Cognitive Algorithms
Lecture 4

Kernel Methods

Klaus-Robert Müller, Johannes Niediek,
Augustin Krause, Joanina Oltersdorff, Ken Schreiber

Technische Universität Berlin
Machine Learning Group

Linear regression

The most popular loss function to optimize $\boldsymbol{w}$
is the **least-square error** [Gauß, 1809; Legendre, 1805]

$$\mathcal{E}_{LSQ}(\boldsymbol{w}) = \sum_{i=1}^{N} (y_i - \boldsymbol{w}^\top X_i)^2$$

C. F. Gauß (1777–1855)

A. M. Legendre (1752–1833)

Gauss-Markov Theorem

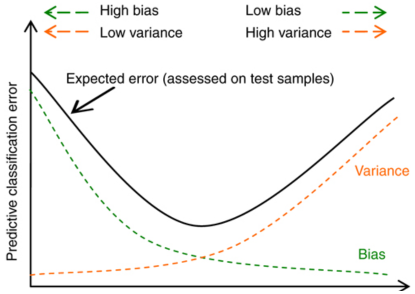
Under the model assumption $y = \boldsymbol{w}^\top \cdot \boldsymbol{x} + \epsilon$ with uncorrelated noise ϵ, our ordinary least squares estimator $\hat{\boldsymbol{w}} = (XX^\top)^{-1}X\boldsymbol{y}$ is the Best Linear Unbiased Estimator (BLUE), that is, the minimum variance unbiased estimator that is linear in the y.

Gauss-Markov Theorem

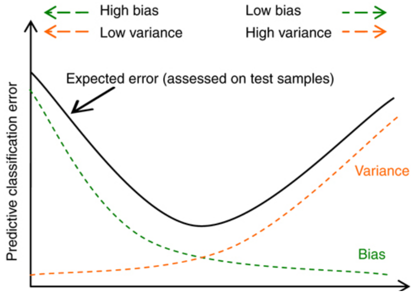
Under the model assumption $y = \boldsymbol{w}^\top \cdot \boldsymbol{x} + \epsilon$ with uncorrelated noise ϵ, our ordinary least squares estimator $\hat{\boldsymbol{w}} = (XX^\top)^{-1}X\boldsymbol{y}$ is the Best Linear Unbiased Estimator (BLUE), that is, the minimum variance unbiased estimator that is linear in the y.

But in some cases biased estimators with lower variance might be more suitable.

Bias-variance trade-off



Bias-variance trade-off



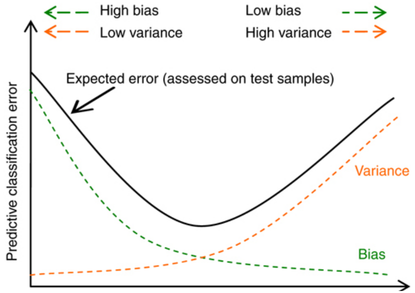
Model Complexity

optimum

$\longleftrightarrow$

underfitting overfitting

Bias-variance trade-off



Model Complexity

optimum

$\longleftrightarrow$

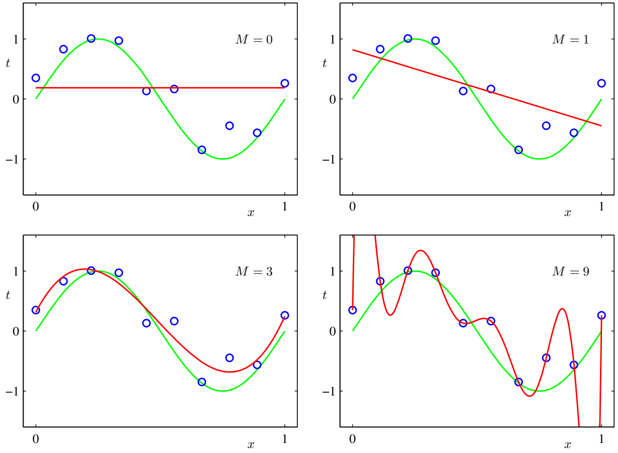
underfitting overfitting

$$\mathbb{E}_{\mathcal{D}}\left[\{y(\mathbf{x};\mathcal{D})-h(\mathbf{x})\}^2\right]$$
$$= \underbrace{\{\mathbb{E}_{\mathcal{D}}[y(\mathbf{x};\mathcal{D})]-h(\mathbf{x})\}^2}_{(\text{bias})^2} + \underbrace{\mathbb{E}_{\mathcal{D}}\left[\{y(\mathbf{x};\mathcal{D})-\mathbb{E}_{\mathcal{D}}[y(\mathbf{x};\mathcal{D})]\}^2\right]}_{\text{variance}}. \quad (3.40)$$

[Bishop, 2007, Section 3.2]

Example: Linear regression for a polynomial function

$$\hat{y}(x) = w_0 + w_1 \cdot x^1 + \ldots + w_M \cdot x^M$$

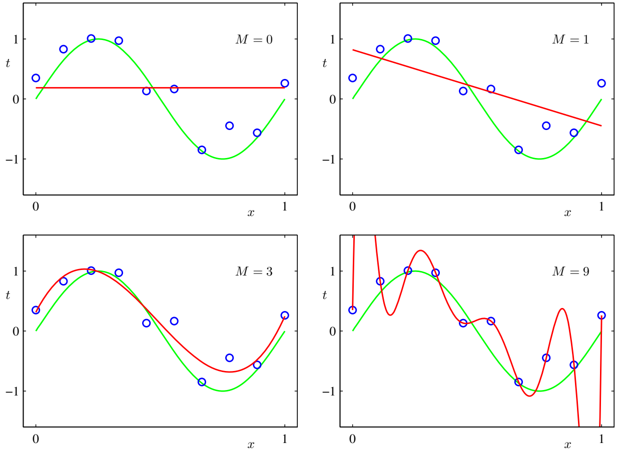


[Bishop, 2007]

Example: Linear regression for a polynomial function

$$\hat{y}(x) = w_0 + w_1 \cdot x^1 + \ldots + w_M \cdot x^M$$

We use the basis function $\phi_M(x) = (x^0, x^1, \ldots, x^M)$, which has a $(M+1)$-dimensional feature space $\mathcal{F}$



[Bishop, 2007]

Ridge regression

Control the complexity of the solution $\boldsymbol{w}$.

This is done by constraining the norm of $\boldsymbol{w}$,

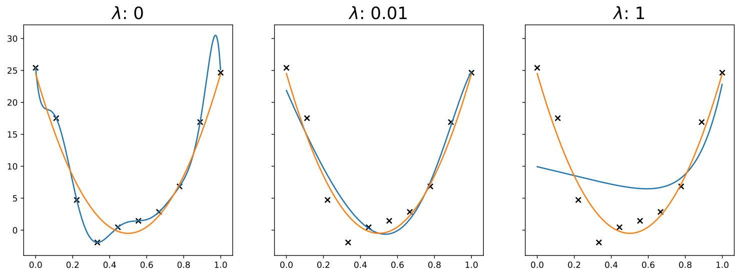
$$\mathcal{E}_{RR}(\boldsymbol{w}) = \|\boldsymbol{y} - \boldsymbol{w}^\top X\|^2 + \lambda\|\boldsymbol{w}\|^2$$

Ridge regression

Control the complexity of the solution $\boldsymbol{w}$.

This is done by constraining the norm of $\boldsymbol{w}$,

$$\mathcal{E}_{RR}(\boldsymbol{w}) = \|\boldsymbol{y} - \boldsymbol{w}^\top X\|^2 + \lambda \|\boldsymbol{w}\|^2$$



Ridge regression

Computing the derivative with respect to $\boldsymbol{w}$ yields

$$\frac{\partial \mathcal{E}_{RR}(\boldsymbol{w})}{\partial \boldsymbol{w}} = -2X\boldsymbol{y}^\top + 2XX^\top \boldsymbol{w} + \lambda 2\boldsymbol{w}.$$

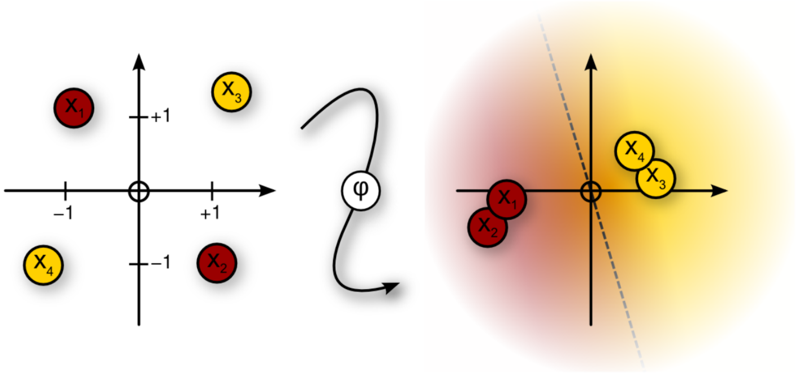
Setting the gradient to zero and rearranging terms the optimal $\boldsymbol{w}$ is

$$\begin{aligned} 2XX^\top \boldsymbol{w} + \lambda 2\boldsymbol{w} =& 2X\boldsymbol{y}^\top \\ (XX^\top + \lambda I)\boldsymbol{w} =& X\boldsymbol{y}^\top \\ \boldsymbol{w} =& (XX^\top + \lambda I)^{-1} X\boldsymbol{y}^\top \end{aligned}$$

$\Rightarrow$ Biased estimator, but smaller variance

[Hoerl and Kennar, 1970; Tychonoff, 1943]

Kernelizing linear methods



[Jäkel et al., 2009]

1. Map the data into a (high dimensional) feature space, $\boldsymbol{x} \mapsto \varphi(\boldsymbol{x})$
2. Look for linear relations/decision boundaries in the feature space

What is a kernel?

Given φ, a mapping to a feature space $\mathcal{F}$ (equipped with a scalar product),

$$\begin{aligned}\varphi : \mathcal{X} &\to \mathcal{F}\\ x &\mapsto \varphi(x),\end{aligned}$$

we define the *kernel* corresponding to φ as the function

$$\begin{aligned}k &: \mathcal{X} \times \mathcal{X} \to \mathbb{R}\\ k(x_i, x_j) &= \varphi(x_i)^T \cdot \varphi(x_j).\end{aligned}$$

The kernel trick (also called kernel substitution)

For any algorithm that can be formulated such that the input vector $\boldsymbol{x}$ enters only in terms of scalar products $\boldsymbol{x}^T \cdot \boldsymbol{x}'$, we can replace each scalar product by a kernel $k(\boldsymbol{x}, \boldsymbol{x}') = \varphi(\boldsymbol{x})^T \cdot \varphi(\boldsymbol{x}')$.

Why should we do that?

The kernel trick (also called kernel substitution)

For any algorithm that can be formulated such that the input vector $\boldsymbol{x}$ enters only in terms of scalar products $\boldsymbol{x}^T \cdot \boldsymbol{x}'$, we can replace each scalar product by a kernel $k(\boldsymbol{x}, \boldsymbol{x}') = \varphi(\boldsymbol{x})^T \cdot \varphi(\boldsymbol{x}')$.

Why should we do that?

For $\boldsymbol{x}, \boldsymbol{x}' \in \mathbb{R}^{d \times 1} \Rightarrow \boldsymbol{x}^T \cdot \boldsymbol{x}' \in \mathbb{R}^{1 \times 1}$ regardless of d.

The kernel trick (also called kernel substitution)

For any algorithm that can be formulated such that the input vector $\boldsymbol{x}$ enters only in terms of scalar products $\boldsymbol{x}^T \cdot \boldsymbol{x}'$, we can replace each scalar product by a kernel $k(\boldsymbol{x}, \boldsymbol{x}') = \varphi(\boldsymbol{x})^T \cdot \varphi(\boldsymbol{x}')$.

Why should we do that?

For $\boldsymbol{x}, \boldsymbol{x}' \in \mathbb{R}^{d\times 1} \Rightarrow \boldsymbol{x}^T \cdot \boldsymbol{x}' \in \mathbb{R}^{1\times 1}$ regardless of d.

For $\varphi(\boldsymbol{x}) \in \mathbb{R}^{\tilde{d}\times 1}$ instead of $\boldsymbol{x}$ with typically $\tilde{d} \gg d$

We can construct more complex (powerful) models.

By using kernel $k(\boldsymbol{x}, \boldsymbol{x}') \in \mathbb{R}^{1\times 1}$

We do not need to explicitly calculate high-dimensional $\varphi(\boldsymbol{x})$.

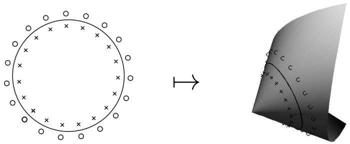
Example kernel

$$\varphi : \boldsymbol{x} = (x_1, x_2)^\top \mapsto \varphi(\boldsymbol{x}) = \begin{pmatrix} x_1^2 \\ \sqrt{2}x_1x_2 \\ x_2^2 \end{pmatrix}$$

Example kernel

$$\varphi : \boldsymbol{x} = (x_1, x_2)^\top \mapsto \varphi(\boldsymbol{x}) = \begin{pmatrix} x_1^2 \\ \sqrt{2} x_1 x_2 \\ x_2^2 \end{pmatrix}$$

Visualizing $\varphi(\boldsymbol{x})$



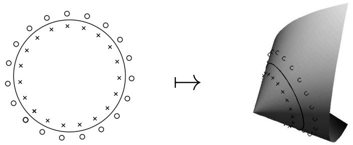
Example kernel

$$\varphi : \boldsymbol{x} = (x_1, x_2)^\top \mapsto \varphi(\boldsymbol{x}) = \begin{pmatrix} x_1^2 \\ \sqrt{2}x_1 x_2 \\ x_2^2 \end{pmatrix}$$

The corresponding kernel:

$$k(\boldsymbol{x}, \boldsymbol{y}) = \varphi(\boldsymbol{x})^\top \cdot \varphi(\boldsymbol{y})$$

Visualizing $\varphi(\boldsymbol{x})$



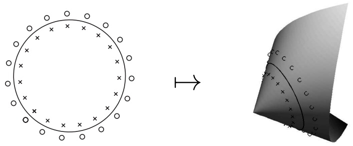
Example kernel

$$\varphi : \boldsymbol{x} = (x_1, x_2)^\top \mapsto \varphi(\boldsymbol{x}) = \begin{pmatrix} x_1^2 \\ \sqrt{2}x_1x_2 \\ x_2^2 \end{pmatrix}$$

The corresponding kernel:

$$\begin{aligned} k(\boldsymbol{x}, \boldsymbol{y}) &= \varphi(\boldsymbol{x})^\top \cdot \varphi(\boldsymbol{y}) \\ &= (x_1^2, \sqrt{2}x_1x_2, x_2^2) \cdot (y_1^2, \sqrt{2}y_1y_2, y_2^2)^T \end{aligned}$$

Visualizing $\varphi(\boldsymbol{x})$



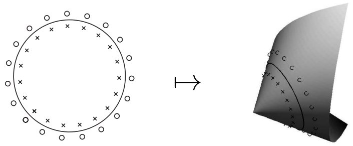
Example kernel

$$\varphi : \boldsymbol{x} = (x_1, x_2)^\top \mapsto \varphi(\boldsymbol{x}) = \begin{pmatrix} x_1^2 \\ \sqrt{2}x_1x_2 \\ x_2^2 \end{pmatrix}$$

The corresponding kernel:

$$\begin{aligned} k(\boldsymbol{x}, \boldsymbol{y}) &= \varphi(\boldsymbol{x})^\top \cdot \varphi(\boldsymbol{y}) \\ &= (x_1^2, \sqrt{2}x_1x_2, x_2^2) \cdot (y_1^2, \sqrt{2}y_1y_2, y_2^2)^T \\ &= x_1^2y_1^2 + 2x_1x_2y_1y_2 + x_2^2y_2^2 \\ &= (x_1y_1 + x_2y_2)^2 = (\boldsymbol{x}^\top\boldsymbol{y})^2 \end{aligned}$$

Visualizing $\varphi(\boldsymbol{x})$



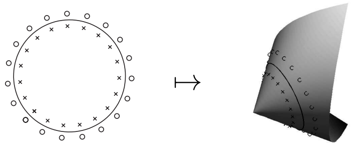
Example kernel

$$\varphi : \boldsymbol{x} = (x_1, x_2)^\top \mapsto \varphi(\boldsymbol{x}) = \begin{pmatrix} x_1^2 \\ \sqrt{2}x_1x_2 \\ x_2^2 \end{pmatrix}$$

The corresponding kernel:

$$\begin{aligned} k(\boldsymbol{x}, \boldsymbol{y}) &= \varphi(\boldsymbol{x})^\top \cdot \varphi(\boldsymbol{y}) \\ &= (x_1^2, \sqrt{2}x_1x_2, x_2^2) \cdot (y_1^2, \sqrt{2}y_1y_2, y_2^2)^T \\ &= x_1^2y_1^2 + 2x_1x_2y_1y_2 + x_2^2y_2^2 \\ &= (x_1y_1 + x_2y_2)^2 = (\boldsymbol{x}^\top \boldsymbol{y})^2 \end{aligned}$$

Visualizing $\varphi(\boldsymbol{x})$



With $k(\cdot, \cdot)$ we implicitly work in $\mathbb{R}^3$, but only operate in $\mathbb{R}^2$

Definition (Positive semi-definite symmetric kernels)

A kernel

$$k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$$

is said to be positive semi-definite symmetric

if for any $\{x_1, \ldots, x_n\} \subseteq \mathcal{X}$, the matrix $K = [k(x_i, x_j)]_{ij} \in \mathbb{R}^{n \times n}$ is symmetric positive semi-definite. [1]

A matrix A is called *symmetric* if $A = A^\intercal$.
A matrix A is called *positive semi-definite* if $x^\intercal A x \geq 0 \quad \forall\, x$.
For a symmetric matrix A: A is positive semi-definite if all eigenvalues of A are non-negative.

[1]For ease of notation we may use $K(X, X') = [K(\boldsymbol{x}_i, \boldsymbol{x}'_j)]_{ij}$ for describing the matrix of the kernel-function evaluated on all sample-pairs. $K(X, X)$ is often called the *Gram matrix of X*.

Mercer's Theorem [Mercer, 1909]

(non-technical version)
If a kernel $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is positive semi-definite symmetric, then one can construct a feature space $\mathcal{F}$ with a scalar product and a map $\varphi : \mathcal{X} \to \mathcal{F}$ such that

$$k(x, x') = \varphi(x)^T \varphi(x').$$

Mercer's Theorem [Mercer, 1909]

(non-technical version)
If a kernel $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is positive semi-definite symmetric, then one can construct a feature space $\mathcal{F}$ with a scalar product and a map $\varphi : \mathcal{X} \to \mathcal{F}$ such that

$$k(x, x') = \varphi(x)^T \varphi(x').$$

- To see if your kernel is valid, show it is positive semi-definite symmetric!
- You can construct kernels from other kernels, e.g. by sum, product or exponentiation
- Alternatively show $k(x, x') = \varphi(x)^T \varphi(x')$

For a helpful explanation and proof of the theorem, see Shawe-Taylor and Cristianini [2004], Theorems 3.11 and 3.13 (available on ISIS).

Some popular kernel functions

Linear kernel

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

Some popular kernel functions

Linear kernel

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

Polynomial kernel

$$k(\boldsymbol{x}_i, \boldsymbol{x}_j) = (\boldsymbol{x}_i^\top \boldsymbol{x}_j + c)^p$$

Some popular kernel functions

Linear kernel

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

Polynomial kernel

$$k(\boldsymbol{x}_i, \boldsymbol{x}_j) = (\boldsymbol{x}_i^\top \boldsymbol{x}_j + c)^p$$

Gaussian kernel (radial basis function, more on this later)

$$k(\boldsymbol{x}_i, \boldsymbol{x}_j) = e^{\frac{\|x_i - x_j\|^2}{-2\sigma^2}}$$

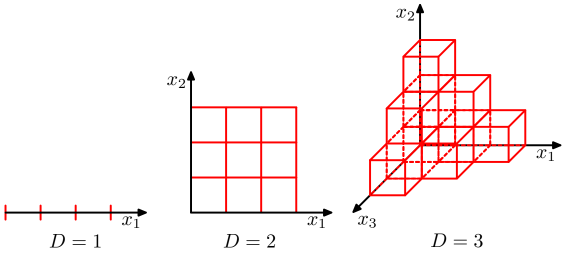
Note that we are never directly operating in the feature space $\mathcal{F}$!

The curse of dimensionality

A big problem with high dimensional features spaces:
When the dimensionality increases, the volume of the space increases so fast that the available data becomes sparse.

The curse of dimensionality

A big problem with high dimensional features spaces:
When the dimensionality increases, the volume of the space increases so fast that the available data becomes sparse.

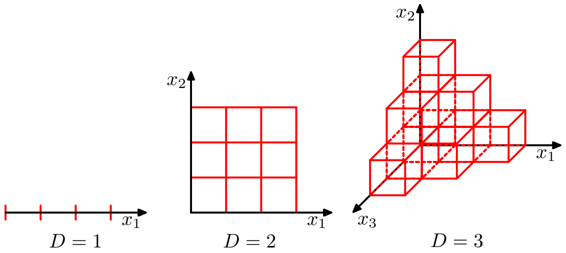


[Bishop, 2007]

The curse of dimensionality

A big problem with high dimensional features spaces:
When the dimensionality increases, the volume of the space increases so fast that the available data becomes sparse.

The amount of data needed for a reliable result often grows exponentially with the dimensionality.



[Bishop, 2007]

How do we find the optimal weight

$$\boldsymbol{w} \in \mathbb{R}^{\tilde{d}}?$$

Seems impossible with $\tilde{d} \gg n$.

Sven Sachsalber hunting for a needle in a haystack.

Representer Theorem [Kimeldorf and Wahba, 1971][2]

(non-technical version)
The minimizing function f^* of a **regularized** error function on some training data $\boldsymbol{x}_i$ can be written in terms of the kernel k as

$$f^*(\boldsymbol{x}) = \sum_{i=1}^{N} \alpha_i k(\boldsymbol{x}, \boldsymbol{x}_i) \, .$$

Note: For the model $f(x) = \boldsymbol{w}^\top \varphi(\boldsymbol{x})$, the Representer Theorem implies that $\boldsymbol{w}$ can be written as

$$\boldsymbol{w} = \sum_{i=1}^{N} \alpha_i \varphi(\boldsymbol{x}_i) \, .$$

[2] For background and proof, see e.g. `https://en.wikipedia.org/wiki/Representer_theorem`

Kernelizing algorithms

From before:

Kernel trick (kernel substitution)

For any algorithm that can be formulated such that the input vector $\boldsymbol{x}$ enters only in terms of scalar products $\boldsymbol{x}^T \cdot \boldsymbol{x}'$, we can replace each scalar product by a kernel $k(\boldsymbol{x}, \boldsymbol{x}') = \varphi(\boldsymbol{x})^T \cdot \varphi(\boldsymbol{x}')$.

Kernelizing algorithms

From before:

Kernel trick (kernel substitution)

For any algorithm that can be formulated such that the input vector $\boldsymbol{x}$ enters only in terms of scalar products $\boldsymbol{x}^T \cdot \boldsymbol{x}'$, we can replace each scalar product by a kernel $k(\boldsymbol{x}, \boldsymbol{x}') = \varphi(\boldsymbol{x})^T \cdot \varphi(\boldsymbol{x}')$.

So let's see an example!

- In the following we will kernelize *ridge regression*
- We will recast the problem into an equivalent dual representation (can be done with many linear models)

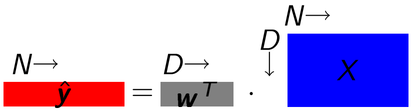
Recap: linear regression

The linear regression model in matrix notation

$$\hat{\boldsymbol{y}} = \boldsymbol{w}^\top X.$$

Linear regression minimizes the least-squares loss function

$$\mathcal{E}_{LSQ}(\boldsymbol{w}) = \sum_{i=1}^{n} (y_i - \boldsymbol{w}^\top \boldsymbol{x}_i)^2 = \|\boldsymbol{y} - \boldsymbol{w}^\top X\|^2$$



($\boldsymbol{y}$ is a row vector, $\boldsymbol{w}$ is a column vector.)

Recap: ridge regression

Linear ridge regression finds $\boldsymbol{w}$ that minimizes the prediction error under constraints on the norm $\|\boldsymbol{w}\|$.
We can write this term in several equivalent ways:

$$\begin{aligned}\mathcal{E}_{RR}(\boldsymbol{w}) &= \sum_n (y_n - \boldsymbol{w}^\top \boldsymbol{x}_n)^2 + \lambda \sum_d w_d^2 \\ &= ||\boldsymbol{y} - \boldsymbol{w}^\top X||^2 + \lambda ||\boldsymbol{w}||^2 \\ &= \boldsymbol{y}\boldsymbol{y}^\top - 2\boldsymbol{w}^\top X \boldsymbol{y}^\top + \boldsymbol{w}^\top X X^\top \boldsymbol{w} + \lambda \boldsymbol{w}^\top \boldsymbol{w}\end{aligned}$$

Recap: ridge regression

Linear ridge regression finds $\boldsymbol{w}$ that minimizes the prediction error under constraints on the norm $\|\boldsymbol{w}\|$.
We can write this term in several equivalent ways:

$$
\begin{aligned}
\mathcal{E}_{RR}(\boldsymbol{w}) = & \sum_n (y_n - \boldsymbol{w}^\top \boldsymbol{x}_n)^2 + \lambda \sum_d w_d^2 \\
= & ||\boldsymbol{y} - \boldsymbol{w}^\top X||^2 + \lambda ||\boldsymbol{w}||^2 \\
= & \boldsymbol{y}\boldsymbol{y}^\top - 2\boldsymbol{w}^\top X \boldsymbol{y}^\top + \boldsymbol{w}^\top X X^\top \boldsymbol{w} + \lambda \boldsymbol{w}^\top \boldsymbol{w}
\end{aligned}
$$

Kernel trick: we can use kernels if algorithm depends on training data X and test sample $\boldsymbol{x}$ only through scalar products.

From linear to kernel ridge regression

$$\mathcal{E}_{RR}(\boldsymbol{w}) = \boldsymbol{y}\boldsymbol{y}^\top - 2\boldsymbol{w}^\top X\boldsymbol{y}^\top + \boldsymbol{w}^\top XX^\top\boldsymbol{w} + \lambda\boldsymbol{w}^\top\boldsymbol{w}$$

From linear to kernel ridge regression

$$\mathcal{E}_{RR}(\boldsymbol{w}) = \boldsymbol{y}\boldsymbol{y}^\top - 2\boldsymbol{w}^\top X\boldsymbol{y}^\top + \boldsymbol{w}^\top XX^\top\boldsymbol{w} + \lambda\boldsymbol{w}^\top\boldsymbol{w}$$

Computing the derivative with respect to $\boldsymbol{w}$ yields

$$\frac{\partial\mathcal{E}_{RR}(\boldsymbol{w})}{\partial\boldsymbol{w}} = -2X\boldsymbol{y}^T + 2XX^T\boldsymbol{w} + 2\lambda\boldsymbol{w}$$

From linear to kernel ridge regression

$$\mathcal{E}_{RR}(\boldsymbol{w}) = \boldsymbol{y}\boldsymbol{y}^\top - 2\boldsymbol{w}^\top X\boldsymbol{y}^\top + \boldsymbol{w}^\top XX^\top\boldsymbol{w} + \lambda\boldsymbol{w}^\top\boldsymbol{w}$$

Computing the derivative with respect to $\boldsymbol{w}$ yields

$$\frac{\partial\mathcal{E}_{RR}(\boldsymbol{w})}{\partial\boldsymbol{w}} = -2X\boldsymbol{y}^T + 2XX^T\boldsymbol{w} + 2\lambda\boldsymbol{w}$$

Setting the gradient to 0 and rearranging terms the optimal $\boldsymbol{w}$ satisfies

$$\boldsymbol{w} = X\underbrace{\frac{1}{\lambda}(\boldsymbol{y}^T - X^T\boldsymbol{w})}_{:=\boldsymbol{\alpha}\in\mathbb{R}^{n\times 1}}$$

From linear to kernel ridge regression

$$\mathcal{E}_{RR}(\boldsymbol{w}) = \boldsymbol{y}\boldsymbol{y}^\top - 2\boldsymbol{w}^\top X \boldsymbol{y}^\top + \boldsymbol{w}^\top X X^\top \boldsymbol{w} + \lambda \boldsymbol{w}^\top \boldsymbol{w}$$

Computing the derivative with respect to $\boldsymbol{w}$ yields

$$\frac{\partial \mathcal{E}_{RR}(\boldsymbol{w})}{\partial \boldsymbol{w}} = -2X\boldsymbol{y}^T + 2XX^T\boldsymbol{w} + 2\lambda\boldsymbol{w}$$

Setting the gradient to 0 and rearranging terms the optimal $\boldsymbol{w}$ satisfies

$$\boldsymbol{w} = X \underbrace{\frac{1}{\lambda}(\boldsymbol{y}^T - X^T\boldsymbol{w})}_{:=\boldsymbol{\alpha}\in\mathbb{R}^{n\times 1}}$$

From linear to kernel ridge regression

$$\mathcal{E}_{RR}(\boldsymbol{w}) = \boldsymbol{y}\boldsymbol{y}^\top - 2\boldsymbol{w}^\top X\boldsymbol{y}^\top + \boldsymbol{w}^\top XX^\top\boldsymbol{w} + \lambda\boldsymbol{w}^\top\boldsymbol{w}$$

Computing the derivative with respect to $\boldsymbol{w}$ yields

$$\frac{\partial\mathcal{E}_{RR}(\boldsymbol{w})}{\partial\boldsymbol{w}} = -2X\boldsymbol{y}^T + 2XX^T\boldsymbol{w} + 2\lambda\boldsymbol{w}$$

Setting the gradient to 0 and rearranging terms the optimal $\boldsymbol{w}$ satisfies

$$\boldsymbol{w} = X\underbrace{\frac{1}{\lambda}(\boldsymbol{y}^T - X^T\boldsymbol{w})}_{:=\boldsymbol{\alpha}\in\mathbb{R}^{n\times 1}} = \sum_i^n \alpha_i\boldsymbol{x}_i$$

We showed that the Representer Theorem is valid for ridge regression!

From linear to kernel ridge regression

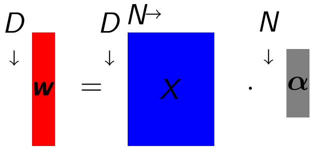
$$\mathcal{E}_{RR}(\boldsymbol{w}) = \boldsymbol{y}\boldsymbol{y}^\top - 2\boldsymbol{w}^\top X \boldsymbol{y}^\top + \boldsymbol{w}^\top X X^\top \boldsymbol{w} + \lambda \boldsymbol{w}^\top \boldsymbol{w}$$

From linear to kernel ridge regression

$$\mathcal{E}_{RR}(\boldsymbol{w}) = \boldsymbol{y}\boldsymbol{y}^\top - 2\boldsymbol{w}^\top X\boldsymbol{y}^\top + \boldsymbol{w}^\top XX^\top\boldsymbol{w} + \lambda\boldsymbol{w}^\top\boldsymbol{w}$$

$$\boldsymbol{w} = X\boldsymbol{\alpha}$$

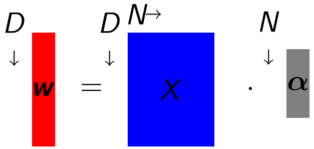
Next, plug $\boldsymbol{w} = X\boldsymbol{\alpha}$ into the error function $\mathcal{E}_{RR}$:



From linear to kernel ridge regression

$$\mathcal{E}_{RR}(\boldsymbol{w}) = \boldsymbol{y}\boldsymbol{y}^\top - 2\boldsymbol{w}^\top X \boldsymbol{y}^\top + \boldsymbol{w}^\top X X^\top \boldsymbol{w} + \lambda \boldsymbol{w}^\top \boldsymbol{w}$$

$$\boldsymbol{w} = X\boldsymbol{\alpha}$$



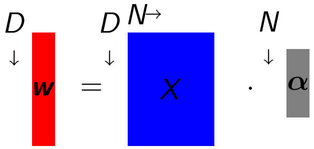
Next, plug $\boldsymbol{w} = X\boldsymbol{\alpha}$ into the error function $\mathcal{E}_{RR}$:

$$\mathcal{E}_{RR}(\boldsymbol{\alpha}) = \boldsymbol{y}\boldsymbol{y}^\top - 2\boldsymbol{\alpha}^T \underbrace{X^T X}_{K} \boldsymbol{y}^\top + \boldsymbol{\alpha}^T \underbrace{X^T X}_{K} \underbrace{X^\top X}_{K} \boldsymbol{\alpha} + \lambda \boldsymbol{\alpha}^T \underbrace{X^T X}_{K} \boldsymbol{\alpha}$$

From linear to kernel ridge regression

$$\mathcal{E}_{RR}(\boldsymbol{w}) = \boldsymbol{y}\boldsymbol{y}^\top - 2\boldsymbol{w}^\top X\boldsymbol{y}^\top + \boldsymbol{w}^\top XX^\top \boldsymbol{w} + \lambda \boldsymbol{w}^\top \boldsymbol{w}$$

$$\boldsymbol{w} = X\boldsymbol{\alpha}$$



Next, plug $\boldsymbol{w} = X\boldsymbol{\alpha}$ into the error function $\mathcal{E}_{RR}$:

$$\mathcal{E}_{RR}(\boldsymbol{\alpha}) = \boldsymbol{y}\boldsymbol{y}^\top - 2\boldsymbol{\alpha}^T \underbrace{X^T X}_{K} \boldsymbol{y}^\top + \boldsymbol{\alpha}^T \underbrace{X^T X}_{K}\underbrace{X^\top X}_{K}\boldsymbol{\alpha} + \lambda \boldsymbol{\alpha}^T \underbrace{X^T X}_{K}\boldsymbol{\alpha}$$

- We call this form **dual representation**
- Only scalar products appear, thus we can put in kernels:

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

- We write $k(X, X)$ as K

Kernel ridge regression (KRR)

For KRR we write $\varphi(\boldsymbol{x}) \in \mathbb{R}^{\tilde{D}}$ instead of $\boldsymbol{x}$

We defined $\alpha_i = \frac{1}{\lambda}(y_i - \varphi(\boldsymbol{x}_i)^T \boldsymbol{w})$

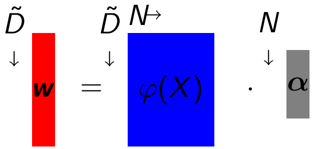
Optimal $\boldsymbol{w} = \varphi(X_{train})\boldsymbol{\alpha}$

Kernel ridge regression (KRR)

For KRR we write $\varphi(\boldsymbol{x}) \in \mathbb{R}^{\tilde{D}}$ instead of $\boldsymbol{x}$

We defined $\alpha_i = \frac{1}{\lambda}(y_i - \varphi(\boldsymbol{x}_i)^T \boldsymbol{w})$

Optimal $\boldsymbol{w} = \varphi(X_{train})\boldsymbol{\alpha}$



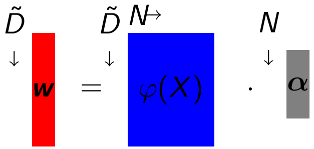
Kernel ridge regression (KRR)

$$\boldsymbol{\alpha} = \frac{1}{\lambda}(\boldsymbol{y}^\top - \varphi(X_{train})^\top \boldsymbol{w})$$

For KRR we write $\varphi(\boldsymbol{x}) \in \mathbb{R}^{\tilde{D}}$ instead of $\boldsymbol{x}$

We defined $\alpha_i = \frac{1}{\lambda}(y_i - \varphi(\boldsymbol{x}_i)^T \boldsymbol{w})$

Optimal $\boldsymbol{w} = \varphi(X_{train})\boldsymbol{\alpha}$



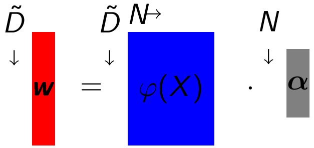
Kernel ridge regression (KRR)

$$\boldsymbol{\alpha} = \frac{1}{\lambda}(\boldsymbol{y}^\top - \varphi(X_{train})^\top \boldsymbol{w})$$
$$\lambda\boldsymbol{\alpha} = \boldsymbol{y}^\top - \varphi(X_{train})^\top \varphi(X_{train})\boldsymbol{\alpha}$$

For KRR we write $\varphi(\boldsymbol{x}) \in \mathbb{R}^{\tilde{D}}$ instead of $\boldsymbol{x}$

We defined $\alpha_i = \frac{1}{\lambda}(y_i - \varphi(\boldsymbol{x}_i)^T \boldsymbol{w})$

Optimal $\boldsymbol{w} = \varphi(X_{train})\boldsymbol{\alpha}$



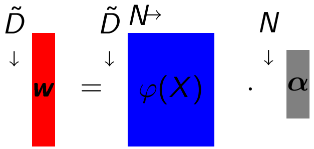
Kernel ridge regression (KRR)

$$
\begin{aligned}
\boldsymbol{\alpha} &= \frac{1}{\lambda}(\boldsymbol{y}^\top - \varphi(X_{train})^\top \boldsymbol{w}) \\
\lambda\boldsymbol{\alpha} &= \boldsymbol{y}^\top - \varphi(X_{train})^\top \varphi(X_{train})\boldsymbol{\alpha} \\
\boldsymbol{y}^\top &= (\varphi(X_{train})^\top \varphi(X_{train}) + \lambda I)\boldsymbol{\alpha}
\end{aligned}
$$

For KRR we write $\varphi(\boldsymbol{x}) \in \mathbb{R}^{\tilde{D}}$ instead of $\boldsymbol{x}$

We defined $\alpha_i = \frac{1}{\lambda}(y_i - \varphi(\boldsymbol{x}_i)^T \boldsymbol{w})$

Optimal $\boldsymbol{w} = \varphi(X_{train})\boldsymbol{\alpha}$



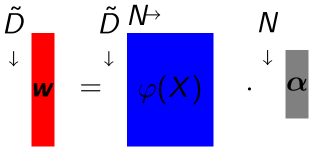
Kernel ridge regression (KRR)

$$\begin{aligned}
\boldsymbol{\alpha} &= \frac{1}{\lambda}(\boldsymbol{y}^\top - \varphi(X_{train})^\top \boldsymbol{w}) \\
\lambda\boldsymbol{\alpha} &= \boldsymbol{y}^\top - \varphi(X_{train})^\top \varphi(X_{train})\boldsymbol{\alpha} \\
\boldsymbol{y}^\top &= (\varphi(X_{train})^\top \varphi(X_{train}) + \lambda I)\boldsymbol{\alpha} \\
\boldsymbol{\alpha} &= (\varphi(X_{train})^\top \varphi(X_{train}) + \lambda I)^{-1}\boldsymbol{y}^\top
\end{aligned}$$

For KRR we write $\varphi(\boldsymbol{x}) \in \mathbb{R}^{\tilde{D}}$ instead of $\boldsymbol{x}$

We defined $\alpha_i = \frac{1}{\lambda}(y_i - \varphi(\boldsymbol{x}_i)^T \boldsymbol{w})$

Optimal $\boldsymbol{w} = \varphi(X_{train})\boldsymbol{\alpha}$



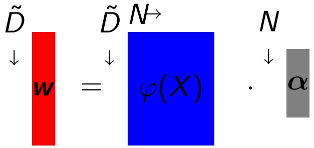
Kernel ridge regression (KRR)

$$\begin{aligned}
\boldsymbol{\alpha} =& \frac{1}{\lambda}(\boldsymbol{y}^\top - \varphi(X_{train})^\top \boldsymbol{w}) \\
\lambda\boldsymbol{\alpha} =& \boldsymbol{y}^\top - \varphi(X_{train})^\top \varphi(X_{train})\boldsymbol{\alpha} \\
\boldsymbol{y}^\top =& (\varphi(X_{train})^\top \varphi(X_{train}) + \lambda I)\boldsymbol{\alpha} \\
\boldsymbol{\alpha} =& (\varphi(X_{train})^\top \varphi(X_{train}) + \lambda I)^{-1}\boldsymbol{y}^\top \\
\boldsymbol{\alpha} =& (K + \lambda I)^{-1}\boldsymbol{y}^\top
\end{aligned}$$

For KRR we write $\varphi(\boldsymbol{x}) \in \mathbb{R}^{\tilde{D}}$ instead of $\boldsymbol{x}$

We defined $\alpha_i = \frac{1}{\lambda}(y_i - \varphi(\boldsymbol{x}_i)^T \boldsymbol{w})$

Optimal $\boldsymbol{w} = \varphi(X_{train})\boldsymbol{\alpha}$



Kernel ridge regression (KRR)

$$\begin{aligned}
\boldsymbol{\alpha} =& \frac{1}{\lambda}(\boldsymbol{y}^\top - \varphi(X_{train})^\top \boldsymbol{w}) \\
\lambda\boldsymbol{\alpha} =& \boldsymbol{y}^\top - \varphi(X_{train})^\top \varphi(X_{train})\boldsymbol{\alpha} \\
\boldsymbol{y}^\top =& (\varphi(X_{train})^\top \varphi(X_{train}) + \lambda I)\boldsymbol{\alpha} \\
\boldsymbol{\alpha} =& (\varphi(X_{train})^\top \varphi(X_{train}) + \lambda I)^{-1}\boldsymbol{y}^\top \\
\boldsymbol{\alpha} =& (K + \lambda I)^{-1}\boldsymbol{y}^\top
\end{aligned}$$

This $\boldsymbol{\alpha}$ minimizes $\mathcal{E}_{RR}(\boldsymbol{\alpha})$.

For KRR we write $\varphi(\boldsymbol{x}) \in \mathbb{R}^{\tilde{D}}$ instead of $\boldsymbol{x}$

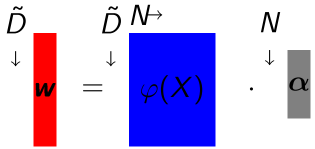
We defined $\alpha_i = \frac{1}{\lambda}(y_i - \varphi(\boldsymbol{x}_i)^T \boldsymbol{w})$

Optimal $\boldsymbol{w} = \varphi(X_{train})\boldsymbol{\alpha}$

Ridge regression (RR)

Train $\boldsymbol{w}$ which minimizes $\mathcal{E}_{RR}(\boldsymbol{w})$:

$$\boldsymbol{w} = (\varphi(X)\varphi(X)^T + \lambda I)^{-1}\varphi(X)y^T$$



Kernel ridge regression (KRR)

$$\begin{aligned}
\boldsymbol{\alpha} =& \frac{1}{\lambda}(\boldsymbol{y}^\top - \varphi(X_{train})^\top \boldsymbol{w}) \\
\lambda\boldsymbol{\alpha} =& \boldsymbol{y}^\top - \varphi(X_{train})^\top \varphi(X_{train})\boldsymbol{\alpha} \\
\boldsymbol{y}^\top =& (\varphi(X_{train})^\top \varphi(X_{train}) + \lambda I)\boldsymbol{\alpha} \\
\boldsymbol{\alpha} =& (\varphi(X_{train})^\top \varphi(X_{train}) + \lambda I)^{-1}\boldsymbol{y}^\top \\
\boldsymbol{\alpha} =& (K + \lambda I)^{-1}\boldsymbol{y}^\top
\end{aligned}$$

This $\boldsymbol{\alpha}$ minimizes $\mathcal{E}_{RR}(\boldsymbol{\alpha})$.

Ridge regression (RR)

Train $\boldsymbol{w}$ which minimizes $\mathcal{E}_{RR}(\boldsymbol{w})$:

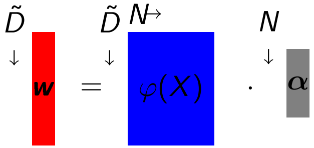
$$\boldsymbol{w} = (\varphi(X)\varphi(X)^T + \lambda I)^{-1}\varphi(X)y^T$$

For KRR we write $\varphi(\boldsymbol{x}) \in \mathbb{R}^{\tilde{D}}$ instead of $\boldsymbol{x}$

We defined $\alpha_i = \frac{1}{\lambda}(y_i - \varphi(\boldsymbol{x}_i)^T \boldsymbol{w})$

Optimal $\boldsymbol{w} = \varphi(X_{train})\boldsymbol{\alpha}$

KRR trains $\boldsymbol{\alpha} \in \mathbb{R}^n$, RR trains $\boldsymbol{w} \in \mathbb{R}^{\tilde{D}}$



Kernel ridge regression (KRR)

$$
\begin{aligned}
\boldsymbol{\alpha} =& \frac{1}{\lambda}(\boldsymbol{y}^\top - \varphi(X_{train})^\top \boldsymbol{w})\\
\lambda\boldsymbol{\alpha} =& \boldsymbol{y}^\top - \varphi(X_{train})^\top \varphi(X_{train})\boldsymbol{\alpha}\\
\boldsymbol{y}^\top =& (\varphi(X_{train})^\top \varphi(X_{train}) + \lambda I)\boldsymbol{\alpha}\\
\boldsymbol{\alpha} =& (\varphi(X_{train})^\top \varphi(X_{train}) + \lambda I)^{-1}\boldsymbol{y}^\top\\
\boldsymbol{\alpha} =& (K + \lambda I)^{-1}\boldsymbol{y}^\top
\end{aligned}
$$

This $\boldsymbol{\alpha}$ minimizes $\mathcal{E}_{RR}(\boldsymbol{\alpha})$.

Ridge regression (RR)

Train $\boldsymbol{w}$ which minimizes $\mathcal{E}_{RR}(\boldsymbol{w})$:

$$\boldsymbol{w} = (\varphi(X)\varphi(X)^T + \lambda I)^{-1}\varphi(X)y^T$$

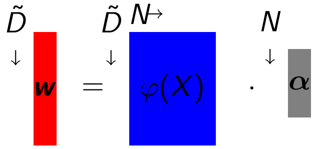
For KRR we write $\varphi(\boldsymbol{x}) \in \mathbb{R}^{\tilde{D}}$ instead of $\boldsymbol{x}$

We defined $\alpha_i = \frac{1}{\lambda}(y_i - \varphi(\boldsymbol{x}_i)^T \boldsymbol{w})$

Optimal $\boldsymbol{w} = \varphi(X_{train})\boldsymbol{\alpha}$

KRR trains $\boldsymbol{\alpha} \in \mathbb{R}^n$, RR trains $\boldsymbol{w} \in \mathbb{R}^{\tilde{D}}$

For $\varphi(x)^T\varphi(x') = k(x, x')$ the two models are equivalent (but not the runtime complexity)



Predictions for new data $\boldsymbol{x}_{new}$

$$y_{new} = \boldsymbol{w}^T \varphi(\boldsymbol{x}_{new})$$

Optimal $\boldsymbol{\alpha} = (K + \lambda I)^{-1} \boldsymbol{y}^T$

$\boldsymbol{w} = \varphi(X_{train})\boldsymbol{\alpha}$

$K(a, b) = K(b, a)^T$

We call $K = k(X_{train}, X_{train})$ the *Gram matrix*

Predictions for new data $\boldsymbol{x}_{new}$

$$\begin{aligned} y_{new} &= \boldsymbol{w}^T \varphi(\boldsymbol{x}_{new}) \\ &= (\varphi(X_{train})\boldsymbol{\alpha})^T \varphi(\boldsymbol{x}_{new}) \end{aligned}$$

Optimal $\boldsymbol{\alpha} = (K + \lambda I)^{-1} \boldsymbol{y}^T$

$\boldsymbol{w} = \varphi(X_{train})\boldsymbol{\alpha}$

$K(a, b) = K(b, a)^T$

We call $K = k(X_{train}, X_{train})$ the *Gram matrix*

Predictions for new data $\boldsymbol{x}_{new}$

$$\begin{aligned} y_{new} &= \boldsymbol{w}^T \varphi(\boldsymbol{x}_{new}) \\ &= (\varphi(X_{train})\boldsymbol{\alpha})^T \varphi(\boldsymbol{x}_{new}) \\ &= \boldsymbol{\alpha}^T \varphi(X_{train})^T \varphi(\boldsymbol{x}_{new}) \end{aligned}$$

Optimal $\boldsymbol{\alpha} = (K + \lambda I)^{-1} \boldsymbol{y}^T$

$\boldsymbol{w} = \varphi(X_{train})\boldsymbol{\alpha}$

$K(a, b) = K(b, a)^T$

We call $K = k(X_{train}, X_{train})$ the *Gram matrix*

Predictions for new data $\boldsymbol{x}_{new}$

$$\begin{aligned} y_{new} &= \boldsymbol{w}^T \varphi(\boldsymbol{x}_{new}) \\ &= (\varphi(X_{train})\boldsymbol{\alpha})^T \varphi(\boldsymbol{x}_{new}) \\ &= \boldsymbol{\alpha}^T \varphi(X_{train})^T \varphi(\boldsymbol{x}_{new}) \\ &= \boldsymbol{\alpha}^T k(X_{train}, \boldsymbol{x}_{new}) \end{aligned}$$

Optimal $\boldsymbol{\alpha} = (K + \lambda I)^{-1} \boldsymbol{y}^T$

$\boldsymbol{w} = \varphi(X_{train})\boldsymbol{\alpha}$

$K(a, b) = K(b, a)^T$

We call $K = k(X_{train}, X_{train})$ the *Gram matrix*

Predictions for new data $\boldsymbol{x}_{new}$

$$\begin{aligned} y_{new} &= \boldsymbol{w}^T \varphi(\boldsymbol{x}_{new}) \\ &= (\varphi(X_{train})\boldsymbol{\alpha})^T \varphi(\boldsymbol{x}_{new}) \\ &= \boldsymbol{\alpha}^T \varphi(X_{train})^T \varphi(\boldsymbol{x}_{new}) \\ &= \boldsymbol{\alpha}^T k(X_{train}, \boldsymbol{x}_{new}) \\ &= \boldsymbol{y}_{train}(K + \lambda I)^{-1} k(X_{train}, \boldsymbol{x}_{new}) \end{aligned}$$

Optimal $\boldsymbol{\alpha} = (K + \lambda I)^{-1} \boldsymbol{y}^T$

$\boldsymbol{w} = \varphi(X_{train})\boldsymbol{\alpha}$

$K(a, b) = K(b, a)^T$

We call $K = k(X_{train}, X_{train})$ the *Gram matrix*

Summary kernel ridge regression

- Input: kernel function $k(\boldsymbol{x}, \boldsymbol{x}') = \varphi(\boldsymbol{x})^T \varphi(\boldsymbol{x}')$, regularization hyperparameter λ, training dataset $(X_{train}, \boldsymbol{y}_{train})$ and test datapoints X_{test}
- Training [3]:

$$\boldsymbol{\alpha} = (k(X_{train}, X_{train}) + \lambda I)^{-1} \boldsymbol{y}_{train}^T$$

- Predicting:

$$\hat{\boldsymbol{y}}_{test} = \boldsymbol{\alpha}^T k(X_{train}, X_{test})$$

[3]Since we need X_{train} for predictions, we cannot forget the data (this is different in RR!)

Kernels as similarity measures

Prediction step:

$$f^*(\boldsymbol{x}_{\text{new}}) = \sum_{i=1}^{N} \alpha_i k(\boldsymbol{x}_{\text{new}}, \boldsymbol{x}_i)$$

Kernel methods are *memory-based* methods:

- Store the entire training set
- Define similarity of data points by kernel function

$$k(\cdot,\cdot) : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$$

- New predictions require comparison with previously learned examples

Universal law of generalization

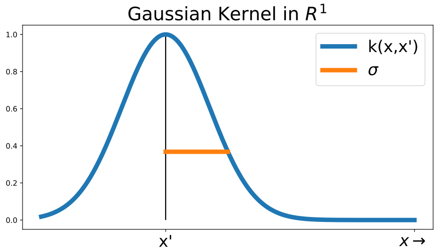
- Roger N. Shepard suggested that the **perceptual similarity** of new data $\boldsymbol{x}$ decays exponentially with distance from a prototype $\boldsymbol{x}'$ [Shepard, 1987].

Universal law of generalization

- Roger N. Shepard suggested that the **perceptual similarity** of new data $\boldsymbol{x}$ decays exponentially with distance from a prototype $\boldsymbol{x}'$ [Shepard, 1987].
    - Motivation for using the Gaussian kernel (also called *radial basis function (RBF)*)

Universal law of generalization

- Roger N. Shepard suggested that the **perceptual similarity** of new data $\boldsymbol{x}$ decays exponentially with distance from a prototype $\boldsymbol{x}'$ [Shepard, 1987].
  - Motivation for using the Gaussian kernel (also called *radial basis function (RBF)*)

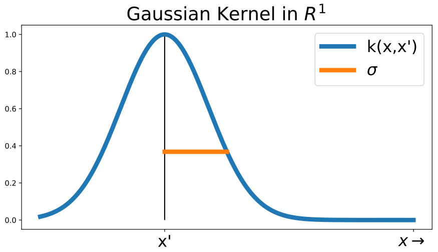


Gaussian kernel (RBF kernel)

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

Universal law of generalization

- Roger N. Shepard suggested that the **perceptual similarity** of new data $\boldsymbol{x}$ decays exponentially with distance from a prototype $\boldsymbol{x}'$ [Shepard, 1987].
    - Motivation for using the Gaussian kernel (also called *radial basis function (RBF)*)



Gaussian kernel (RBF kernel)

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

Because $\exp(x) = \sum_{k=0}^{\infty} \frac{x^k}{k!}$, the basis function $\varphi(\boldsymbol{x})$ is infinite dimensional

Let’s see KRR with the Gaussian kernel in action

Kernel ridge regression example

KRR with Gaussian kernels

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

Predictions:

$$y_{new} = y_{train}(K+\lambda I)^{-1} k(X_{train}, \boldsymbol{x}_{new})$$

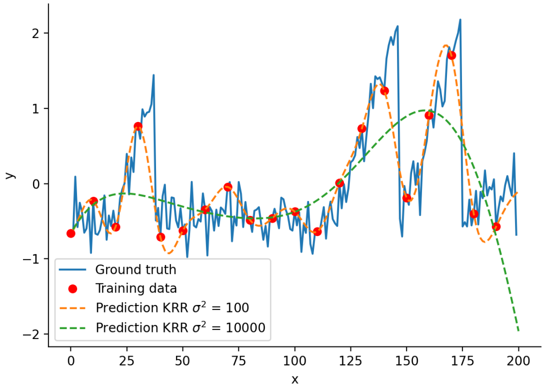
Kernel ridge regression example

KRR with Gaussian kernels

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

Predictions:

$$y_{new} = y_{train}(K + \lambda I)^{-1} k(X_{train}, \boldsymbol{x}_{new})$$



Kernel methods — Pros & Cons

+ Powerful modeling tool
  (non-linear problems become linear in kernel space)
+ Omni-purpose Kernels
  (Gaussian works well in many cases)
+ Kernel methods can handle symbolic objects
+ When you have less data points than your data has dimensions, kernel methods can offer a dramatic speedup
+ Existing methods can be kernelized

– Difficult to understand what's happening in kernel space
– Model complexity increases with number of data points

→ If you have too much data, kernel methods can be slow

Generalization and model selection

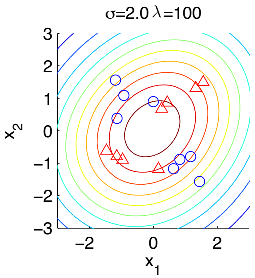
The best model is the model that *generalizes best*

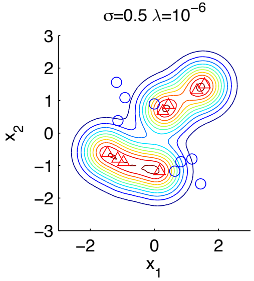
Generalization and model selection

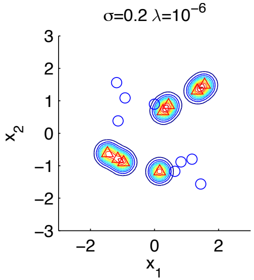
The best model is the model that *generalizes best*
Which is best in this context?

Generalization and model selection

The best model is the model that *generalizes best*
Which is best in this context?



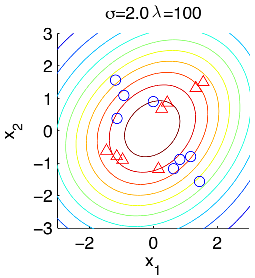




Generalization and model selection

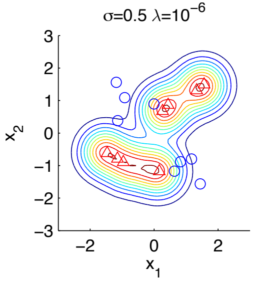
The best model is the model that *generalizes best*
Which is best in this context?

Underfitting



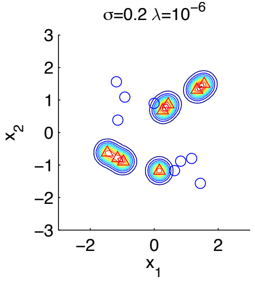
Model is too simple
$\rightarrow$ *Bad generalization*

Better fit



Appropriate complexity
$\rightarrow$ Good generalization

Overfitting



Model is too complex

Recap: cross-validation

Split data set in F different **training** and **test** data

fold 1 $[\ \underbrace{x_1,\ x_2,\ x_3,\ x_4,}_{\mathcal{F}_1^{\text{train}}}\ \underbrace{x_5,\ x_6}_{\mathcal{F}_1^{\text{test}}}\ ]$

fold 2 $[\ \underbrace{x_1,\ x_2,}_{\mathcal{F}_1^{\text{test}}}\ \underbrace{x_3,\ x_4,\ x_5,\ x_6}_{\mathcal{F}_1^{\text{train}}}\ ]$

fold 3 . . .

For each fold:

Train your model on the training data

Test your model on the test data

How to achieve good generalization?

When using powerful algorithms (MLPs, KRR, . . .)
every data set can be modeled perfectly! (overfitting)

But we want to model new data well (generalization)

Cross-validation can be used for **either**:

Model selection
Optimize hyper-parameters of a model for generalization performance
Model evaluation
Test how good an algorithm with fixed parameters actually is

Cross-validation: model selection or model evaluation

Model evaluation
Report **mean evaluation score** – e.g. accuracy – across folds

Model selection
Take hyper-parameter with the highest mean score across folds

[4] e.g. see this sklearn example

Cross-validation: model selection or model evaluation

Model evaluation

Report **mean evaluation score** – e.g. accuracy – across folds

Model selection

Take hyper-parameter with the highest mean score across folds

We want to estimate the performance of a model which we optimize on unseen data:

[4] e.g. see this sklearn example

Cross-validation: model selection or model evaluation

Model evaluation
Report **mean evaluation score** – e.g. accuracy – across folds

Model selection
Take hyper-parameter with the highest mean score across folds

We want to estimate the performance of a model which we optimize on unseen data:

- If we did selection and evaluation on the same test fold:

[4] e.g. see this sklearn example

Cross-validation: model selection or model evaluation

Model evaluation
Report **mean evaluation score** – e.g. accuracy – across folds

Model selection
Take hyper-parameter with the highest mean score across folds

We want to estimate the performance of a model which we optimize on unseen data:

- If we did selection and evaluation on the same test fold:
    - We would be too optimistic

[4] e.g. see this sklearn example

Cross-validation: model selection or model evaluation

Model evaluation
Report **mean evaluation score** – e.g. accuracy – across folds

Model selection
Take hyper-parameter with the highest mean score across folds

We want to estimate the performance of a model which we optimize on unseen data:

- If we did selection and evaluation on the same test fold:
  - We would be too optimistic
  - → because we use same set for optimizing and evaluating [4]

[4] e.g. see this sklearn example

Cross-validation: model selection or model evaluation

Model evaluation
Report **mean evaluation score** – e.g. accuracy – across folds

Model selection
Take hyper-parameter with the highest mean score across folds

We want to estimate the performance of a model which we optimize on unseen data:

- If we did selection and evaluation on the same test fold:
  - We would be too optimistic

  $\rightarrow$ because we use same set for optimizing and evaluating [4]

  $\rightarrow$ would be similar to reporting train error

[4] e.g. see this sklearn example

Cross-validation: model selection or model evaluation

Model evaluation
Report **mean evaluation score** – e.g. accuracy – across folds

Model selection
Take hyper-parameter with the highest mean score across folds

We want to estimate the performance of a model which we optimize on unseen data:

- If we did selection and evaluation on the same test fold:
  - We would be too optimistic
  - → because we use same set for optimizing and evaluating [4]
  - → would be similar to reporting train error

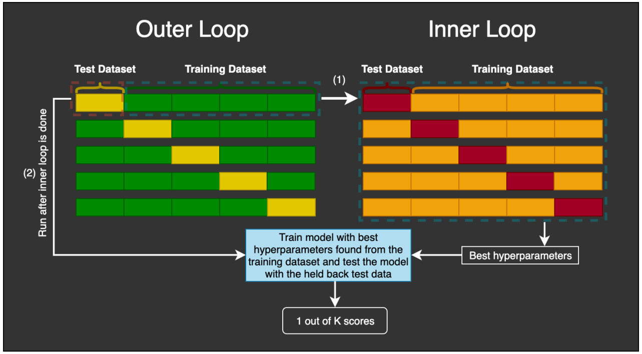
→ Solution: **Nested cross-validation**

[4] e.g. see this sklearn example

Nested cross-validation

It is simply
CV for model selection
nested inside
CV for model evaluation

- Two loops, two hold out folds
- Only gives you an estimator for performance
- **Does not replace parameter tuning**


Outer Loop
Inner Loop
Test Dataset
Training Dataset
Test Dataset
Training Dataset
(1)
(2)
Run after inner loop is done
Train model with best hyperparameters found from the training dataset and test the model with the held back test data
Best hyperparameters
1 out of K scores

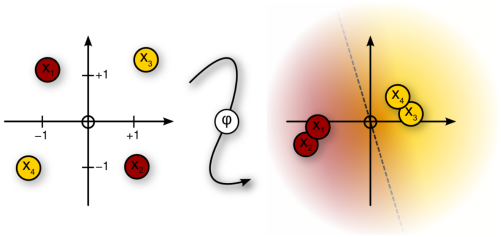
Source: mlfromscratch.com/nested-cross-validation-python-code

Nested cross-validation

Algorithm 1: Cross-Validation for Model Selection and Evaluation

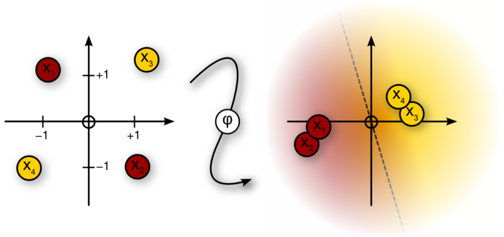
```
Require: Data $(x_1, y_1) \dots, (x_N, y_N)$, parameters $\sigma_1, \dots, \sigma_S$, Number of CV folds $F$
 1: Split data in $F$ disjunct folds
 2: for Outer folds $f_{\text{outer}} = 1, \dots, F$ do
 3:     Pick folds $\{1, \dots, F\} \setminus f_{\text{outer}}$ for Model Selection
 4:     Model Selection
 5:     for Fold $f_{\text{inner}} = 1, \dots, F-1$ do
 6:         for Parameter $s = 1, \dots, S$ do
 7:             Train model on folds $\{1, \dots, F\} \setminus \{f_{\text{outer}}, f_{\text{inner}}\}$ with parameter $\sigma_s$
 8:             Compute prediction on fold $f_{\text{inner}}$
 9:         end for
10:     end for
11:     Pick best parameter $\sigma_s$ for all $f_{\text{inner}}$
12:     Model Evaluation
13:     Train model on folds $\{1, \dots, F\} \setminus f_{\text{outer}}$ with parameter $\sigma_s$
14:     $\text{Performance}_{\text{outer}} \leftarrow$ Test model on fold $f_{\text{outer}}$
15: end for
16: return Average of $\text{Performance}_{\text{outer}}$
```

Kernelizing linear methods



[Jäkel et al., 2009]

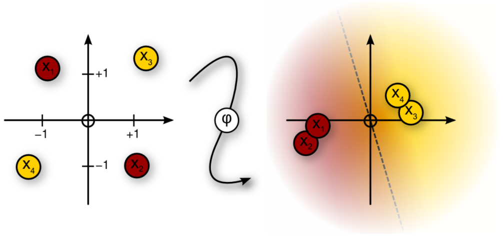
Kernelizing linear methods



[Jäkel et al., 2009]

1. Map the data into a (high dimensional) feature space, $\boldsymbol{x} \mapsto \varphi(\boldsymbol{x})$

Kernelizing linear methods



[Jäkel et al., 2009]

1. Map the data into a (high dimensional) feature space, $\boldsymbol{x} \mapsto \varphi(\boldsymbol{x})$
2. Look for linear relations in the feature space
   - Work in that space by considering scalar product of data points, $k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \langle \varphi(\boldsymbol{x}_i), \varphi(\boldsymbol{x}_j) \rangle$
   - Many linear models have a *dual representation* that only uses scalars products between the data points
   - k is called the *kernel function*

Summary

Kernel ridge regression

- Non-linear regression
- Predictions involve comparison of new and old data
- Predictions based on linear combination of (non-linear) similarity measures
- Optimization requires inversion of kernel matrix
  ($N \times N, \rightarrow \mathcal{O}(N^3)$)
  (difficult for very large data sets)

Summary

Kernel ridge regression

- Non-linear regression
- Predictions involve comparison of new and old data
- Predictions based on linear combination of (non-linear) similarity measures
- Optimization requires inversion of kernel matrix
  $(N \times N, \rightarrow \mathcal{O}(N^3))$
  (difficult for very large data sets)

Generalization and model selection

- Good prediction on new data is called generalization
- Cross-validation is a simple and powerful framework for model selection
- Nested Cross-validation gives you a valid estimate for generalization of your model class
  (without giving you parameters or hyperparameters)

References

C. M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. 2007.

C. F. Gauß. Theoria motus corporum coelestium in sectionibus conicis solem ambientium. *Göttingen*, 1809.

A. E. Hoerl and R. W. Kennar. Ridge regression: Applications to nonorthogonal problems. *Technometrics*, 12(1):69–82, 1970.

F. Jäkel, B. Schölkopf, and F. A. Wichmann. Does cognitive science need kernels? *Trends Cogn Sci*, 13(9):381–388, 2009. doi: 016/j.tics.2009.06.002.

G. Kimeldorf and G. Wahba. Some results on tchebycheffian spline functions. *Journal of mathematical analysis and applications*, 33(1):82–95, 1971.

A.-M. Legendre. *Nouvelles méthodes pour la détermination des orbites des comètes*, chapter Sur la methode des moindres quarres. Firmin Didot, http://imgbase-scd-ulp.u-strasbg.fr/displayimage.php?pos=-141297, 1805.

J. Mercer. Xvi. functions of positive and negative type, and their connection the theory of integral equations. *Philosophical transactions of the royal society of London. Series A, containing papers of a mathematical or physical character*, 209(441-458):415–446, 1909.

J. Shawe-Taylor and N. Cristianini. *Kernel methods for pattern analysis*. Cambridge University Press, 2004.

R. N. Shepard. Toward a universal law of generalization for psychological science. *Science*, 237(4820):1317–23, 1987.

A. N. Tychonoff. On the stability of inverse problems. *Doklady Akademii Nauk SSSR*, 39(5):195–198, 1943.