Cognitive Algorithms
Lecture 4

Kernel Methods

Klaus-Robert Müller, **Ali Hashemi**, **Lorenz Vaitl**,
Augustin Krause, Joanina Obersdorff, Ken Schreiber

Berlin Institute of Technology
Dept. Machine Learning

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)

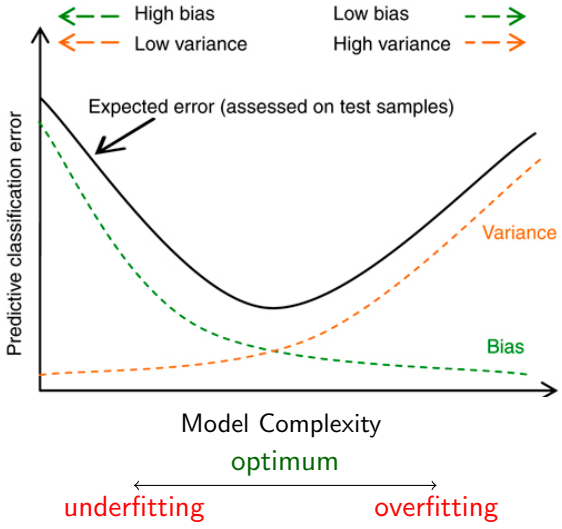
A.M. Legendre (1752-1833)

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), i.e. the minimum variance unbiased estimator that is linear in the y.

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

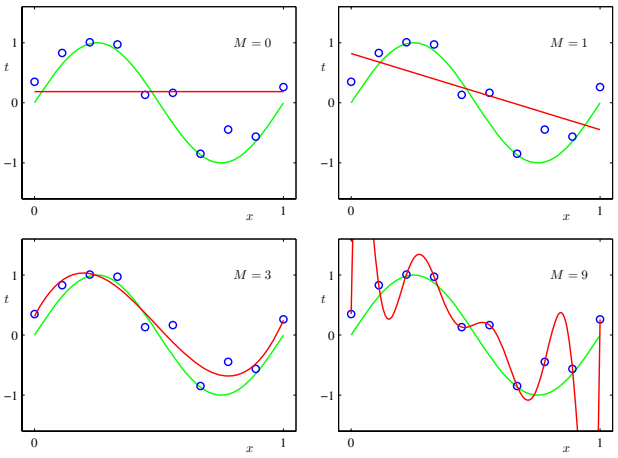
The Bias-Variance Tradeoff



Example: Linear regression for a polynomial function

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

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



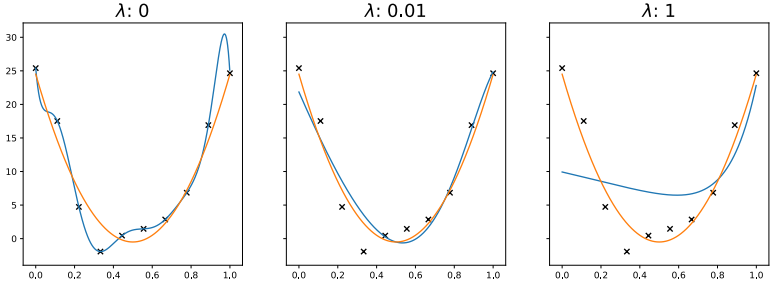
[Bishop, 2007]

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

Computing the derivative w.r.t. w yields

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

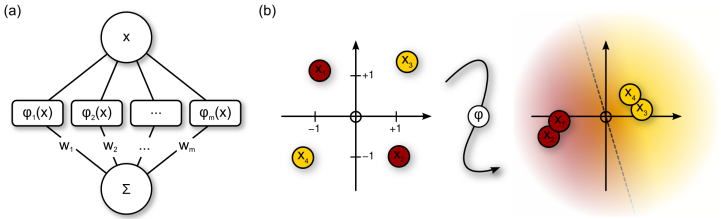
Setting the gradient to zero and rearranging terms the optimal $\boldsymbol{w}$ is

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

$\Rightarrow$ 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 in the feature space

Definition Kernel

Definition

We define a Kernel function k as a dot product in feature space $\mathcal{F}$ where

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

and so

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

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 $k(\boldsymbol{x}, \boldsymbol{x}') = \varphi(\boldsymbol{x})^T \cdot \varphi(\boldsymbol{x}')$.

Why do it?

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 $\tilde{d} \gg d$

$\Rightarrow$ we get 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 $\varphi(\boldsymbol{x})$

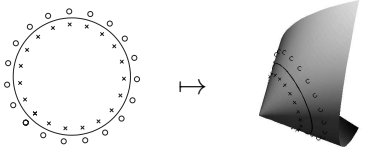
Example kernel

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

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 $\varphi(\boldsymbol{x})$



$\rightarrow$ 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 semidefinite. [1]

Symmetric Matrix:

$$A = A^\mathsf{T}$$

Positive semi-definite Matrix:

$$x^\mathsf{T} A x \geq 0 \quad \forall\, x$$

alternatively, all eigenvalues λ of A 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 formulation:

construct a feature space F with a scalar product

If kernel is positive semi-definite symmetric then one can find a map φ such that $k(x, x') = \varphi(x)^T \varphi(x')$ for almost all $x \in \mathcal{X}$

- → 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 exponentiating

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

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/ RBF (more on this later)

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

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

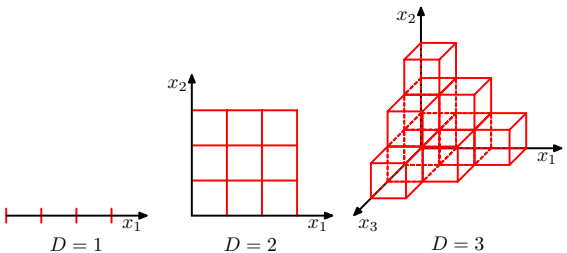
The curse of dimensionality

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

稀少的

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 $\tilde{d} >> n$

Representer Theorem [Kimeldorf and Wahba, 1971]

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

number of points

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

I.e. only scalar products between $\varphi(\boldsymbol{x})$ and the training data $\varphi(\boldsymbol{x}_i)$ are relevant.
For the model $f(x) = \boldsymbol{w}^\top \varphi(\boldsymbol{x})$ this means that $\boldsymbol{w}$ can be written as

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

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 how can we formulate an algorithm like this?

→ In the following we will kernelize Ridge Regression as an example on how to kernelize linear models

→ We'll recast the problem into an equivalent dual representation (can be done with many linear models)

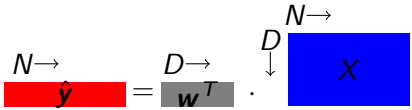
Recap: Linear Regression

The Linear Regression model in matrix notation then becomes

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

Linear Regression minimizes the least-squares loss function

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



Beware: $\boldsymbol{y}$ is a row vector, while $\boldsymbol{w}$ is a column vector.

Recap: Ridge Regression

Linear ridge regression looks for a linear combination of features $\boldsymbol{w}$ that minimizes the prediction error and has a small norm
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

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

Computing the derivative w.r.t. $\boldsymbol{w}$ 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

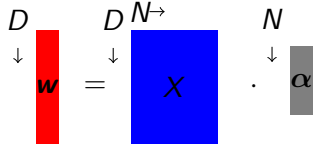
$$\boldsymbol{w} = X\underbrace{\frac{1}{\lambda}(\boldsymbol{y}^T - X^T\boldsymbol{w})}_{:=\boldsymbol{\alpha}(\in\mathbb{R}^{n\times 1})} = \sum_i^n \alpha_i \boldsymbol{x}_i$$

$\rightarrow$ we showed that the Representer Theorem is valid for RR!

From Linear to Kernel Ridge Regression

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

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



Let's look at the error function $\mathcal{E}_{RR}$ w.r.t. $\boldsymbol{\alpha}$

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

We call this form **dual representation**

Only scalar products $\rightarrow$ 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:

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

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

Ridge Regression

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

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

Training KRR

Elements for RR: $\varphi(X)\cdot\varphi(X)^T = \begin{pmatrix} \sum_i \varphi_1(x_i)\varphi_1(x_i) & \cdots \\ \sum_i \varphi_2(x_i)\varphi_1(x_i) & \cdots \\ \vdots & \end{pmatrix}$

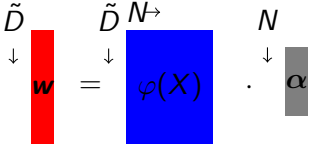
For KRR we write $\varphi(\boldsymbol{x}) \in \mathbb{R}^{\tilde{D}}$ instead of $\boldsymbol{x}$

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 $\boldsymbol{x}_{new}$

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

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 [2]:

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

- Predicting:

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

[2] Since we need X_{train} again for predictions, we cannot forget the data like for RR

Kernels as Similarity Measures

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

Kernel methods are *memory-based* methods:

- store the entire training set
- define similarity of data points by kernel function

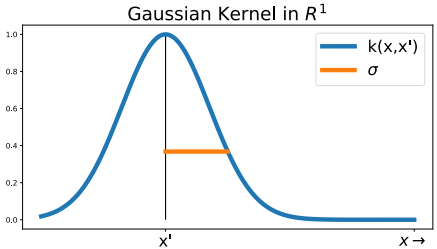
$$k(\cdot, \cdot) : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$$

- new predictions require comparision with previously learned examples

Law of Universal Generalization

- Roger Shepard argued that the **Perceptual Similarity** of new data $\boldsymbol{x}$ decays exponentially with distance from prototype $\boldsymbol{x}'$ [Shepard, 1987]

  $\rightarrow$ motivation for using the Gaussian Kernel



Gaussian Kernel (RBF Kernel)

$$k(\boldsymbol{x}', \boldsymbol{x}) = e^{\frac{-(\boldsymbol{x}' - \boldsymbol{x})^2}{2\sigma^2}}$$

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

Kernel Ridge Regression:

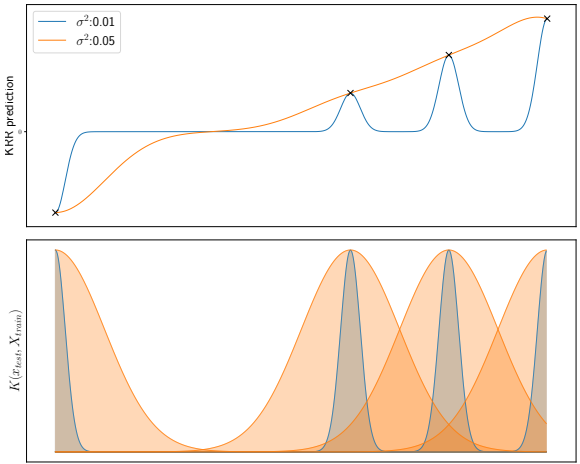
Example

KRR with Gaussian Kernels:

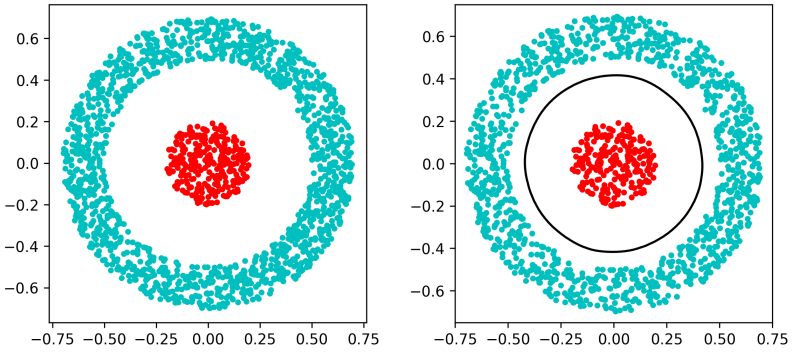
$$k(x, x') = \exp\left\{-\frac{||x - x'||^2}{2\sigma^2}\right\}$$

Predictions:

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



We can also kernelize LDA with e.g. a Gaussian Kernel ($\sigma = 0.1$)



$\rightarrow$ We are able to do non-linear classification!

Kernel Methods - Pros and Cons

+ Powerful Modeling Tool
  (non-linear problems become linear in kernel space)
+ Omnipurpose 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

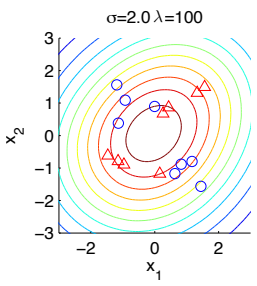
– Difficult to understand what's happening in kernel space
– Model complexity increases with number of data points
→ If you have too much data, kernel methods can be slow

+ 强大的建模工具
  (非线性问题在核空间中变为线性)
+ 通用内核
  (高斯在许多情况下效果很好)
+ 内核方法可以处理符号对象
+ 当您的数据点少于数据的维度时，内核方法可以提供显著的加速
– 难以理解内核空间中发生了什么
– 模型复杂性随着数据点数量的增加而增加
→ 如果数据过多，内核方法可能会很慢

Generalization and Model Selection

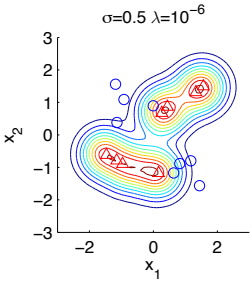
The best model is the model that *generalizes best*

Underfitting



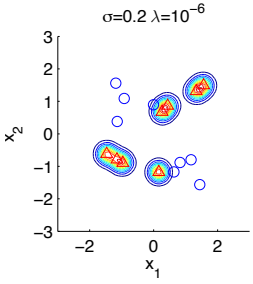
Model is too simple
→ *Bad generalization*

Better fit



Appropriate complexity
→ 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

Want to estimate the performance 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 [3]
  - $\rightarrow$ would be similar to reporting train error

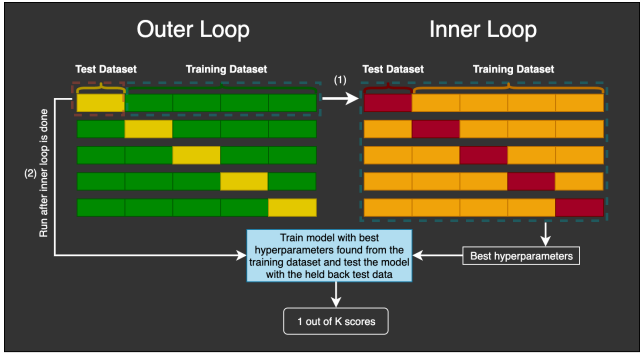
$\rightarrow$ Solution: **Nested Cross-Validation**

[3] 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**



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

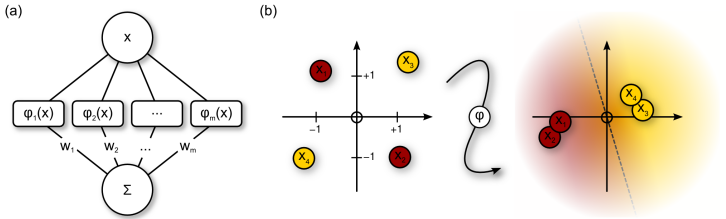
Nested Cross-Validation

Algorithm 1: Cross-Validation for Model Selection and Evaluation

Require: Data $(x_1, y_1) \ldots, (x_N, y_N)$, parameters $\sigma_1, \ldots, \sigma_S$, Number of CV folds F

Split data in F **disjunct** folds
for Outer folds $f_{\text{outer}} = 1, \ldots, F$ **do**
    Pick folds $\{1, \ldots, F\} \setminus f_{\text{outer}}$ for Model Selection
    Model Selection
    for Fold $f_{\text{inner}} = 1, \ldots, F - 1$ **do**
        for Parameter $s = 1, \ldots, S$ **do**
            Train model on folds $\{1, \ldots, F\} \setminus \{f_{\text{outer}}, f_{\text{inner}}\}$ with parameter σ_s
            Compute prediction on fold f_{inner}
        end for
    end for
    Pick best parameter σ_s for all f_{inner}
    Model Evaluation
    Train model on folds $\{1, \ldots, F\} \setminus f_{\text{outer}}$ with parameter σ_s
    $\text{Performance}_{\text{outer}} \leftarrow$ Test model on fold f_{outer}
end for
return Average of $\text{Performance}_{\text{outer}}$

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
($N \times N$, $\rightarrow \mathcal{O}(N^3)$)

$\rightarrow$ 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.

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.

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.

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.