.21)\\
k(\mathbf{x}, \mathbf{x}') &= k_a(\mathbf{x}_a, \mathbf{x}_a') k_b(\mathbf{x}_b, \mathbf{x}_b') && (6.22)
\end{aligned}
$$

where $c > 0$ is a constant, $f(\cdot)$ is any function, $q(\cdot)$ is a polynomial with nonnegative coefficients, $\boldsymbol{\phi}(\mathbf{x})$ is a function from $\mathbf{x}$ to $\mathbb{R}^M$, $k_3(\cdot, \cdot)$ is a valid kernel in $\mathbb{R}^M$, $\mathbf{A}$ is a symmetric positive semidefinite matrix, $\mathbf{x}_a$ and $\mathbf{x}_b$ are variables (not necessarily disjoint) with $\mathbf{x} = (\mathbf{x}_a, \mathbf{x}_b)$, and k_a and k_b are valid kernel functions over their respective spaces.

Kernel Ridge Regression

- Ridge regression = linear regression + L^2 regularization to avoid overfitting.

$$\widehat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} \frac{1}{n}\|\mathbf{X}\boldsymbol{\theta} - \mathbf{y}\|_2^2 + \lambda\|\boldsymbol{\theta}\|_2^2. \tag{5.23}$$

Kernel Ridge Regression

- Ridge regression = linear regression + L^2 regularization to avoid overfitting.

$$\widehat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} \frac{1}{n} \|\mathbf{X}\boldsymbol{\theta} - \mathbf{y}\|_2^2 + \lambda \|\boldsymbol{\theta}\|_2^2. \tag{5.23}$$

- As linear regression, ridge regression has a closed-form solution.

$$\widehat{\boldsymbol{\theta}} = (\mathbf{X}^\mathsf{T}\mathbf{X} + n\lambda \mathbf{I}_{p+1})^{-1} \mathbf{X}^\mathsf{T}\mathbf{y}. \tag{5.25}$$

Kernel Ridge Regression

- Ridge regression = linear regression + L^2 regularization to avoid overfitting.

$$\widehat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} \frac{1}{n}\|\mathbf{X}\boldsymbol{\theta} - \mathbf{y}\|_2^2 + \lambda\|\boldsymbol{\theta}\|_2^2. \qquad (5.23)$$

- As linear regression, ridge regression has a closed-form solution.

$$\widehat{\boldsymbol{\theta}} = (\mathbf{X}^\mathsf{T}\mathbf{X} + n\lambda\mathbf{I}_{p+1})^{-1}\mathbf{X}^\mathsf{T}\mathbf{y}. \qquad (5.25)$$

- Kernel ridge regression = ridge regression in the feature space defined by the mapping $\phi(\cdot)$ rather than in the input space.

$$\widehat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} \frac{1}{n}\sum_{i=1}^{n}\Big(\underbrace{\boldsymbol{\theta}^\mathsf{T}\boldsymbol{\phi}(\mathbf{x}_i)}_{\widehat{y}(\mathbf{x}_i)} - y_i\Big)^2 + \lambda\|\boldsymbol{\theta}\|_2^2 = (\boldsymbol{\Phi}(\mathbf{X})^\mathsf{T}\boldsymbol{\Phi}(\mathbf{X}) + n\lambda\mathbf{I})^{-1}\boldsymbol{\Phi}(\mathbf{X})^\mathsf{T}\mathbf{y}, \qquad (8.4a)$$

Kernel Ridge Regression

- Ridge regression = linear regression + L^2 regularization to avoid overfitting.

$$\widehat{\theta} = \arg\min_{\theta} \frac{1}{n}\|\mathbf{X}\theta - \mathbf{y}\|_2^2 + \lambda\|\theta\|_2^2. \tag{5.23}$$

- As linear regression, ridge regression has a closed-form solution.

$$\widehat{\theta} = (\mathbf{X}^\mathsf{T}\mathbf{X} + n\lambda\mathbf{I}_{p+1})^{-1}\mathbf{X}^\mathsf{T}\mathbf{y}. \tag{5.25}$$

- Kernel ridge regression = ridge regression in the feature space defined by the mapping $\phi(\cdot)$ rather than in the input space.

$$\widehat{\theta} = \arg\min_{\theta} \frac{1}{n}\sum_{i=1}^{n}\Big(\underbrace{\theta^\mathsf{T}\phi(\mathbf{x}_i)}_{\widehat{y}(\mathbf{x}_i)} - y_i\Big)^2 + \lambda\|\theta\|_2^2 = (\mathbf{\Phi}(\mathbf{X})^\mathsf{T}\mathbf{\Phi}(\mathbf{X}) + n\lambda\mathbf{I})^{-1}\mathbf{\Phi}(\mathbf{X})^\mathsf{T}\mathbf{y}, \tag{8.4a}$$

- Computing $\widehat{\boldsymbol{\theta}}$ can be **problematic** for some feature spaces, since they can be high or even infinite dimensional. However, $\widehat{\boldsymbol{\theta}}$ is just needed for prediction.

$$\widehat{y}(\mathbf{x}_\star) = \underbrace{\widehat{\theta}^\mathsf{T}}_{1\times d}\underbrace{\phi(\mathbf{x}_\star)}_{d\times 1} = \underbrace{\mathbf{y}^\mathsf{T}}_{1\times n}\underbrace{(\mathbf{\Phi}(\mathbf{X})\mathbf{\Phi}(\mathbf{X})^\mathsf{T} + n\lambda\mathbf{I})^{-1}}_{n\times n}\underbrace{\mathbf{\Phi}(\mathbf{X})\phi(\mathbf{x}_\star)}_{n\times 1}.$$

Kernel Ridge Regression

- Ridge regression = linear regression + L^2 regularization to avoid overfitting.

$$\widehat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} \frac{1}{n} \|\mathbf{X}\boldsymbol{\theta} - \mathbf{y}\|_2^2 + \lambda \|\boldsymbol{\theta}\|_2^2. \qquad (5.23)$$

- As linear regression, ridge regression has a closed-form solution.

$$\widehat{\boldsymbol{\theta}} = (\mathbf{X}^\mathsf{T}\mathbf{X} + n\lambda \mathbf{I}_{p+1})^{-1}\mathbf{X}^\mathsf{T}\mathbf{y}. \qquad (5.25)$$

- Kernel ridge regression = ridge regression in the feature space defined by the mapping $\boldsymbol{\phi}(\cdot)$ rather than in the input space.

$$\widehat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} \frac{1}{n} \sum_{i=1}^{n} \Big(\underbrace{\boldsymbol{\theta}^\mathsf{T}\boldsymbol{\phi}(\mathbf{x}_i)}_{\widehat{y}(\mathbf{x}_i)} - y_i\Big)^2 + \lambda \|\boldsymbol{\theta}\|_2^2 = (\boldsymbol{\Phi}(\mathbf{X})^\mathsf{T}\boldsymbol{\Phi}(\mathbf{X}) + n\lambda\mathbf{I})^{-1}\boldsymbol{\Phi}(\mathbf{X})^\mathsf{T}\mathbf{y}, \qquad (8.4a)$$

- Computing $\widehat{\boldsymbol{\theta}}$ can be **problematic** for some feature spaces, since they can be high or even infinite dimensional. However, $\widehat{\boldsymbol{\theta}}$ is just needed for prediction.

$$\widehat{y}(\mathbf{x}_\star) = \underbrace{\widehat{\boldsymbol{\theta}}^\mathsf{T}}_{1\times d} \underbrace{\boldsymbol{\phi}(\mathbf{x}_\star)}_{d\times 1} = \underbrace{\mathbf{y}^\mathsf{T}}_{1\times n} \underbrace{(\boldsymbol{\Phi}(\mathbf{X})\boldsymbol{\Phi}(\mathbf{X})^\mathsf{T} + n\lambda\mathbf{I})^{-1}}_{n\times n} \underbrace{\boldsymbol{\Phi}(\mathbf{X})\boldsymbol{\phi}(\mathbf{x}_\star)}_{n\times 1}.$$

- Since the feature space only enters the prediction via inner products, we design/select the inner product instead of the feature space. This is the **kernel trick**.

$$\widehat{y}(\mathbf{x}_\star) = \underbrace{\mathbf{y}^\mathsf{T}}_{1\times n} \underbrace{(\boldsymbol{K}(\mathbf{X},\mathbf{X}) + n\lambda\mathbf{I})^{-1}}_{n\times n} \underbrace{\boldsymbol{K}(\mathbf{X},\mathbf{x}_\star)}_{n\times 1}, \qquad (8.12a)$$

Kernel Ridge Regression

- Ridge regression = linear regression + L^2 regularization to avoid overfitting.

$$\widehat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} \frac{1}{n}\|\mathbf{X}\boldsymbol{\theta} - \mathbf{y}\|_2^2 + \lambda\|\boldsymbol{\theta}\|_2^2. \qquad (5.23)$$

- As linear regression, ridge regression has a closed-form solution.

$$\widehat{\boldsymbol{\theta}} = (\mathbf{X}^\mathsf{T}\mathbf{X} + n\lambda\mathbf{I}_{p+1})^{-1}\mathbf{X}^\mathsf{T}\mathbf{y}. \qquad (5.25)$$

- Kernel ridge regression = ridge regression in the feature space defined by the mapping $\phi(\cdot)$ rather than in the input space.

$$\widehat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} \frac{1}{n}\sum_{i=1}^{n}\Big(\underbrace{\boldsymbol{\theta}^\mathsf{T}\boldsymbol{\phi}(\mathbf{x}_i)}_{\widehat{y}(\mathbf{x}_i)} - y_i\Big)^2 + \lambda\|\boldsymbol{\theta}\|_2^2 = (\boldsymbol{\Phi}(\mathbf{X})^\mathsf{T}\boldsymbol{\Phi}(\mathbf{X}) + n\lambda\mathbf{I})^{-1}\boldsymbol{\Phi}(\mathbf{X})^\mathsf{T}\mathbf{y}, \qquad (8.4a)$$

- Computing $\widehat{\boldsymbol{\theta}}$ can be **problematic** for some feature spaces, since they can be high or even infinite dimensional. However, $\widehat{\boldsymbol{\theta}}$ is just needed for prediction.

$$\widehat{y}(\mathbf{x}_\star) = \underbrace{\widehat{\boldsymbol{\theta}}^\mathsf{T}}_{1\times d}\underbrace{\boldsymbol{\phi}(\mathbf{x}_\star)}_{d\times 1} = \underbrace{\mathbf{y}^\mathsf{T}}_{1\times n}\underbrace{(\boldsymbol{\Phi}(\mathbf{X})\boldsymbol{\Phi}(\mathbf{X})^\mathsf{T} + n\lambda\mathbf{I})^{-1}}_{n\times n}\underbrace{\boldsymbol{\Phi}(\mathbf{X})\boldsymbol{\phi}(\mathbf{x}_\star)}_{n\times 1}.$$

- Since the feature space only enters the prediction via inner products, we design/select the inner product instead of the feature space. This is the **kernel trick**.

$$\widehat{y}(\mathbf{x}_\star) = \underbrace{\mathbf{y}^\mathsf{T}}_{1\times n}\underbrace{(\boldsymbol{K}(\mathbf{X},\mathbf{X}) + n\lambda\mathbf{I})^{-1}}_{n\times n}\underbrace{\boldsymbol{K}(\mathbf{X},\mathbf{x}_\star)}_{n\times 1}, \qquad (8.12a)$$

- Note that nothing prevents the feature space from being high or even infinite dimensional, because we do not compute $\boldsymbol{\Phi}(\boldsymbol{X})$ or $\phi(\boldsymbol{x}_*)$.

Kernel Ridge Regression

- Ridge regression = linear regression + L^2 regularization to avoid overfitting.

$$\widehat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} \frac{1}{n}\|\mathbf{X}\boldsymbol{\theta} - \mathbf{y}\|_2^2 + \lambda\|\boldsymbol{\theta}\|_2^2. \tag{5.23}$$

- As linear regression, ridge regression has a closed-form solution.

$$\widehat{\boldsymbol{\theta}} = (\mathbf{X}^\mathsf{T}\mathbf{X} + n\lambda\mathbf{I}_{p+1})^{-1}\mathbf{X}^\mathsf{T}\mathbf{y}. \tag{5.25}$$

- Kernel ridge regression = ridge regression in the feature space defined by the mapping $\phi(\cdot)$ rather than in the input space.

$$\widehat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} \frac{1}{n}\sum_{i=1}^{n}\Big(\underbrace{\boldsymbol{\theta}^\mathsf{T}\boldsymbol{\phi}(\mathbf{x}_i)}_{\widehat{y}(\mathbf{x}_i)} - y_i\Big)^2 + \lambda\|\boldsymbol{\theta}\|_2^2 = (\boldsymbol{\Phi}(\mathbf{X})^\mathsf{T}\boldsymbol{\Phi}(\mathbf{X}) + n\lambda\mathbf{I})^{-1}\boldsymbol{\Phi}(\mathbf{X})^\mathsf{T}\mathbf{y}, \tag{8.4a}$$

- Computing $\widehat{\boldsymbol{\theta}}$ can be **problematic** for some feature spaces, since they can be high or even infinite dimensional. However, $\widehat{\boldsymbol{\theta}}$ is just needed for prediction.

$$\widehat{y}(\mathbf{x}_\star) = \underbrace{\widehat{\boldsymbol{\theta}}^\mathsf{T}}_{1\times d}\underbrace{\boldsymbol{\phi}(\mathbf{x}_\star)}_{d\times 1} = \underbrace{\mathbf{y}^\mathsf{T}}_{1\times n}\underbrace{(\boldsymbol{\Phi}(\mathbf{X})\boldsymbol{\Phi}(\mathbf{X})^\mathsf{T} + n\lambda\mathbf{I})^{-1}}_{n\times n}\underbrace{\boldsymbol{\Phi}(\mathbf{X})\boldsymbol{\phi}(\mathbf{x}_\star)}_{n\times 1}.$$

- Since the feature space only enters the prediction via inner products, we design/select the inner product instead of the feature space. This is the **kernel trick**.

$$\widehat{y}(\mathbf{x}_\star) = \underbrace{\mathbf{y}^\mathsf{T}}_{1\times n}\underbrace{(\boldsymbol{K}(\mathbf{X},\mathbf{X}) + n\lambda\mathbf{I})^{-1}}_{n\times n}\underbrace{\boldsymbol{K}(\mathbf{X},\mathbf{x}_\star)}_{n\times 1}, \tag{8.12a}$$

- Note that nothing prevents the feature space from being high or even infinite dimensional, because we do not compute $\boldsymbol{\Phi}(\boldsymbol{X})$ or $\phi(\boldsymbol{x}_*)$.
- Is this a parametric model?
  - No, it is a kernel weighted average of the first two terms in the rhs of Equation 8.12a

Kernel Ridge Regression

Example 8.2: Linear regression with kernels

We consider again the car stopping distance problem from Example 2.2, and apply kernel ridge regression to it. We use $\lambda = 0.01$ here, and explore what happens when using different kernels.

We start, in the left panel in Figure 8.3, using the squared exponential kernel with $\ell = 1$ (blue line). We see that it does not really interpolate well between the data points, whereas $\ell = 3$ (green line) gives a more sensible behavior. (We could select ℓ using cross validation, but we do not pursue that any further here.)

It is interesting to note that prediction reverts to zero when extrapolating beyond the range of the training data. This is in fact a general property of the squared exponential kernel, as well as many other commonly used kernel. The reason for this behavior is that the kernel $\kappa(\mathbf{x}, \mathbf{x}')$ by construction drops to zero as the distance between $\mathbf{x}$ and $\mathbf{x}'$ increases. Intuitively, this means that the resulting predictions are based on local interpolation, and as we extrapolate far beyond the range of the training data the method will revert to a "default prediction" of zero. This can be seen from (8.14b)—if $\mathbf{x}_\star$ is far from the training data points, then all elements of the vector $\boldsymbol{K}(\mathbf{X}, \mathbf{x}_\star)$ will be close to zero (for a kernel with the aforementioned property) and so will the resulting prediction.

We have previously seen that this data, to some extent, follows a quadratic function. As we will discuss in Section 8.4, the sum of two kernels is another kernel. In the right panel in Figure 8.3 we therefore try using the sum of the squared exponential kernel (with $\ell = 3$) *and* the polynomial kernel of degree 2 ($d = 3$) (red line). As a reference we also include kernel ridge regression with only the polynomial kernel of degree 2 ($d = 3$) (dashed blue line; equivalent to L^2-regularized polynomial regression). The combined kernel gives a more flexible model than only a quadratic function, but it also (for this example) seems to extrapolate better than only using the squared exponential kernel.

This can be understood by noting that the polynomial kernel is *not local* (in the sense discussed above). That, it does not drop to zero for test data points that are far from the training data. Instead it corresponds to a polynomial trend and the predictions will follow this trend when extrapolating. Note that the two kernels considered here result in very similar extrapolation. The reason for this is that the squared exponential component of the combined kernel will only "be active" when interpolating. For extrapolation the combined kernel will thus revert to using only the polynomial component.

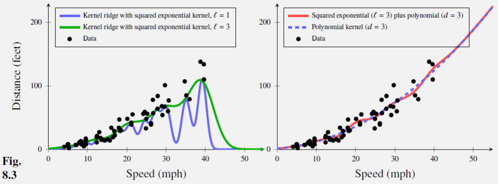


Fig. 8.3

By studying Figure 8.3, we can see that kernel ridge regression is a very flexible model, and the result is highly dependent on the choice of kernel. As we will stress throughout this (and the next) chapter, the kernel is indeed a crucial choice for the machine learning engineer when using kernel methods.

Summary

- Nonparametric Kernel Methods
  - Histogram, Moving Window, and Kernel Classification
  - Histogram, Moving Window, and Kernel Regression
- Kernel Trick
- Parametric Kernel Methods
  - Kernel Ridge Regression