ML Intro 1: shallow stuff

F. Noé[1]

September 4, 2019

Regression

- **Supervised learning** problem
- **Training set**: N points $(\mathbf{x}_i, y_i)$, $\mathbf{x}_i \in \mathbb{R}^n$, $\mathbf{y}_i \in \mathbb{R}$, $i = 1, \ldots N$.
- **Aim**: *approximate* the equation $f : \mathbb{R}^n \mapsto \mathbb{R}$ underlying the data with a **model** $\hat{y}(\mathbf{x}; \mathbf{w})$ using **parameters** $\mathbf{w} \in \mathbb{R}^n$:

$$\hat{y}_i = \hat{y}(\mathbf{x}_i; \mathbf{w}) \approx f(\mathbf{x}_i)$$

- **Learning problem**: Find $\mathbf{w}$ such that we minimize the residuals $\mathbf{\Delta}$:

$$\hat{\mathbf{w}} = \arg\min_{\mathbf{w}} \|\mathbf{\Delta}\|$$

defined by

$$\Delta_i = y_i - \hat{y}(\mathbf{x}_i; \mathbf{w}).$$

Regression

- **Supervised learning** problem
- **Training set**: N points $(\mathbf{x}_i, y_i)$, $\mathbf{x}_i \in \mathbb{R}^n$, $\mathbf{y}_i \in \mathbb{R}$, $i = 1, \ldots N$.
- **Aim**: *approximate* the equation $f : \mathbb{R}^n \mapsto \mathbb{R}$ underlying the data with a **model** $\hat{y}(\mathbf{x}; \mathbf{w})$ using **parameters** $\mathbf{w} \in \mathbb{R}^n$:

$$\hat{y}_i = \hat{y}(\mathbf{x}_i; \mathbf{w}) \approx f(\mathbf{x}_i)$$

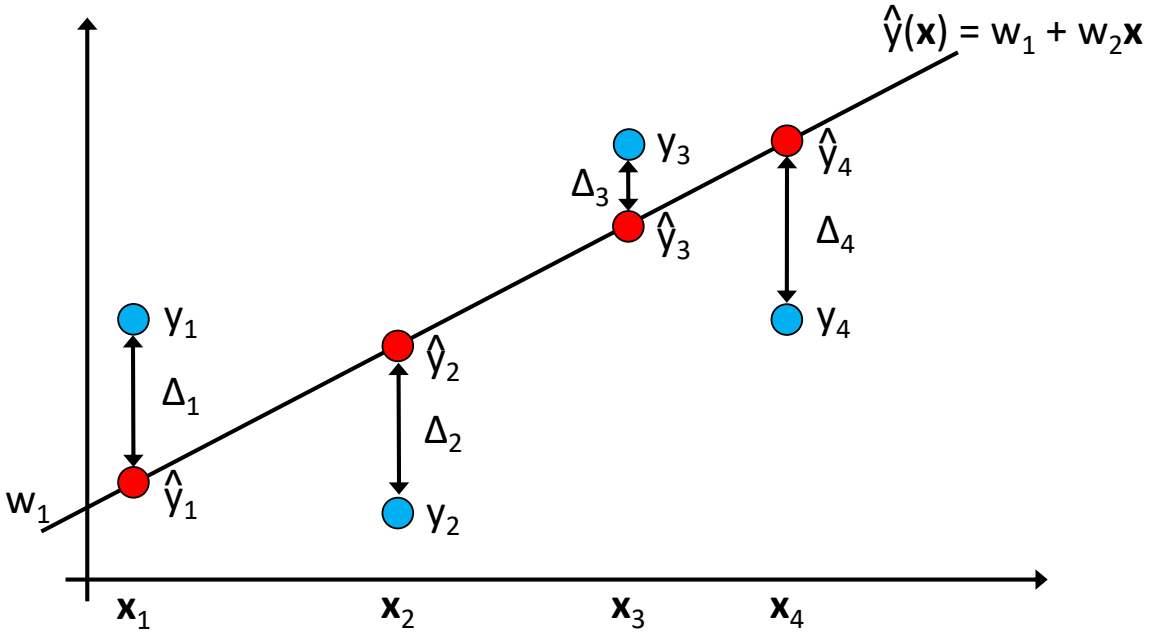
- **Learning problem**: Find $\mathbf{w}$ such that we minimize the residuals $\boldsymbol{\Delta}$:

$$\hat{\mathbf{w}} = \arg\min_{\mathbf{w}} \|\boldsymbol{\Delta}\|$$

defined by

$$\Delta_i = y_i - \hat{y}(\mathbf{x}_i; \mathbf{w}).$$

Regression



Example: Fit data with linear function $\hat{y}(\mathbf{x}) = w_1 + w_2\mathbf{x}$ by finding the parameters $\mathbf{w} = (w_1, w_2)$.

We define $\|\mathbf{\Delta}\|$ using the p-norm of $\mathbf{\Delta} \in \mathbb{R}^n$:

$$\|\mathbf{\Delta}\|_p = \left(\sum_{i=1}^{n} |\Delta_i|^p \right)^{1/p},$$

Choice of p determines the type of regression problem:

p	Learning Problem	Name
1	$\min_{\mathbf{w}} \sum_{i=1}^{n} \lvert\Delta_i\rvert$	L^1 (linear) optimization
2	$\min_{\mathbf{w}} \sum_{i=1}^{n} \Delta_i^2$	Least squares (Gauss $\sim$1800)
∞	$\min_{\mathbf{w}} \max_i \Delta_i$	Tschebyscheff regression

Loss function
Least-squares regression

- Most supervised machine learning problems can be formulated as the problem to minimize a **cost** or **loss** function $C(\mathbf{X}, \mathbf{Y}, \theta)$.
  - $\mathbf{X} \in \mathbb{R}^{N \times n}$ is the matrix of N input data points or features.
  - $\mathbf{Y} \in \mathbb{R}^{N \times I}$ are the labels. A label has I dimensions.
  - **Training data**: $(\mathbf{X}, \mathbf{Y}) = (\mathbf{x}_i, \mathbf{y}_i)_{i=1,\ldots,N}$
- Here: call $\theta = \mathbf{w}$ and consider univariate function regression – labels can be written as a vector $\mathbf{y} \in \mathbb{R}^N$.
  We choose the **mean squared error** as loss function:

$$C(\mathbf{X}, \mathbf{y}, \mathbf{w}) = \frac{1}{N} \sum_{i=1}^{N} \left(y_i - \hat{y}(\mathbf{x}_i; \mathbf{w}) \right)^2$$

- **Learning problem**: seek $\mathbf{w}$ such that the loss function is minimal:

$$\hat{\mathbf{w}} = \arg\min_{\mathbf{w}} \sum_{i=1}^{N} \left[y_i - \hat{y}(x_i; \mathbf{w}) \right]^2$$

The prefactor N^{-1} has no influence on $\hat{\mathbf{w}}$.

- Most supervised machine learning problems can be formulated as the problem to minimize a **cost** or **loss** function $C(\mathbf{X}, \mathbf{Y}, \theta)$.
  - $\mathbf{X} \in \mathbb{R}^{N \times n}$ is the matrix of N input data points or features.
  - $\mathbf{Y} \in \mathbb{R}^{N \times l}$ are the labels. A label has l dimensions.
  - **Training data**: $(\mathbf{X}, \mathbf{Y}) = (\mathbf{x}_i, \mathbf{y}_i)_{i=1,\ldots,N}$
- Here: call $\theta = \mathbf{w}$ and consider univariate function regression – labels can be written as a vector $\mathbf{y} \in \mathbb{R}^N$.
  We choose the **mean squared error** as loss function:

$$C(\mathbf{X}, \mathbf{y}, \mathbf{w}) = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}(\mathbf{x}_i; \mathbf{w}))^2$$

- **Learning problem**: seek $\mathbf{w}$ such that the loss function is minimal:

$$\hat{\mathbf{w}} = \arg\min_{\mathbf{w}} \sum_{i=1}^{N} [y_i - \hat{y}(x_i; \mathbf{w})]^2$$

The prefactor N^{-1} has no influence on $\hat{\mathbf{w}}$.

- Most supervised machine learning problems can be formulated as the problem to minimize a **cost** or **loss** function $C(\mathbf{X}, \mathbf{Y}, \theta)$.
  - $\mathbf{X} \in \mathbb{R}^{N \times n}$ is the matrix of N input data points or features.
  - $\mathbf{Y} \in \mathbb{R}^{N \times l}$ are the labels. A label has l dimensions.
  - **Training data**: $(\mathbf{X}, \mathbf{Y}) = (\mathbf{x}_i, \mathbf{y}_i)_{i=1,\ldots,N}$
- Here: call $\theta = \mathbf{w}$ and consider univariate function regression – labels can be written as a vector $\mathbf{y} \in \mathbb{R}^N$.
  We choose the **mean squared error** as loss function:

$$C(\mathbf{X}, \mathbf{y}, \mathbf{w}) = \frac{1}{N} \sum_{i=1}^{N} \left(y_i - \hat{y}(\mathbf{x}_i; \mathbf{w}) \right)^2$$

- **Learning problem**: seek $\mathbf{w}$ such that the loss function is minimal:

$$\hat{\mathbf{w}} = \arg\min_{\mathbf{w}} \sum_{i=1}^{N} \left[y_i - \hat{y}(x_i; \mathbf{w}) \right]^2$$

The prefactor N^{-1} has no influence on $\hat{\mathbf{w}}$.

Equivalent statistical interpretation (Gauss): Assume that observations y_i are produced from $\hat{y}(\mathbf{x}_i; \mathbf{w})$ with an additive measurement error that is *independent, identically distributed (iid)* from a *Normal distribution*.

$$y_i = \hat{y}(\mathbf{x}_i; \mathbf{w}) + \Delta_i \qquad \Delta_i \sim \mathcal{N}(0, 1) = \frac{1}{\sqrt{2\pi}} e^{-\Delta_i^2/2}$$

We seek the **maximum likelihood estimator** $\hat{\mathbf{w}} = \arg\max_{\mathbf{w}} L(\mathbf{X}, \mathbf{Y}, \mathbf{w})$ with

$$L(\mathbf{X}, \mathbf{Y}, \mathbf{w}) = \mathbb{P}[y_1, ..., y_N \mid \mathbf{w}] = \prod_{i=1}^{N} \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}(y_i - \hat{y}(\mathbf{x}_i; \mathbf{w}))^2}$$

This is equivalent with:

$$\begin{aligned} \arg\max L(\mathbf{X}, \mathbf{Y}, \mathbf{w}) &= \arg\max \log L(\mathbf{X}, \mathbf{Y}, \mathbf{w}) \\ &= \arg\max \sum_{i=1}^{N} -\frac{1}{2}(y_i - \hat{y}(\mathbf{x}_i; \mathbf{w}))^2 - \log\left(\sqrt{2\pi}\right) \\ &= \arg\min \sum_{i=1}^{N} (y_i - \hat{y}(\mathbf{x}_i; \mathbf{w}))^2 \end{aligned}$$

Equivalent statistical interpretation (Gauss): Assume that observations y_i are produced from $\hat{y}(\mathbf{x}_i; \mathbf{w})$ with an additive measurement error that is *independent, identically distributed (iid)* from a *Normal distribution*.

$$y_i = \hat{y}(\mathbf{x}_i; \mathbf{w}) + \Delta_i \qquad \Delta_i \sim \mathcal{N}(0, 1) = \frac{1}{\sqrt{2\pi}} e^{-\Delta_i^2/2}$$

We seek the **maximum likelihood estimator** $\hat{\mathbf{w}} = \arg\max_{\mathbf{w}} L(\mathbf{X}, \mathbf{Y}, \mathbf{w})$ with

$$L(\mathbf{X}, \mathbf{Y}, \mathbf{w}) = \mathbb{P}[y_1, ..., y_N \mid \mathbf{w}] = \prod_{i=1}^{N} \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}(y_i - \hat{y}(\mathbf{x}_i; \mathbf{w}))^2}$$

This is equivalent with:

$$\begin{aligned} \arg\max L(\mathbf{X}, \mathbf{Y}, \mathbf{w}) &= \arg\max \log L(\mathbf{X}, \mathbf{Y}, \mathbf{w}) \\ &= \arg\max \sum_{i=1}^{N} -\frac{1}{2}(y_i - \hat{y}(\mathbf{x}_i; \mathbf{w}))^2 - \log\left(\sqrt{2\pi}\right) \\ &= \arg\min \sum_{i=1}^{N} (y_i - \hat{y}(\mathbf{x}_i; \mathbf{w}))^2 \end{aligned}$$

Equivalent statistical interpretation (Gauss): Assume that observations y_i are produced from $\hat{y}(\mathbf{x}_i; \mathbf{w})$ with an additive measurement error that is *independent, identically distributed (iid)* from a *Normal distribution*.

$$y_i = \hat{y}(\mathbf{x}_i; \mathbf{w}) + \Delta_i \qquad \Delta_i \sim \mathcal{N}(0, 1) = \frac{1}{\sqrt{2\pi}} \mathrm{e}^{-\Delta_i^2/2}$$

We seek the **maximum likelihood estimator** $\hat{\mathbf{w}} = \arg\max_{\mathbf{w}} L(\mathbf{X}, \mathbf{Y}, \mathbf{w})$ with

$$L(\mathbf{X}, \mathbf{Y}, \mathbf{w}) = \mathbb{P}[y_1, ..., y_N \mid \mathbf{w}] = \prod_{i=1}^{N} \frac{1}{\sqrt{2\pi}} \mathrm{e}^{-\frac{1}{2}(y_i - \hat{y}(\mathbf{x}_i; \mathbf{w}))^2}$$

This is equivalent with:

$$\begin{aligned} \arg\max L(\mathbf{X}, \mathbf{Y}, \mathbf{w}) &= \arg\max \log L(\mathbf{X}, \mathbf{Y}, \mathbf{w}) \\ &= \arg\max \sum_{i=1}^{N} -\frac{1}{2} (y_i - \hat{y}(\mathbf{x}_i; \mathbf{w}))^2 - \log\left(\sqrt{2\pi}\right) \\ &= \arg\min \sum_{i=1}^{N} (y_i - \hat{y}(\mathbf{x}_i; \mathbf{w}))^2 \end{aligned}$$

Linear Least Squares Regression

- In a linear regression problem, the model has the form:

$$\hat{y}(\mathbf{x}_i; \mathbf{w}) = \mathbf{x}_i^\top \mathbf{w} = \sum_{j=1}^{n} x_{ij} w_j$$

- Matrix notation: $\hat{\mathbf{y}} = \mathbf{X}\mathbf{w}$. Linear least squares (LLS) regression problem:

$$\min_{\mathbf{w}} \|\mathbf{y} - \mathbf{X}\mathbf{w}\|_2^2$$

- **Featurization**: Starting from some initial data $\mathbf{r}_i \in \mathbb{R}^d$, we define n **basis functions** or **feature functions** $\phi_j : \mathbb{R}^d \to \mathbb{R}$. Thus every datapoint is featurized:

$$\mathbf{r}_i \to \mathbf{x}_i = (\phi_1(\mathbf{r}_i), ..., \phi_n(\mathbf{r}_i))^\top .$$

- Our linear regression model then has the form

$$\hat{y}(\mathbf{r}_i; \mathbf{w}) = w_1 \phi_1(\mathbf{r}_i) + ... + w_n \phi_n(\mathbf{r}_i)$$

  with can be nonlinear in $\mathbf{r}$.
- **Example**: with $\phi = \{1, r, r^2\}$ and $\mathbf{w} = (w_1, w_2, w_3)$ we can fit a quadratic function $f(r) = w_1 + w_2 r + w_3 r^2$.

Linear Least Squares Regression

- In a linear regression problem, the model has the form:

$$\hat{y}(\mathbf{x}_i; \mathbf{w}) = \mathbf{x}_i^\top \mathbf{w} = \sum_{j=1}^{n} x_{ij} w_j$$

- Matrix notation: $\hat{\mathbf{y}} = \mathbf{X}\mathbf{w}$. Linear least squares (LLS) regression problem:

$$\min_{\mathbf{w}} \|\mathbf{y} - \mathbf{X}\mathbf{w}\|_2^2$$

- **Featurization**: Starting from some initial data $\mathbf{r}_i \in \mathbb{R}^d$, we define n **basis functions** or **feature functions** $\phi_j : \mathbb{R}^d \to \mathbb{R}$. Thus every datapoint is featurized:

$$\mathbf{r}_i \to \mathbf{x}_i = (\phi_1(\mathbf{r}_i), ..., \phi_n(\mathbf{r}_i))^\top .$$

- Our linear regression model then has the form

$$\hat{y}(\mathbf{r}_i; \mathbf{w}) = w_1 \phi_1(\mathbf{r}_i) + ... + w_n \phi_n(\mathbf{r}_i)$$

  with can be nonlinear in $\mathbf{r}$.

- **Example**: with $\phi = \{1, r, r^2\}$ and $\mathbf{w} = (w_1, w_2, w_3)$ we can fit a quadratic function $f(r) = w_1 + w_2 r + w_3 r^2$.

Linear Least Squares Regression

- In a linear regression problem, the model has the form:

$$\hat{y}(\mathbf{x}_i; \mathbf{w}) = \mathbf{x}_i^\top \mathbf{w} = \sum_{j=1}^{n} x_{ij} w_j$$

- Matrix notation: $\hat{\mathbf{y}} = \mathbf{X}\mathbf{w}$. Linear least squares (LLS) regression problem:

$$\min_{\mathbf{w}} \|\mathbf{y} - \mathbf{X}\mathbf{w}\|_2^2$$

- **Featurization**: Starting from some initial data $\mathbf{r}_i \in \mathbb{R}^d$, we define n **basis functions** or **feature functions** $\phi_j : \mathbb{R}^d \to \mathbb{R}$. Thus every datapoint is featurized:

$$\mathbf{r}_i \to \mathbf{x}_i = (\phi_1(\mathbf{r}_i), ..., \phi_n(\mathbf{r}_i))^\top .$$

- Our linear regression model then has the form

$$\hat{y}(\mathbf{r}_i; \mathbf{w}) = w_1 \phi_1(\mathbf{r}_i) + ... + w_n \phi_n(\mathbf{r}_i)$$

  with can be nonlinear in $\mathbf{r}$.
- **Example**: with $\phi = \{1, r, r^2\}$ and $\mathbf{w} = (w_1, w_2, w_3)$ we can fit a quadratic function $f(r) = w_1 + w_2 r + w_3 r^2$.

The vector $\mathbf{w} \in \mathbb{R}^n$ is the solution to $\min \|\mathbf{y} - \mathbf{X}\mathbf{w}\|_2$ exactly if it fulfills the normal equations

$$\mathbf{X}^\top \mathbf{X}\mathbf{w} = \mathbf{X}^\top \mathbf{y}$$

The linear regression problem has a unique solution exactly if the rank of $\mathbf{X}$ is maximal, i.e. $\mathrm{rk}(\mathbf{X}) = n$.

Direct inversion (numerically unstable and inefficient):

$$\mathbf{w} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y} = \mathbf{X}^+ \mathbf{y}.$$

where $\mathbf{X}^+$ is the Moore-Penrose pseudoinverse of $\mathbf{X}$.
Defining the covariance matrices

$$\mathbf{C}_{XX} = \mathbf{X}^\top \mathbf{X}$$
$$\mathbf{C}_{XY} = \mathbf{X}^\top \mathbf{y}$$

(here $\mathbf{C}_{XY} \in \mathbb{R}^{n \times 1}$, but it is a matrix if we have multiple regression targets) the formal solution can be written as:

$$\mathbf{w} = \mathbf{C}_{XX}^{-1} \mathbf{C}_{XY}.$$

The vector $\mathbf{w} \in \mathbb{R}^n$ is the solution to $\min \|\mathbf{y} - \mathbf{X}\mathbf{w}\|_2$ exactly if it fulfills the normal equations

$$\mathbf{X}^\top \mathbf{X}\mathbf{w} = \mathbf{X}^\top \mathbf{y}$$

The linear regression problem has a unique solution exactly if the rank of $\mathbf{X}$ is maximal, i.e. $\mathrm{rk}(\mathbf{X}) = n$.
Direct inversion (numerically unstable and inefficient):

$$\mathbf{w} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y} = \mathbf{X}^+ \mathbf{y}.$$

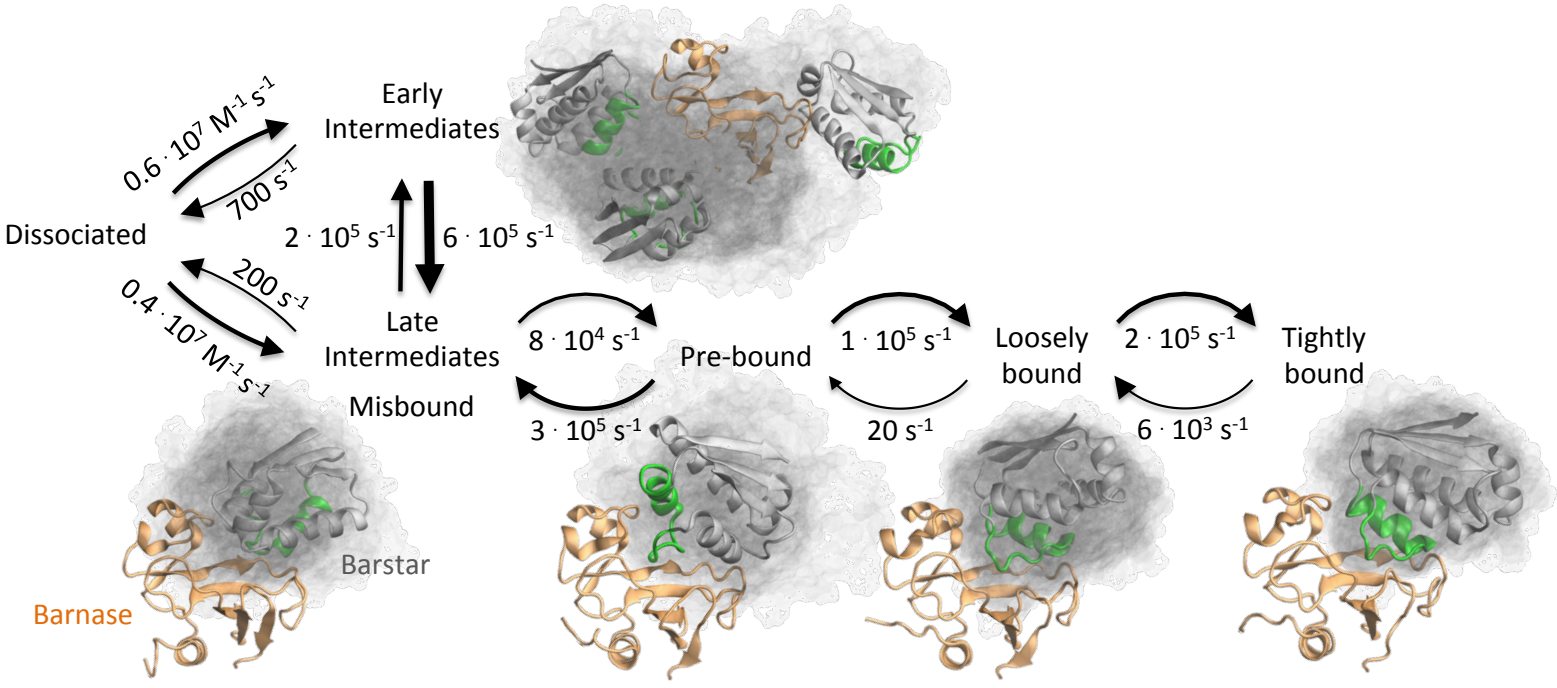
where $\mathbf{X}^+$ is the Moore-Penrose pseudoinverse of $\mathbf{X}$.
Defining the covariance matrices

$$\mathbf{C}_{XX} = \mathbf{X}^\top \mathbf{X}$$
$$\mathbf{C}_{XY} = \mathbf{X}^\top \mathbf{y}$$

(here $\mathbf{C}_{XY} \in \mathbb{R}^{n \times 1}$, but it is a matrix if we have multiple regression targets) the formal solution can be written as:

$$\mathbf{w} = \mathbf{C}_{XX}^{-1} \mathbf{C}_{XY}.$$

Example: Markov models



Plattner, Doerr, De Fabritiis, Noé, **Nature Chemistry** (2017)

Example: Markov models

- Define feature functions $\phi_i(\mathbf{x})$ with $i = 1, ..., n$. Transform time series:

$$X_{ij} = \phi_j(\mathbf{x}_i)$$
$$Y_{ij} = \phi_j(\mathbf{x}_{i+\tau})$$

  with feature matrices $\mathbf{X}, \mathbf{Y} \in \mathbf{R}^{m \times n}$ and $m = T - \tau$.

- Solve regression problem:

$$\min_{\mathbf{K}} \|\mathbf{Y} - \mathbf{X}\mathbf{K}\|_2$$

  in order to parametrize the Markovian dynamical model $\mathbf{K}$ that describes the time evolution:

$$\mathbf{Y} \approx \mathbf{X}\mathbf{K},$$

  or in other words $\mathbf{x}_{t+\tau}^\top \approx \mathbf{x}_t^\top \mathbf{K}$.

- Solution has the form (MSMs, TICA, EDMD, ...):

$$\mathbf{K} = \left(\mathbf{X}^\top \mathbf{X}\right)^{-1} \mathbf{X}^\top \mathbf{Y} = \mathbf{C}_{XX}^{-1} \mathbf{C}_{XY}$$

Example: Markov models

- Define feature functions $\phi_i(\mathbf{x})$ with $i = 1, ..., n$. Transform time series:

$$X_{ij} = \phi_j(\mathbf{x}_i)$$
$$Y_{ij} = \phi_j(\mathbf{x}_{i+\tau})$$

  with feature matrices $\mathbf{X}, \mathbf{Y} \in \mathbf{R}^{m \times n}$ and $m = T - \tau$.

- Solve regression problem:

$$\min_{\mathbf{K}} \|\mathbf{Y} - \mathbf{X}\mathbf{K}\|_2$$

  in order to parametrize the Markovian dynamical model $\mathbf{K}$ that describes the time evolution:

$$\mathbf{Y} \approx \mathbf{X}\mathbf{K},$$

  or in other words $\mathbf{x}_{t+\tau}^\top \approx \mathbf{x}_t^\top \mathbf{K}$.

- Solution has the form (MSMs, TICA, EDMD, ...):

$$\mathbf{K} = \left(\mathbf{X}^\top \mathbf{X}\right)^{-1} \mathbf{X}^\top \mathbf{Y} = \mathbf{C}_{XX}^{-1} \mathbf{C}_{XY}$$

Example: Markov models

- Define feature functions $\phi_i(\mathbf{x})$ with $i = 1, ..., n$. Transform time series:

$$X_{ij} = \phi_j(\mathbf{x}_i)$$
$$Y_{ij} = \phi_j(\mathbf{x}_{i+\tau})$$

  with feature matrices $\mathbf{X}, \mathbf{Y} \in \mathbf{R}^{m \times n}$ and $m = T - \tau$.
- Solve regression problem:

$$\min_{\mathbf{K}} \|\mathbf{Y} - \mathbf{X}\mathbf{K}\|_2$$

  in order to parametrize the Markovian dynamical model $\mathbf{K}$ that describes the time evolution:

$$\mathbf{Y} \approx \mathbf{X}\mathbf{K},$$

  or in other words $\mathbf{x}_{t+\tau}^\top \approx \mathbf{x}_t^\top \mathbf{K}$.
- Solution has the form (MSMs, TICA, EDMD, ...):

$$\mathbf{K} = \left(\mathbf{X}^\top \mathbf{X}\right)^{-1} \mathbf{X}^\top \mathbf{Y} = \mathbf{C}_{XX}^{-1} \mathbf{C}_{XY}$$

Markov state models

- Partition molecular state space Ω into substates $S_1, \ldots, S_n$. Define characteristic functions:

$$\phi_j(\mathbf{x}) = \begin{cases} 1 & \mathbf{x} \in S_j \\ 0 & \text{else.} \end{cases}$$

- Correlation matrices $\mathbf{C}_{XX}$ and $\mathbf{C}_{XY}$ then evaluate to:

$$(C_{XX})_{ij} = \sum_{t=1}^{m} \phi_i(\mathbf{x}_t)\phi_j(\mathbf{x}_t) = \delta_{ij} N_i \quad (1)$$

$$(C_{XY})_{ij} = \sum_{t=1}^{m} \phi_i(\mathbf{x}_t)\phi_j(\mathbf{x}_{t+\tau}) = N_{ij} \quad (2)$$

  - N_i: number of times the trajectory was in state i.
  - N_{ij}: number of transitions $i \to j$ in time interval τ.

- Markov model $\mathbf{K}$ can be written as:

$$K_{ij} = N_{ij}/N_i$$

Markov state models

- Partition molecular state space Ω into substates $S_1, \ldots, S_n$. Define characteristic functions:

$$\phi_j(\mathbf{x}) = \begin{cases} 1 & \mathbf{x} \in S_j \\ 0 & \text{else.} \end{cases}$$

- Correlation matrices $\mathbf{C}_{XX}$ and $\mathbf{C}_{XY}$ then evaluate to:

$$(C_{XX})_{ij} = \sum_{t=1}^{m} \phi_i(\mathbf{x}_t)\phi_j(\mathbf{x}_t) = \delta_{ij} N_i \tag{1}$$

$$(C_{XY})_{ij} = \sum_{t=1}^{m} \phi_i(\mathbf{x}_t)\phi_j(\mathbf{x}_{t+\tau}) = N_{ij} \tag{2}$$

  - N_i: number of times the trajectory was in state i
  - N_{ij}: number of transitions $i \to j$ in time interval τ.

- Markov model $\mathbf{K}$ can be written as:

$$K_{ij} = N_{ij}/N_i$$

Markov state models

- Partition molecular state space Ω into substates $S_1, ..., S_n$. Define characteristic functions:

$$\phi_j(\mathbf{x}) = \begin{cases} 1 & \mathbf{x} \in S_j \\ 0 & \text{else.} \end{cases}$$

- Correlation matrices $\mathbf{C}_{XX}$ and $\mathbf{C}_{XY}$ then evaluate to:

$$(C_{XX})_{ij} = \sum_{t=1}^{m} \phi_i(\mathbf{x}_t)\phi_j(\mathbf{x}_t) = \delta_{ij} N_i \tag{1}$$

$$(C_{XY})_{ij} = \sum_{t=1}^{m} \phi_i(\mathbf{x}_t)\phi_j(\mathbf{x}_{t+\tau}) = N_{ij} \tag{2}$$

  - N_i: number of times the trajectory was in state i
  - N_{ij}: number of transitions $i \to j$ in time interval τ.

- Markov model $\mathbf{K}$ can be written as:

$$K_{ij} = N_{ij}/N_i$$

Markov state models

Three different optimization principles. $X_{ij} = \phi_j(\mathbf{x}_i), Y_{ij} = \phi_j(\mathbf{x}_{i+\tau})$.

1. Minimum regression error

$$\hat{\mathbf{K}} = \min_{\mathbf{K}} \|\mathbf{Y} - \mathbf{X}\mathbf{K}\|_2$$

2. Maximum likelihood

$$\hat{\mathbf{K}} = \max_{\mathbf{K}} \prod_{i,j} k_{ij}^{N_{ij}}$$

3. Variational approach of conformation dynamics: Parametrize eigenvalue decomposition $\mathbf{P} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^{-1}$ and maximize eigenvalues by:

$$\max_{\mathbf{U}} \|\mathbf{M}(\mathbf{U})\|_F^2$$

with $\mathbf{M} = \left(\mathbf{U}^\top \mathbf{C}_{XX} \mathbf{U}\right)^{-\frac{1}{2}} \mathbf{U}^\top \mathbf{C}_{XY} \mathbf{U} \left(\mathbf{U}^\top \mathbf{C}_{XX} \mathbf{U}\right)^{-\frac{1}{2}}$ and $\mathbf{\Lambda} = \mathrm{diag}\left(M_{11}, ..., M_{nn}\right)$

For fixed featurization ϕ, all three principles result in:

$$\mathbf{K} = \mathbf{C}_{XX}^{-1} \mathbf{C}_{XY}$$

Markov state models

Three different optimization principles. $X_{ij} = \phi_j(\mathbf{x}_i), Y_{ij} = \phi_j(\mathbf{x}_{i+\tau})$.

1. Minimum regression error

$$\hat{\mathbf{K}} = \min_{\mathbf{K}} \|\mathbf{Y} - \mathbf{X}\mathbf{K}\|_2$$

2. Maximum likelihood

$$\hat{\mathbf{K}} = \max_{\mathbf{K}} \prod_{i,j} k_{ij}^{N_{ij}}$$

3. Variational approach of conformation dynamics: Parametrize eigenvalue decomposition $\mathbf{P} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^{-1}$ and maximize eigenvalues by:

$$\max_{\mathbf{U}} \|\mathbf{M}(\mathbf{U})\|_F^2$$

with $\mathbf{M} = \left(\mathbf{U}^\top \mathbf{C}_{XX} \mathbf{U}\right)^{-\frac{1}{2}} \mathbf{U}^\top \mathbf{C}_{XY} \mathbf{U} \left(\mathbf{U}^\top \mathbf{C}_{XX} \mathbf{U}\right)^{-\frac{1}{2}}$ and $\mathbf{\Lambda} = \mathrm{diag}\left(M_{11}, ..., M_{nn}\right)$

For fixed featurization ϕ, all three principles result in:

$$\mathbf{K} = \mathbf{C}_{XX}^{-1} \mathbf{C}_{XY}$$

Markov state models

Three different optimization principles. $X_{ij} = \phi_j(\mathbf{x}_i), Y_{ij} = \phi_j(\mathbf{x}_{i+\tau})$.

1. Minimum regression error

$$\hat{\mathbf{K}} = \min_{\mathbf{K}} \|\mathbf{Y} - \mathbf{X}\mathbf{K}\|_2$$

2. Maximum likelihood

$$\hat{\mathbf{K}} = \max_{\mathbf{K}} \prod_{i,j} k_{ij}^{N_{ij}}$$

3. Variational approach of conformation dynamics: Parametrize eigenvalue decomposition $\mathbf{P} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^{-1}$ and maximize eigenvalues by:

$$\max_{\mathbf{U}} \|\mathbf{M}(\mathbf{U})\|_F^2$$

with $\mathbf{M} = \left(\mathbf{U}^\top \mathbf{C}_{XX} \mathbf{U}\right)^{-\frac{1}{2}} \mathbf{U}^\top \mathbf{C}_{XY} \mathbf{U} \left(\mathbf{U}^\top \mathbf{C}_{XX} \mathbf{U}\right)^{-\frac{1}{2}}$ and $\mathbf{\Lambda} = \mathrm{diag}\left(M_{11}, ..., M_{nn}\right)$

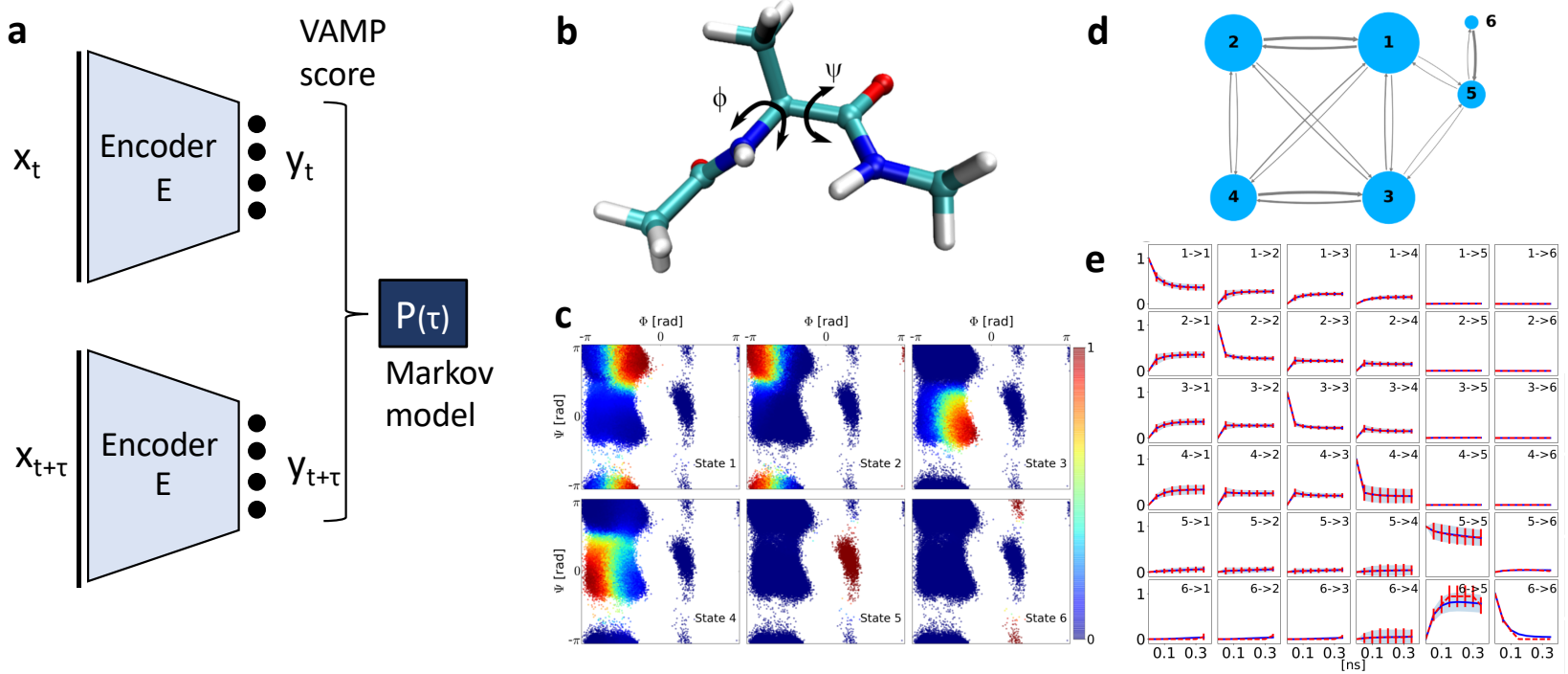
For fixed featurization ϕ, all three principles result in:

$$\mathbf{K} = \mathbf{C}_{XX}^{-1} \mathbf{C}_{XY}$$

Markov state models with feature learning

For learned featurization ϕ:

1. Minimum regression error $\min_{\mathbf{K}} \|\mathbf{Y} - \mathbf{X}\mathbf{K}\|_2 = 0$ for trivial solution (e.g., $\phi = 1$)
2. Maximum likelihood $\max_{\mathbf{K}} \prod_{i,j} k_{ij}^{N_{ij}} = 1$ for trivial solution (assign all configurations to one state).
3. Variational approach of conformation dynamics: Works $\rightarrow$ **VAMPnets**.



Mardt, Pasquali, Wu, Noé, **Nature Commun.** (2018)

Back to Linear Least Squares

- In a linear regression problem, the model has the form:

$$\hat{y}(\mathbf{x}_i; \mathbf{w}) = \mathbf{x}_i^\top \mathbf{w} = \sum_{j=1}^{n} x_{ij} w_j$$

- Matrix notation: $\hat{\mathbf{y}} = \mathbf{X}\mathbf{w}$. Linear least squares (LLS) regression problem:

$$\min_{\mathbf{w}} \|\mathbf{y} - \mathbf{X}\mathbf{w}\|_2^2$$

Validation and hyperparameter selection

- **Validation**: LLS solution gives us the **in-sample training error**:

$$E_{\text{in}} = C(\mathbf{X}^{\text{train}}, \mathbf{y}^{\text{train}}, \hat{\mathbf{w}}) = \frac{1}{N_{\text{in}}} \left\| \mathbf{y}^{\text{train}} - \mathbf{X}^{\text{train}} \hat{\mathbf{w}} \right\|_2 ,$$

  but we would like to validate how good the learnt model **predicts** an independent data set, i.e. the out-of-sample **validation or test error** E_{out}:

$$E_{\text{out}} = C(\mathbf{X}^{\text{val}}, \mathbf{y}^{\text{val}}, \hat{\mathbf{w}}) = \frac{1}{N_{\text{out}}} \left\| \mathbf{y}^{\text{val}} - \mathbf{X}^{\text{val}} \hat{\mathbf{w}} \right\|_2 ,$$

- **Hyperparameter selection**: Hyperparameters cannot not be obtained from the learning algorithm (here LLS). For example, the number of type of feature functions ϕ .
- **Example**: The type of function ϕ used for training cannot be determined by minimizing the training error. For example, the model

$$\hat{y}(\mathbf{x}) = \sum_{i=1}^{N} w_i 1_{\mathbf{x}_i}(\mathbf{x}) \quad \text{with} \quad 1_{\mathbf{x}_i}(\mathbf{x}) = \begin{cases} 1 & \mathbf{x} = \mathbf{x}_i \\ 0 & \mathbf{x} \neq \mathbf{x}_i \end{cases}$$

  has zero training error, but predicts $f(\mathbf{x}) = 0$ for every point $\mathbf{x}$ not in the training set.

Validation and hyperparameter selection

- **Validation**: LLS solution gives us the **in-sample training error**:

$$E_{\text{in}} = C(\mathbf{X}^{\text{train}}, \mathbf{y}^{\text{train}}, \hat{\mathbf{w}}) = \frac{1}{N_{\text{in}}} \left\| \mathbf{y}^{\text{train}} - \mathbf{X}^{\text{train}} \hat{\mathbf{w}} \right\|_2 ,$$

  but we would like to validate how good the learnt model **predicts** an independent data set, i.e. the out-of-sample **validation or test error** E_{out}:

$$E_{\text{out}} = C(\mathbf{X}^{\text{val}}, \mathbf{y}^{\text{val}}, \hat{\mathbf{w}}) = \frac{1}{N_{\text{out}}} \left\| \mathbf{y}^{\text{val}} - \mathbf{X}^{\text{val}} \hat{\mathbf{w}} \right\|_2 ,$$

- **Hyperparameter selection**: Hyperparameters cannot not be obtained from the learning algorithm (here LLS). For example, the number of type of feature functions ϕ .
- **Example**: The type of function ϕ used for training cannot be determined by minimizing the training error. For example, the model

$$\hat{y}(\mathbf{x}) = \sum_{i=1}^{N} w_i 1_{\mathbf{x}_i}(\mathbf{x}) \quad \text{with} \quad 1_{\mathbf{x}_i}(\mathbf{x}) = \begin{cases} 1 & \mathbf{x} = \mathbf{x}_i \\ 0 & \mathbf{x} \neq \mathbf{x}_i \end{cases}$$

  has zero training error, but predicts $f(\mathbf{x}) = 0$ for every point $\mathbf{x}$ not in the training set.

Validation and hyperparameter selection

- **Validation**: LLS solution gives us the **in-sample training error**:

$$E_{\text{in}} = C(\mathbf{X}^{\text{train}}, \mathbf{y}^{\text{train}}, \hat{\mathbf{w}}) = \frac{1}{N_{\text{in}}} \left\| \mathbf{y}^{\text{train}} - \mathbf{X}^{\text{train}} \hat{\mathbf{w}} \right\|_2 ,$$

  but we would like to validate how good the learnt model **predicts** an independent data set, i.e. the out-of-sample **validation or test error** E_{out}:

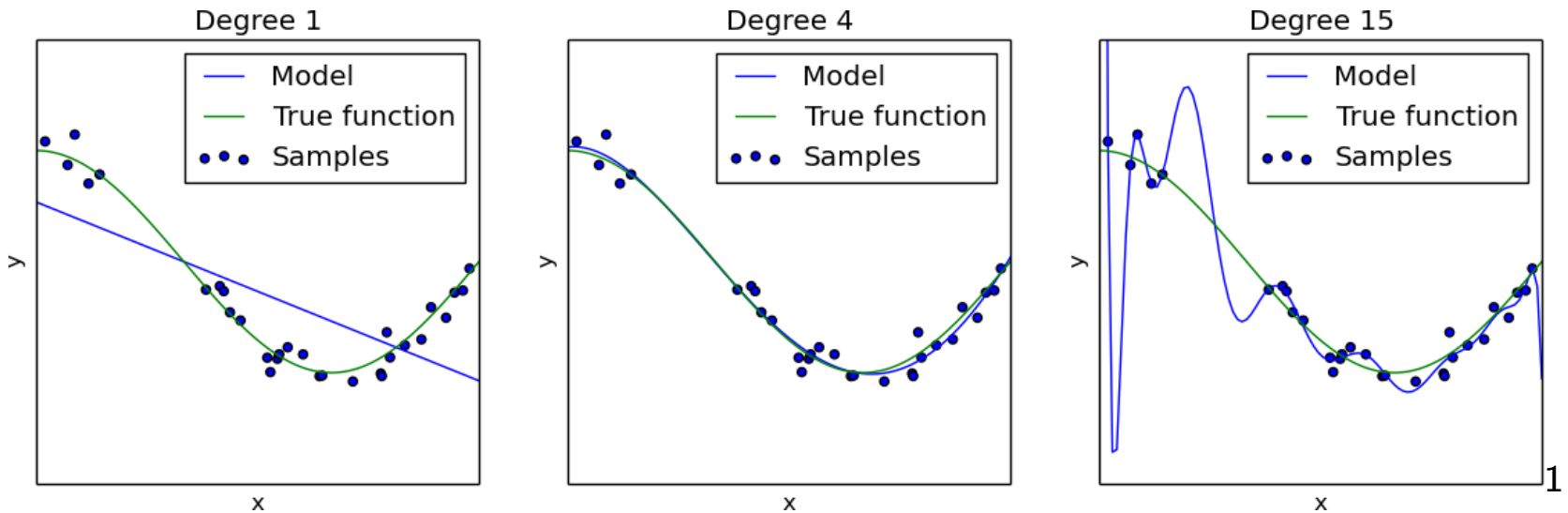
$$E_{\text{out}} = C(\mathbf{X}^{\text{val}}, \mathbf{y}^{\text{val}}, \hat{\mathbf{w}}) = \frac{1}{N_{\text{out}}} \left\| \mathbf{y}^{\text{val}} - \mathbf{X}^{\text{val}} \hat{\mathbf{w}} \right\|_2 ,$$

- **Hyperparameter selection**: Hyperparameters cannot not be obtained from the learning algorithm (here LLS). For example, the number of type of feature functions ϕ .
- **Example**: The type of function ϕ used for training cannot be determined by minimizing the training error. For example, the model

$$\hat{y}(\mathbf{x}) = \sum_{i=1}^{N} w_i 1_{\mathbf{x}_i}(\mathbf{x}) \quad \text{with} \quad 1_{\mathbf{x}_i}(\mathbf{x}) = \begin{cases} 1 & \mathbf{x} = \mathbf{x}_i \\ 0 & \mathbf{x} \neq \mathbf{x}_i \end{cases}$$

  has zero training error, but predicts $f(\mathbf{x}) = 0$ for every point $\mathbf{x}$ not in the training set.

Underfitting vs. Overfitting



[1]

[1] From http://scikit-learn.org

Validation

- **Data-based validation** is an effective way to solve the hyperparameter selection problem:
- Divide dataset into
  - **training set** $(\mathbf{X}^{\text{train}}, \mathbf{y}^{\text{train}})$
  - **validation set** $(\mathbf{X}^{\text{val}}, \mathbf{y}^{\text{val}})$.
- Learn parameters using the training set:

$$\hat{\mathbf{w}} = \arg\min_{\mathbf{w}} \left\| \mathbf{y}^{\text{train}} - \mathbf{X}^{\text{train}} \mathbf{w} \right\|_2$$

  The resulting residual $E_{\text{in}} = N_{\text{in}}^{-1} \left\| \mathbf{y}^{\text{train}} - \mathbf{X}^{\text{train}} \hat{\mathbf{w}} \right\|_2$ is the **training error** or **training loss**.
- The error of the learnt model in predicting data not used for the training,

$$E_{\text{out}} = N_{\text{out}}^{-1} \left\| \mathbf{y}^{\text{val}} - \mathbf{X}^{\text{val}} \hat{\mathbf{w}} \right\|_2$$

  is called the **validation** or **error/loss**. It provides a metric to validate how well the model generalizes to new data
- **Choose hyperparameters** by minimizing the validation error. (careful: selecting hyperparameters *and* computing E_{out} requires a third *test set* or a more advanced validation routine).

Validation

- **Data-based validation** is an effective way to solve the hyperparameter selection problem:
- Divide dataset into
  - **training set** $(\mathbf{X}^{\text{train}}, \mathbf{y}^{\text{train}})$
  - **validation set** $(\mathbf{X}^{\text{val}}, \mathbf{y}^{\text{val}})$.
- Learn parameters using the training set:

$$\hat{\mathbf{w}} = \arg\min_{\mathbf{w}} \left\| \mathbf{y}^{\text{train}} - \mathbf{X}^{\text{train}} \mathbf{w} \right\|_2$$

  The resulting residual $E_{\text{in}} = N_{\text{in}}^{-1} \left\| \mathbf{y}^{\text{train}} - \mathbf{X}^{\text{train}} \hat{\mathbf{w}} \right\|_2$ is the **training error** or **training loss**.
- The error of the learnt model in predicting data not used for the training,

$$E_{\text{out}} = N_{\text{out}}^{-1} \left\| \mathbf{y}^{\text{val}} - \mathbf{X}^{\text{val}} \hat{\mathbf{w}} \right\|_2$$

  is called the **validation** or **error/loss**. It provides a metric to validate how well the model generalizes to new data
- **Choose hyperparameters** by minimizing the validation error. (careful: selecting hyperparameters *and* computing E_{out} requires a third *test set* or a more advanced validation routine).

Validation

- **Data-based validation** is an effective way to solve the hyperparameter selection problem:
- Divide dataset into
  - **training set** $(\mathbf{X}^{\text{train}}, \mathbf{y}^{\text{train}})$
  - **validation set** $(\mathbf{X}^{\text{val}}, \mathbf{y}^{\text{val}})$.
- Learn parameters using the training set:

$$\hat{\mathbf{w}} = \arg\min_{\mathbf{w}} \left\| \mathbf{y}^{\text{train}} - \mathbf{X}^{\text{train}} \mathbf{w} \right\|_2$$

  The resulting residual $E_{\text{in}} = N_{\text{in}}^{-1} \left\| \mathbf{y}^{\text{train}} - \mathbf{X}^{\text{train}} \hat{\mathbf{w}} \right\|_2$ is the **training error** or **training loss**.
- The error of the learnt model in predicting data not used for the training,

$$E_{\text{out}} = N_{\text{out}}^{-1} \left\| \mathbf{y}^{\text{val}} - \mathbf{X}^{\text{val}} \hat{\mathbf{w}} \right\|_2$$

  is called the **validation** or **error/loss**. It provides a metric to validate how well the model generalizes to new data
- **Choose hyperparameters** by minimizing the validation error. (careful: selecting hyperparameters *and* computing E_{out} requires a third *test set* or a more advanced validation routine).

Validation

- **Data-based validation** is an effective way to solve the hyperparameter selection problem:
- Divide dataset into
  - **training set** $(\mathbf{X}^{\text{train}}, \mathbf{y}^{\text{train}})$
  - **validation set** $(\mathbf{X}^{\text{val}}, \mathbf{y}^{\text{val}})$.
- Learn parameters using the training set:

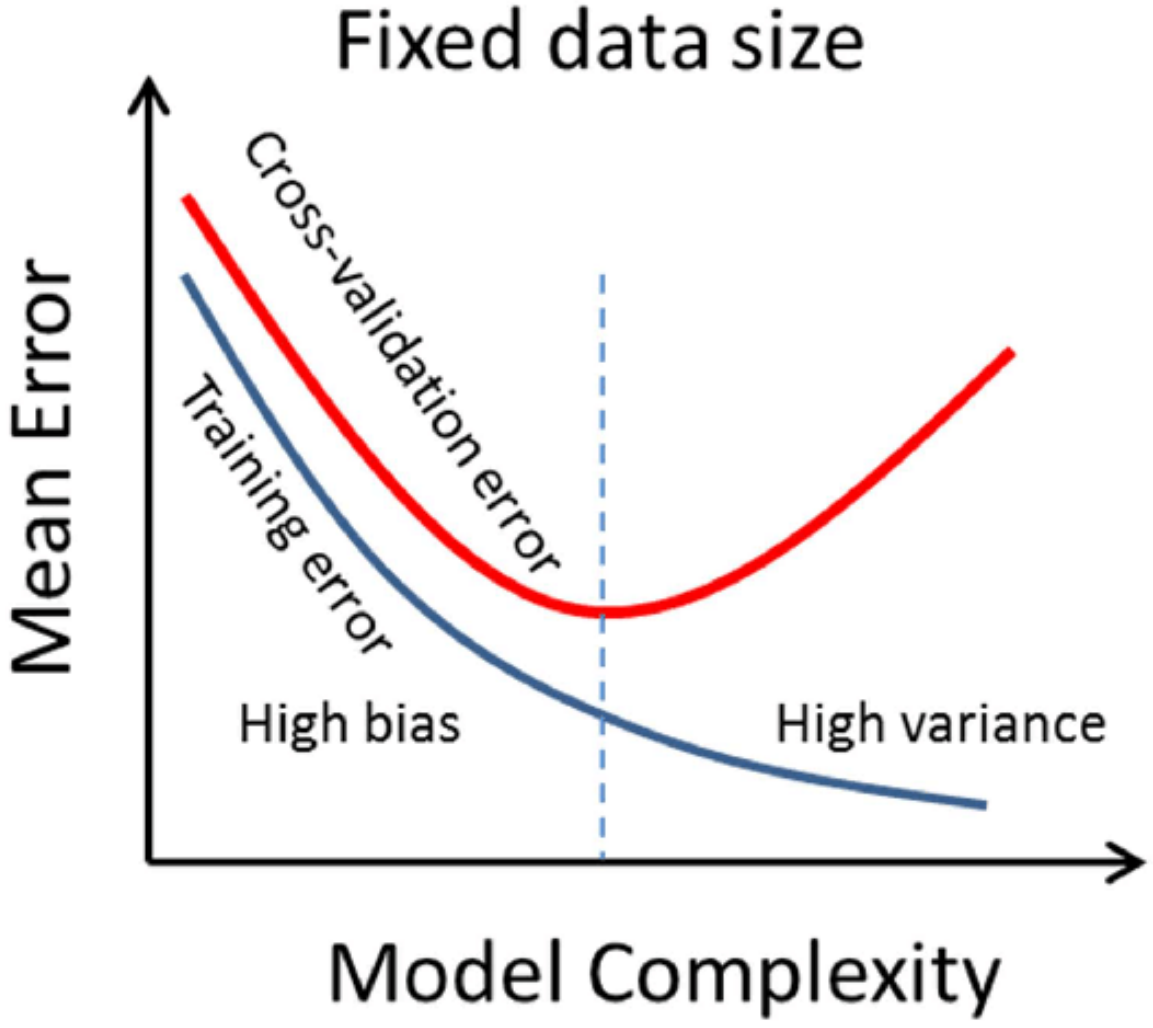
$$\hat{\mathbf{w}} = \arg\min_{\mathbf{w}} \left\| \mathbf{y}^{\text{train}} - \mathbf{X}^{\text{train}} \mathbf{w} \right\|_2$$

  The resulting residual $E_{\text{in}} = N_{\text{in}}^{-1} \left\| \mathbf{y}^{\text{train}} - \mathbf{X}^{\text{train}} \hat{\mathbf{w}} \right\|_2$ is the **training error** or **training loss**.
- The error of the learnt model in predicting data not used for the training,

$$E_{\text{out}} = N_{\text{out}}^{-1} \left\| \mathbf{y}^{\text{val}} - \mathbf{X}^{\text{val}} \hat{\mathbf{w}} \right\|_2$$

  is called the **validation** or **error/loss**. It provides a metric to validate how well the model generalizes to new data
- **Choose hyperparameters** by minimizing the validation error. (careful: selecting hyperparameters *and* computing E_{out} requires a third *test set* or a more advanced validation routine).

Underfitting vs. Overfitting



Cross-validation

- Pathological division where rare events / outliers are included only in training or validation set can lead to undesirable behavior.
- Methods to "shuffle" training and test data to reduce the bias from the data splitting.
- **Cross-validation** is a simple and widely used approach:
  1. Split the data into k nonoverlapping folds $(\mathbf{X}^i, \mathbf{y}^i)$. The complementary sets are $(\mathbf{X}^{-i}, \mathbf{y}^{-i})$ with sizes N^{-i}.
  2. For each fold i:
     1. Train learning algorithm on training data:
     $$\hat{\mathbf{w}}^i = \arg\min_{\mathbf{w}} \left\| \mathbf{y}^{-i} - \mathbf{X}^{-i}\mathbf{w} \right\|_2$$
     2. Compute validation error:
     $$E^i_{\text{out}} = \frac{1}{N^{-i}} \left\| \mathbf{y}^i - \mathbf{X}^i \hat{\mathbf{w}}^i \right\|_2$$
  3. Cross-validation error is then given by:
  $$E_{\text{out}} = \frac{1}{k} \sum_{i=1}^{k} E^k_{\text{out}}.$$

Cross-validation

- Pathological division where rare events / outliers are included only in training or validation set can lead to undesirable behavior.
- Methods to "shuffle" training and test data to reduce the bias from the data splitting.
- **Cross-validation** is a simple and widely used approach:
  1. Split the data into k nonoverlapping folds $(\mathbf{X}^i, \mathbf{y}^i)$. The complementary sets are $(\mathbf{X}^{-i}, \mathbf{y}^{-i})$ with sizes N^{-i}.
  2. For each fold i:
     1. Train learning algorithm on training data:
     $$\hat{\mathbf{w}}^i = \arg\min_{\mathbf{w}} \left\| \mathbf{y}^{-i} - \mathbf{X}^{-i}\mathbf{w} \right\|_2$$
     2. Compute validation error:
     $$E^i_{\text{out}} = \frac{1}{N^{-i}} \left\| \mathbf{y}^i - \mathbf{X}^i \hat{\mathbf{w}}^i \right\|_2$$
  3. Cross-validation error is then given by:
  $$E_{\text{out}} = \frac{1}{k} \sum_{i=1}^{k} E^k_{\text{out}}.$$

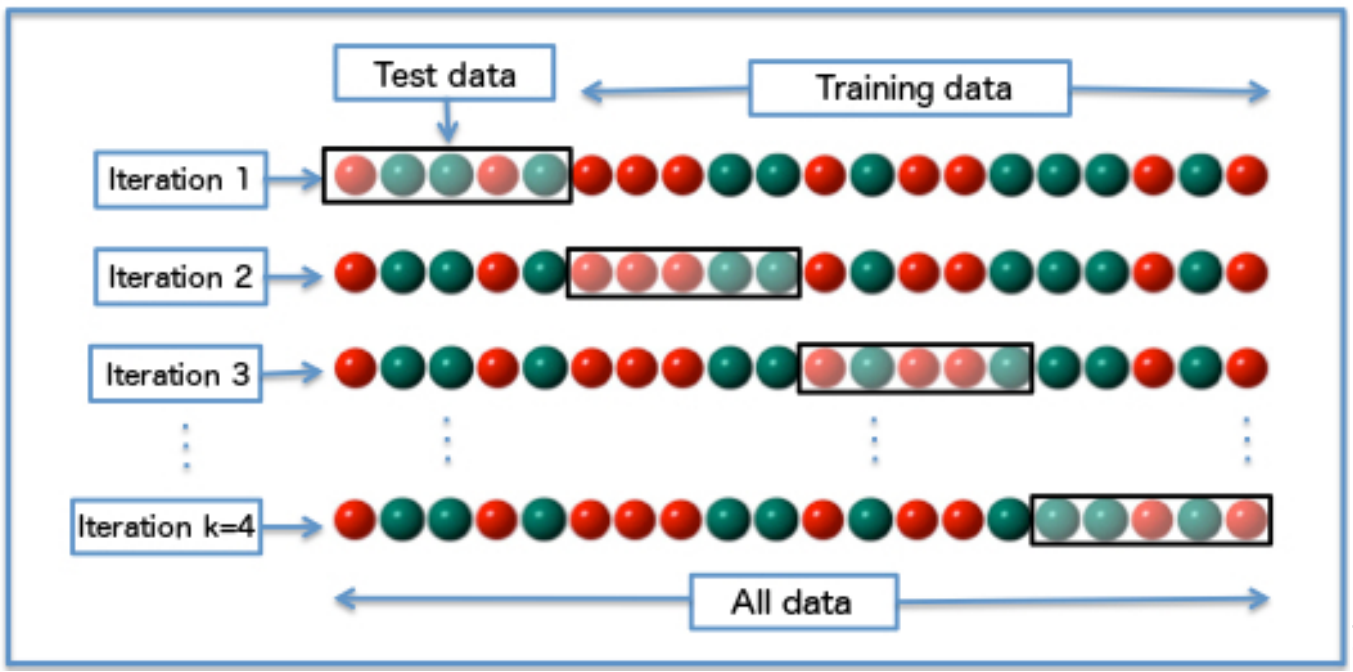
Cross-validation

- Pathological division where rare events / outliers are included only in training or validation set can lead to undesirable behavior.
- Methods to "shuffle" training and test data to reduce the bias from the data splitting.
- **Cross-validation** is a simple and widely used approach:
  1. Split the data into k nonoverlapping folds $(\mathbf{X}^i, \mathbf{y}^i)$. The complementary sets are $(\mathbf{X}^{-i}, \mathbf{y}^{-i})$ with sizes N^{-i}.
  2. For each fold i:
     1. Train learning algorithm on training data:
     $$\hat{\mathbf{w}}^i = \arg\min_{\mathbf{w}} \left\| \mathbf{y}^{-i} - \mathbf{X}^{-i}\mathbf{w} \right\|_2$$
     2. Compute validation error:
     $$E^i_{\text{out}} = \frac{1}{N^{-i}} \left\| \mathbf{y}^i - \mathbf{X}^i \hat{\mathbf{w}}^i \right\|_2$$
  3. Cross-validation error is then given by:
  $$E_{\text{out}} = \frac{1}{k} \sum_{i=1}^{k} E^k_{\text{out}}.$$

Cross-validation

- Pathological division where rare events / outliers are included only in training or validation set can lead to undesirable behavior.
- Methods to "shuffle" training and test data to reduce the bias from the data splitting.
- **Cross-validation** is a simple and widely used approach:
  1. Split the data into k nonoverlapping folds $(\mathbf{X}^i, \mathbf{y}^i)$. The complementary sets are $(\mathbf{X}^{-i}, \mathbf{y}^{-i})$ with sizes N^{-i}.
  2. For each fold i:
     1. Train learning algorithm on training data:
     $$\hat{\mathbf{w}}^i = \arg\min_{\mathbf{w}} \left\| \mathbf{y}^{-i} - \mathbf{X}^{-i}\mathbf{w} \right\|_2$$
     2. Compute validation error:
     $$E^i_{\text{out}} = \frac{1}{N^{-i}} \left\| \mathbf{y}^i - \mathbf{X}^i \hat{\mathbf{w}}^i \right\|_2$$
  3. Cross-validation error is then given by:
  $$E_{\text{out}} = \frac{1}{k} \sum_{i=1}^{k} E^k_{\text{out}}.$$

Cross-Validation



[1]

[1] From https://en.wikipedia.org/wiki/Cross-validation_(statistics)

We now explicitly distinguish between the true function f that is sampled by a given set of observations $(\mathbf{x}_i, y_i)$:

$$y_i = f(\mathbf{x}_i) + \Delta_i \qquad \Delta_i \sim \mathcal{N}(0, 1)$$

and the estimator $\hat{\mathbf{y}}(\mathbf{x}; \mathbf{w})$.

Learning problem

Learn function $f(\mathbf{x})$ by selecting function $\hat{y}(\mathbf{x})$ from a hypothesis set $\mathscr{H}$, which (in some sense) performs a best approximation $\hat{y} \approx f$.

Prediction problem

How can the learning problem be meaningfully defined if $f(x)$ can, in principle, take any value on *unobserved* inputs?

Answer: a meaningful definition of learning is that the fitted model will perform approximately as well in predicting unseen data as it did in approximating training data ($E_{\text{in}} \approx E_{\text{out}}$).

We now explicitly distinguish between the true function f that is sampled by a given set of observations $(\mathbf{x}_i, y_i)$:

$$y_i = f(\mathbf{x}_i) + \Delta_i \qquad \Delta_i \sim \mathcal{N}(0, 1)$$

and the estimator $\hat{\mathbf{y}}(\mathbf{x}; \mathbf{w})$.

Learning problem

Learn function $f(\mathbf{x})$ by selecting function $\hat{y}(\mathbf{x})$ from a hypothesis set $\mathscr{H}$, which (in some sense) performs a best approximation $\hat{y} \approx f$.

Prediction problem

How can the learning problem be meaningfully defined if $f(x)$ can, in principle, take any value on *unobserved* inputs?

Answer: a meaningful definition of learning is that the fitted model will perform approximately as well in predicting unseen data as it did in approximating training data ($E_{\text{in}} \approx E_{\text{out}}$).

We now explicitly distinguish between the true function f that is sampled by a given set of observations $(\mathbf{x}_i, y_i)$:

$$y_i = f(\mathbf{x}_i) + \Delta_i \qquad \Delta_i \sim \mathcal{N}(0, 1)$$

and the estimator $\hat{\mathbf{y}}(\mathbf{x}; \mathbf{w})$.

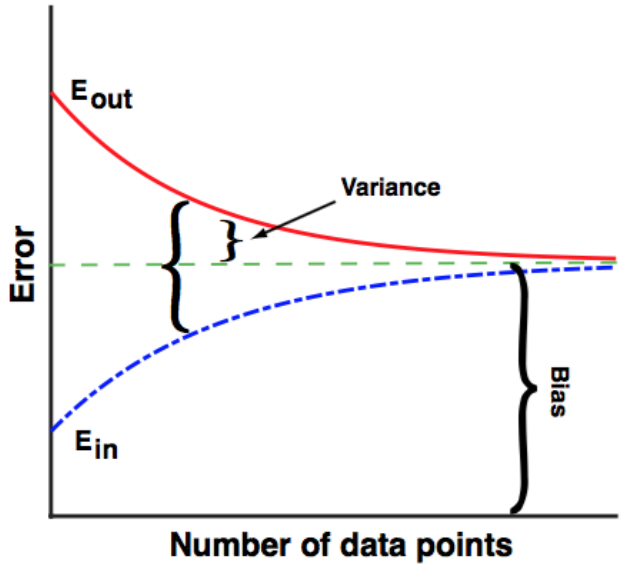
Learning problem

Learn function $f(\mathbf{x})$ by selecting function $\hat{y}(\mathbf{x})$ from a hypothesis set $\mathscr{H}$, which (in some sense) performs a best approximation $\hat{y} \approx f$.

Prediction problem

How can the learning problem be meaningfully defined if $f(x)$ can, in principle, take any value on *unobserved* inputs?

Answer: a meaningful definition of learning is that the fitted model will perform approximately as well in predicting unseen data as it did in approximating training data ($E_{\text{in}} \approx E_{\text{out}}$).



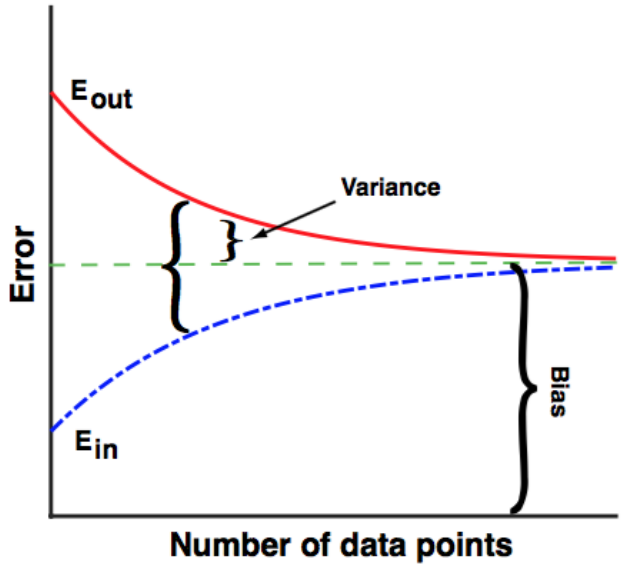
[1]

We assume that the true function f is sufficiently complicated so that we cannot learn it exactly, i.e.

$$\sum_{i=1}^{N} \left(f(\mathbf{x}_i) - \hat{\mathbf{y}}(\mathbf{x}_i; \mathbf{w})\right)^2 > 0 \ \forall \mathbf{w}$$

Even in the limit $N \to \infty$ we maintain an **asymptotic error** $E_{\text{in}} = E_{\text{out}}$, called the **model bias** $\to$ property of the function class $\mathscr{H}$.

[1] From Mehta et al, arXiv:1803.08823v1

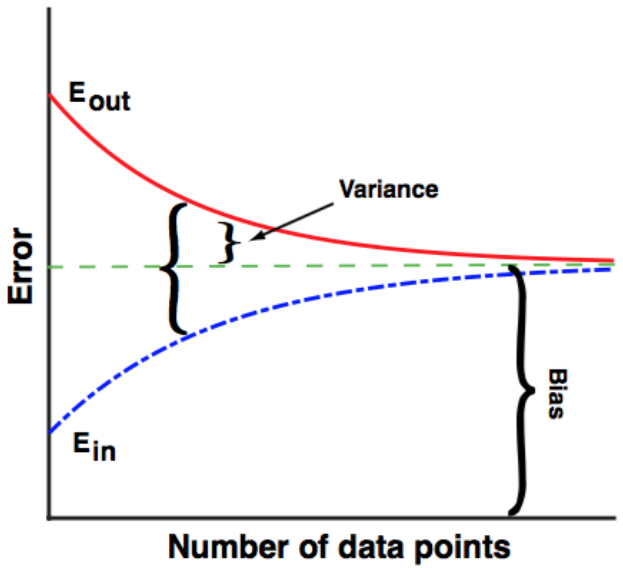


[1]

Typical behavior:

- E_{in} increases with N towards the model bias.
- E_{out} decreases with increasing N as more cases are observed and thus covered by the model.
- The **generalization gap** $E_{\text{out}} - E_{\text{in}}$ (due to overfitting) decreases with increasing N.

[1] From Mehta et al, arXiv:1803.08823v1

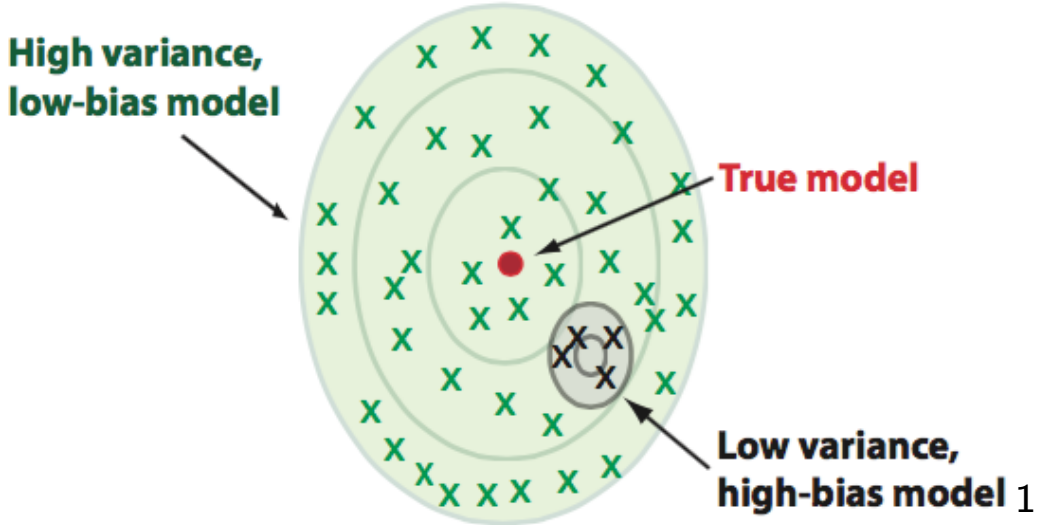


[1]

Insights:

- It is not sufficient to minimize E_{in}, as E_{out} may be large. $\rightarrow$ *regularization*.
- As the true bias is not practically available, one minimizes E_{out}.

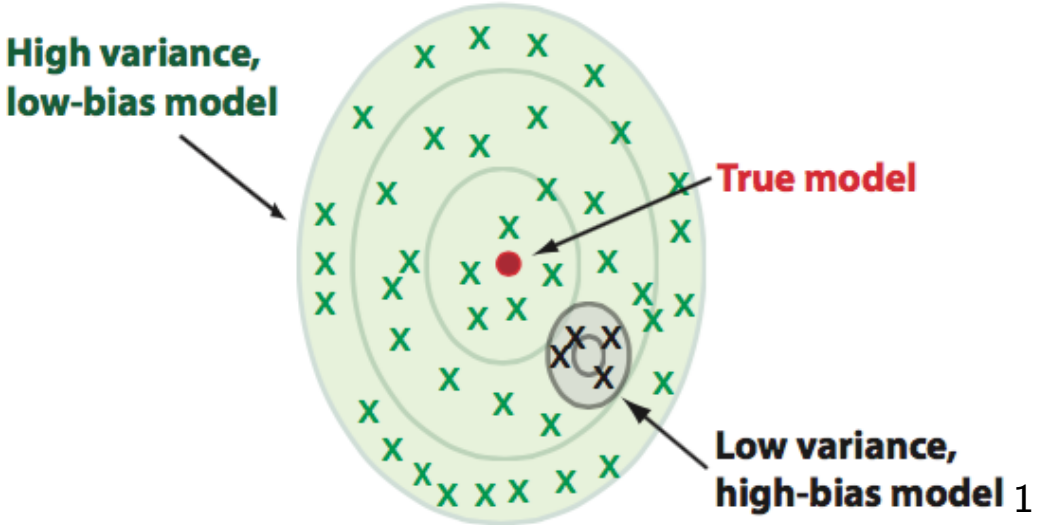
[1] From Mehta et al, arXiv:1803.08823v1



[1]

- **Bias-variance tradeoff**: For fixed N, the more/less expressive the model, the larger/smaller the fluctuations, respectively.
- To minimize E_{out}, it is sometimes better to use a more-biased model with small variance than a less-biased model with large variance.
- Asymptotically, i.e. with increasing training set size N, complex models will perform better than simpler models as they have reduced bias.
- Optimal model selection depends on the amount of training data N.

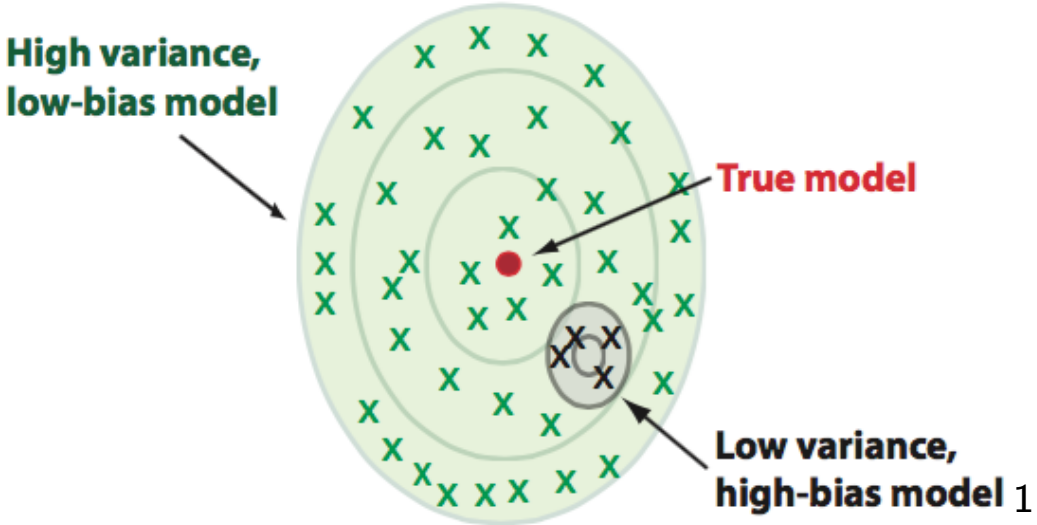
[1] From Mehta et al, arXiv:1803.08823v1



[1]

- **Bias-variance tradeoff**: For fixed N, the more/less expressive the model, the larger/smaller the fluctuations, respectively.
- To minimize E_{out}, it is sometimes better to use a more-biased model with small variance than a less-biased model with large variance.
- Asymptotically, i.e. with increasing training set size N, complex models will perform better than simpler models as they have reduced bias.
- Optimal model selection depends on the amount of training data N.

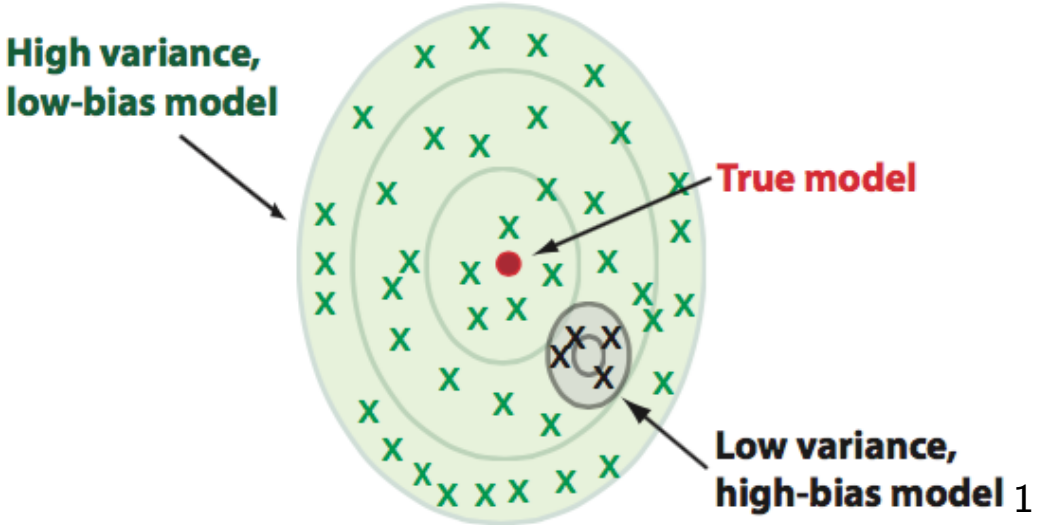
[1] From Mehta et al, arXiv:1803.08823v1



[1]

- **Bias-variance tradeoff**: For fixed N, the more/less expressive the model, the larger/smaller the fluctuations, respectively.
- To minimize E_{out}, it is sometimes better to use a more-biased model with small variance than a less-biased model with large variance.
- Asymptotically, i.e. with increasing training set size N, complex models will perform better than simpler models as they have reduced bias.
- Optimal model selection depends on the amount of training data N.

[1] From Mehta et al, arXiv:1803.08823v1



[1]

- **Bias-variance tradeoff**: For fixed N, the more/less expressive the model, the larger/smaller the fluctuations, respectively.
- To minimize E_{out}, it is sometimes better to use a more-biased model with small variance than a less-biased model with large variance.
- Asymptotically, i.e. with increasing training set size N, complex models will perform better than simpler models as they have reduced bias.
- Optimal model selection depends on the amount of training data N.

[1] From Mehta et al, arXiv:1803.08823v1

Bias-Variance decomposition

Task: for a given estimator, e.g. LLS, model the behavior of the out-of-sample MSE without knowing the true function f:

$$E_{\text{out}} = C(\mathbf{X}^{\text{val}}, \mathbf{y}^{\text{val}}, \hat{\mathbf{y}}^{\text{val}}) = \left\| \mathbf{y}^{\text{val}} - \hat{\mathbf{y}}^{\text{val}} \right\|_2^2,$$

where $\mathbf{X}^{\text{val}} = (\mathbf{x}_1^{\text{val}}, ..., \mathbf{x}_N^{\text{val}})^\top$ are the features of the validation set, $\mathbf{y}^{\text{val}}$ are the corresponding observations and $\hat{\mathbf{y}}^{\text{val}}$ are the predictions of the estimator.

Idea: Compute **expected** E_{out} of given estimator $\hat{y}$ on all data $y = f(\mathbf{x}) + \varepsilon$ drawn from the true model $f(\mathbf{x})$ with following approach:

1. Fix observation points $\mathbf{x}_i$
2. Repeat:
   1. Run experiment, observe training data $(\mathbf{X}, \mathbf{y}^{\text{train}}) = (\mathbf{x}_i, y_i^{\text{train}})_{i=1,...,N}$ and train the estimator $\hat{y}(\mathbf{x})$
   2. Repeat experiment, observe validation data $(\mathbf{X}, \mathbf{y}^{\text{val}}) = (\mathbf{x}_i, y_i^{\text{val}})_{i=1,...,N}$
   3. Compute expectation $\mathbb{E}$ over observations $\mathbf{y}^{\text{train}}$ and $\mathbf{y}^{\text{val}}$ by averaging over noise realizations.

$$\mathbb{E}\left[C(\mathbf{X}, \mathbf{y}^{\text{val}}, \hat{\mathbf{y}}^{\text{val}})\right] = \mathbb{E}\left[\sum_{i=1}^{N} \left(y_i^{\text{val}}(\mathbf{x}_i) - \hat{y}(\mathbf{x}_i)\right)^2\right]$$

Bias-Variance decomposition

Task: for a given estimator, e.g. LLS, model the behavior of the out-of-sample MSE without knowing the true function f:

$$E_{\text{out}} = C(\mathbf{X}^{\text{val}}, \mathbf{y}^{\text{val}}, \hat{\mathbf{y}}^{\text{val}}) = \left\| \mathbf{y}^{\text{val}} - \hat{\mathbf{y}}^{\text{val}} \right\|_2^2,$$

where $\mathbf{X}^{\text{val}} = (\mathbf{x}_1^{\text{val}}, \ldots, \mathbf{x}_N^{\text{val}})^\top$ are the features of the validation set, $\mathbf{y}^{\text{val}}$ are the corresponding observations and $\hat{\mathbf{y}}^{\text{val}}$ are the predictions of the estimator.

Idea: Compute **expected** E_{out} of given estimator $\hat{y}$ on all data $y = f(\mathbf{x}) + \varepsilon$ drawn from the true model $f(\mathbf{x})$ with following approach:

1. Fix observation points $\mathbf{x}_i$
2. Repeat:
   1. Run experiment, observe training data $(\mathbf{X}, \mathbf{y}^{\text{train}}) = (\mathbf{x}_i, y_i^{\text{train}})_{i=1,\ldots N}$ and train the estimator $\hat{y}(\mathbf{x})$.
   2. Repeat experiment, observe validation data $(\mathbf{X}, \mathbf{y}^{\text{val}}) = (\mathbf{x}_i, y_i^{\text{val}})_{i=1,\ldots N}$.
3. Compute expectation $\mathbb{E}$ over observations $\mathbf{y}^{\text{train}}$ and $\mathbf{y}^{\text{val}}$ by averaging over noise realizations.

$$\mathbb{E}\left[C(\mathbf{X}, \mathbf{y}^{\text{val}}, \hat{\mathbf{y}}^{\text{val}})\right] = \mathbb{E}\left[\sum_{i=1}^{N} \left(y_i^{\text{val}}(\mathbf{x}_i) - \hat{y}(\mathbf{x}_i)\right)^2\right]$$

Bias-Variance decomposition

Task: for a given estimator, e.g. LLS, model the behavior of the out-of-sample MSE without knowing the true function f:

$$E_{\text{out}} = C(\mathbf{X}^{\text{val}}, \mathbf{y}^{\text{val}}, \hat{\mathbf{y}}^{\text{val}}) = \left\| \mathbf{y}^{\text{val}} - \hat{\mathbf{y}}^{\text{val}} \right\|_2^2,$$

where $\mathbf{X}^{\text{val}} = (\mathbf{x}_1^{\text{val}}, \ldots, \mathbf{x}_N^{\text{val}})^\top$ are the features of the validation set, $\mathbf{y}^{\text{val}}$ are the corresponding observations and $\hat{\mathbf{y}}^{\text{val}}$ are the predictions of the estimator.

Idea: Compute **expected** E_{out} of given estimator $\hat{y}$ on all data $y = f(\mathbf{x}) + \varepsilon$ drawn from the true model $f(\mathbf{x})$ with following approach:

1. Fix observation points $\mathbf{x}_i$
2. Repeat:
   1. Run experiment, observe training data $(\mathbf{X}, \mathbf{y}^{\text{train}}) = (\mathbf{x}_i, y_i^{\text{train}})_{i=1,\ldots N}$ and train the estimator $\hat{y}(\mathbf{x})$.
   2. Repeat experiment, observe validation data $(\mathbf{X}, \mathbf{y}^{\text{val}}) = (\mathbf{x}_i, y_i^{\text{val}})_{i=1,\ldots N}$.
3. Compute expectation $\mathbb{E}$ over observations $\mathbf{y}^{\text{train}}$ and $\mathbf{y}^{\text{val}}$ by averaging over noise realizations.

$$\mathbb{E}\left[C(\mathbf{X}, \mathbf{y}^{\text{val}}, \hat{\mathbf{y}}^{\text{val}})\right] = \mathbb{E}\left[\sum_{i=1}^{N} \left(y_i^{\text{val}}(\mathbf{x}_i) - \hat{y}(\mathbf{x}_i)\right)^2\right]$$

Bias-Variance decomposition

Expected out-of-sample error, i.e. the **expected loss** of our model can be decomposed as:

$$E_{\text{out}} = \mathbb{E}\left[\sum_{i=1}^{N}\left(y_i^{\text{val}}(\mathbf{x}_i) - \hat{y}(\mathbf{x}_i)\right)^2\right] = \text{Bias}^2 + \text{Var} + \text{Noise}.$$

with $\bar{y}_i = \mathbb{E}[\hat{y}(\mathbf{x}_i)]$ we have:

$$\text{Noise} = \sum_{i=1}^{N}\mathbb{E}\left[\left(y_i^{\text{val}}(\mathbf{x}_i) - f(\mathbf{x}_i)\right)^2\right] = \sum_{i=1}^{N}\mathbb{E}\left[\varepsilon^2\right] = \sigma_\varepsilon^2$$

$$\text{Var} = \sum_{i=1}^{N}\mathbb{E}\left[\left(\hat{y}(\mathbf{x}_i) - \bar{y}_i\right)^2\right]$$

$$\text{Bias}^2 = \sum_{i=1}^{N}\left(f(\mathbf{x}_i) - \bar{y}_i\right)^2$$

Bias-Variance decomposition

Expected out-of-sample error, i.e. the **expected loss** of our model can be decomposed as:

$$E_{\text{out}} = \mathbb{E}\left[\sum_{i=1}^{N}\left(y_i^{\text{val}}(\mathbf{x}_i) - \hat{y}(\mathbf{x}_i)\right)^2\right] = \text{Bias}^2 + \text{Var} + \text{Noise}.$$

with $\bar{y}_i = \mathbb{E}[\hat{y}(\mathbf{x}_i)]$ we have:

$$\text{Noise} = \sum_{i=1}^{N}\mathbb{E}\left[\left(y_i^{\text{val}}(\mathbf{x}_i) - f(\mathbf{x}_i)\right)^2\right] = \sum_{i=1}^{N}\mathbb{E}\left[\varepsilon^2\right] = \sigma_\varepsilon^2$$

$$\text{Var} = \sum_{i=1}^{N}\mathbb{E}\left[\left(\hat{y}(\mathbf{x}_i) - \bar{y}_i\right)^2\right]$$

$$\text{Bias}^2 = \sum_{i=1}^{N}\left(f(\mathbf{x}_i) - \bar{y}_i\right)^2$$

Bias-Variance decomposition

Combining these expressions, we see that the expected out-of-sample error, i.e. the **expected loss** of our model can be decomposed as:

$$E_{\text{out}} = \mathbb{E}\left[\sum_{i=1}^{N}\left(y_i^{\text{val}}(\mathbf{x}_i) - \hat{y}(\mathbf{x}_i)\right)^2\right] = \text{Bias}^2 + \text{Var} + \text{Noise}.$$

- The **optimal model** minimizes the expected loss by **balancing bias and variance**.
- A model is **underfitting** the data if bias is too high.
- A model is **overfitting** the data if variance is too high.
- Since data is often limited, a simple model with a finite bias (i.e. an asymptotic error) may be preferable to a complex model with a high variance.
- Optimal choice depends on the amount of data available. The more data, the more complex models are optimal.

Bias-Variance decomposition

Combining these expressions, we see that the expected out-of-sample error, i.e. the **expected loss** of our model can be decomposed as:

$$E_{\text{out}} = \mathbb{E}\left[\sum_{i=1}^{N}\left(y_i^{\text{val}}(\mathbf{x}_i) - \hat{y}(\mathbf{x}_i)\right)^2\right] = \text{Bias}^2 + \text{Var} + \text{Noise}.$$

- The **optimal model** minimizes the expected loss by **balancing bias and variance**.
- A model is **underfitting** the data if bias is too high.
- A model is **overfitting** the data if variance is too high.
- Since data is often limited, a simple model with a finite bias (i.e. an asymptotic error) may be preferable to a complex model with a high variance.
- Optimal choice depends on the amount of data available. The more data, the more complex models are optimal.

Bias-Variance decomposition

Combining these expressions, we see that the expected out-of-sample error, i.e. the **expected loss** of our model can be decomposed as:

$$E_{\text{out}} = \mathbb{E}\left[\sum_{i=1}^{N}\left(y_i^{\text{val}}(\mathbf{x}_i) - \hat{y}(\mathbf{x}_i)\right)^2\right] = \text{Bias}^2 + \text{Var} + \text{Noise}.$$

- The **optimal model** minimizes the expected loss by **balancing bias and variance**.
- A model is **underfitting** the data if bias is too high.
- A model is **overfitting** the data if variance is too high.
- Since data is often limited, a simple model with a finite bias (i.e. an asymptotic error) may be preferable to a complex model with a high variance.
- Optimal choice depends on the amount of data available. The more data, the more complex models are optimal.

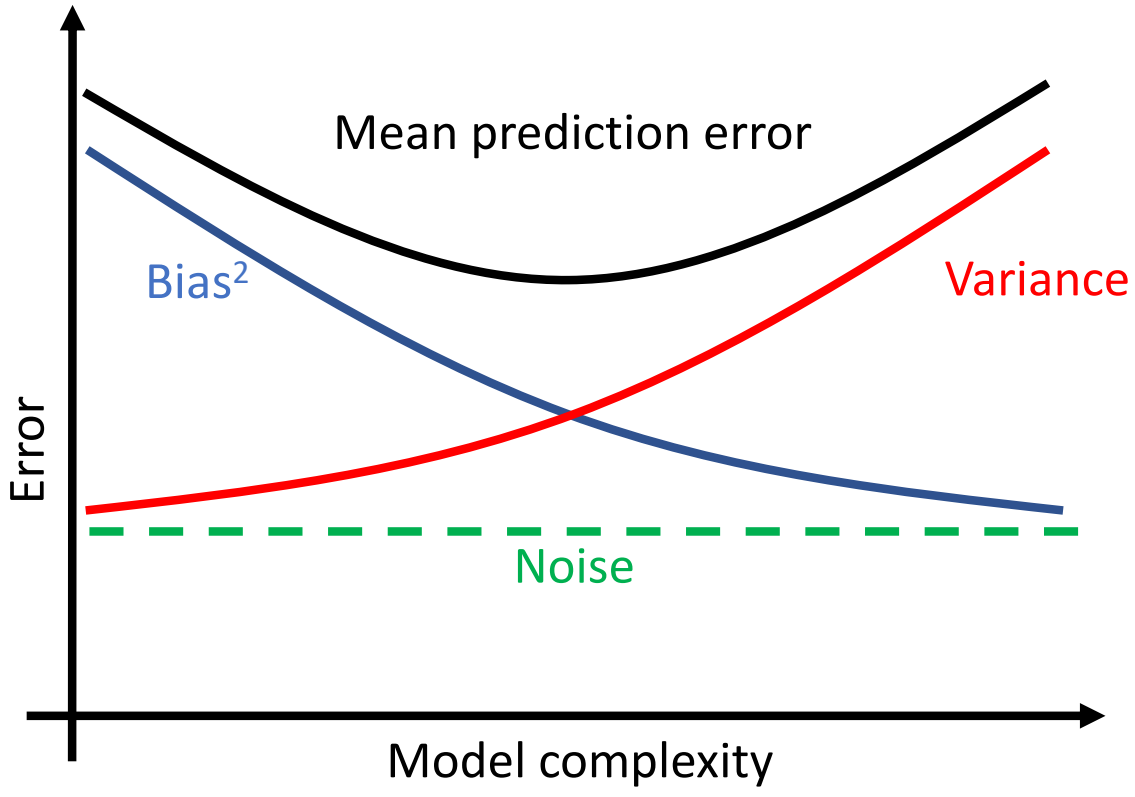
Bias-Variance decomposition

Combining these expressions, we see that the expected out-of-sample error, i.e. the **expected loss** of our model can be decomposed as:

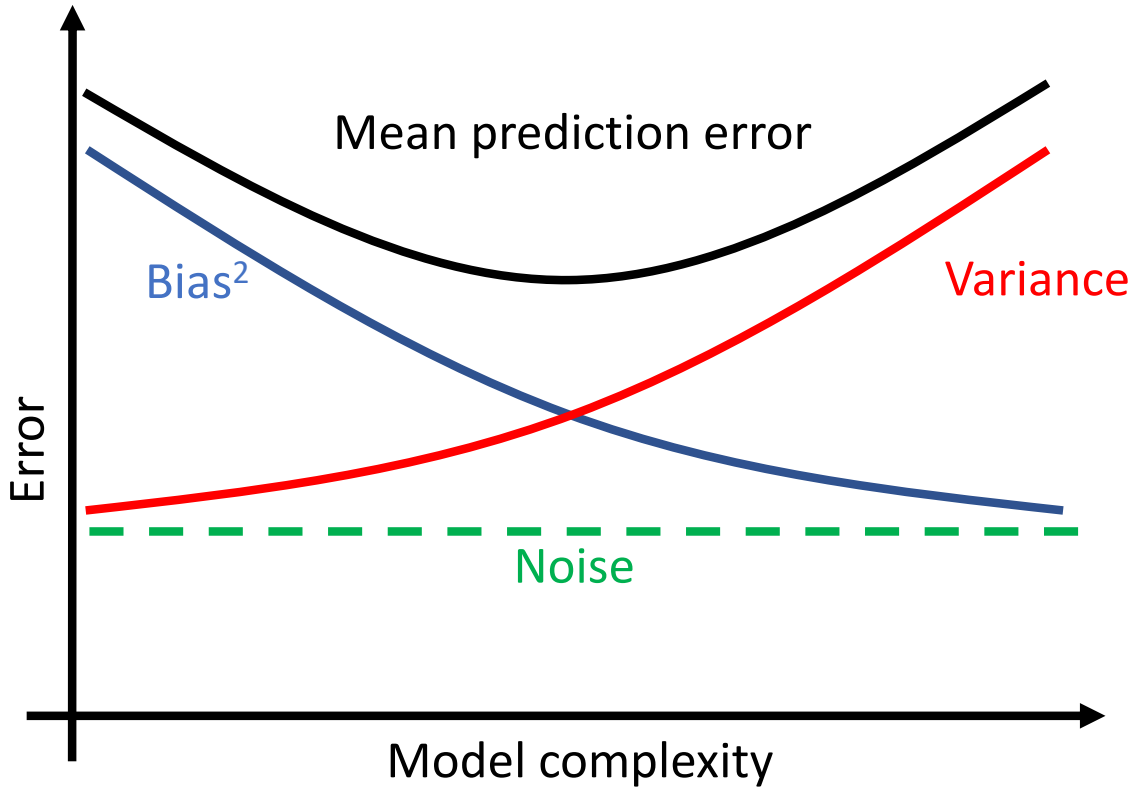
$$E_{\text{out}} = \mathbb{E}\left[\sum_{i=1}^{N}\left(y_i^{\text{val}}(\mathbf{x}_i) - \hat{y}(\mathbf{x}_i)\right)^2\right] = \text{Bias}^2 + \text{Var} + \text{Noise}.$$

- The **optimal model** minimizes the expected loss by **balancing bias and variance**.
- A model is **underfitting** the data if bias is too high.
- A model is **overfitting** the data if variance is too high.
- Since data is often limited, a simple model with a finite bias (i.e. an asymptotic error) may be preferable to a complex model with a high variance.
- Optimal choice depends on the amount of data available. The more data, the more complex models are optimal.

- Model complexity is a property of the function class $\mathscr{H}$. For example, model complexity increases with the number of free parameters (e.g. higher-order polynomials are more complex than the linear model).
- Behavior for fixed N:

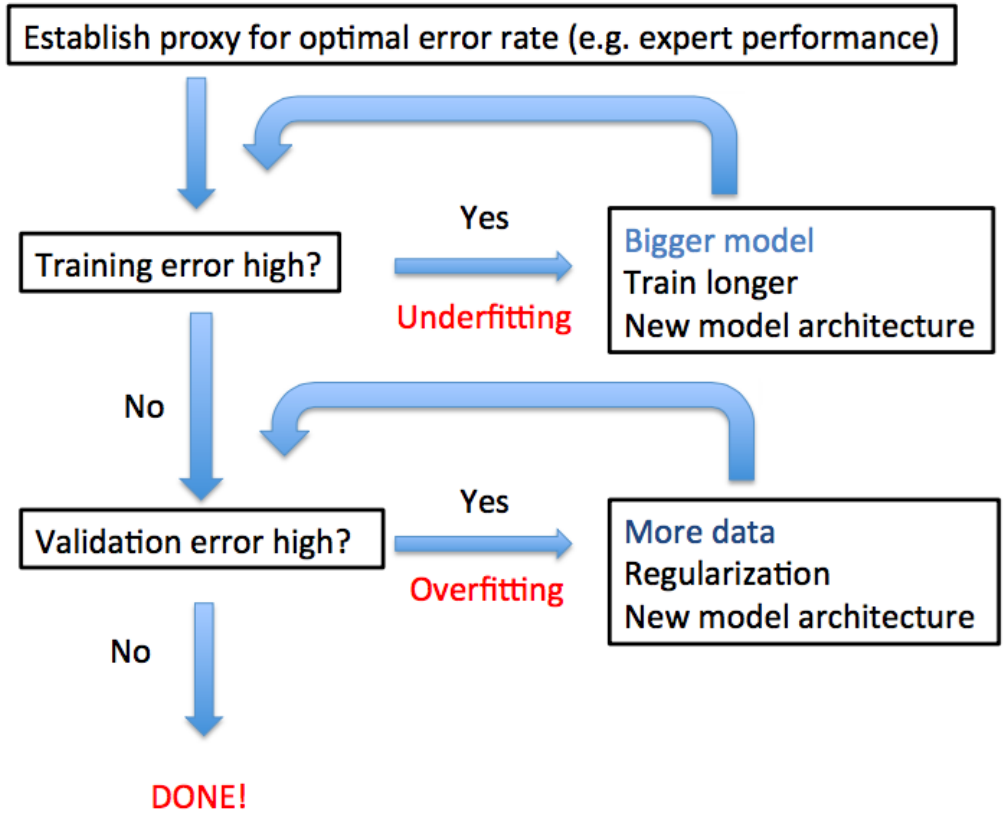


- Model complexity is a property of the function class $\mathscr{H}$. For example, model complexity increases with the number of free parameters (e.g. higher-order polynomials are more complex than the linear model).
- Behavior for fixed N:



Practical workflow

For complex estimators (e.g. neural networks), exhaustive hyperparameter search is unfeasible.
Typical approach:



Regularization

Regularized LLS: add penalty term on $\mathbf{w}$ with suitable norm:

$$\min \|\mathbf{y} - \mathbf{X}\mathbf{w}\|_2^2 + \lambda \|\mathbf{w}\| .$$

Purpose:

- **Statistical**: reduce expressiveness of model by reducing fluctuations of $\mathbf{w}$. Allows to control the bias-variance tradeoff via λ.
- **Numerical**: regularized solutions often numerically better behaved.
- **Structural**: e.g., induce sparsity in solution.

Regularization method depends on penalty type:

Regularization type	Penalty term	Prior	Solution methods
Tikhonov regularization Ridge regression	$\|\mathbf{w}\|_2^2$	Normal	Closed form
Lasso regression	$\|\mathbf{w}\|_1$	Laplace	Proximal gradient descent
l_0 regularization	$\|\mathbf{w}\|_0$	-	Forward selection, Backward elimination
Elastic nets	$(1-\alpha)\|\mathbf{w}\|_1 +$ $\alpha\|\mathbf{w}\|_2$	-	Proximal gradient descent

Regularization

Regularized LLS: add penalty term on $\mathbf{w}$ with suitable norm:

$$\min \|\mathbf{y} - \mathbf{X}\mathbf{w}\|_2^2 + \lambda \|\mathbf{w}\| .$$

Purpose:

- **Statistical**: reduce expressiveness of model by reducing fluctuations of $\mathbf{w}$. Allows to control the bias-variance tradeoff via λ.
- **Numerical**: regularized solutions often numerically better behaved.
- **Structural**: e.g., induce sparsity in solution.

Regularization method depends on penalty type:

Regularization type	Penalty term	Prior	Solution methods
Tikhonov regularization Ridge regression	$\|\mathbf{w}\|_2^2$	Normal	Closed form
Lasso regression	$\|\mathbf{w}\|_1$	Laplace	Proximal gradient descent
l_0 regularization	$\|\mathbf{w}\|_0$	-	Forward selection, Backward elimination
Elastic nets	$(1-\alpha)\|\mathbf{w}\|_1 +$ $\alpha\|\mathbf{w}\|_2$	-	Proximal gradient descent

Regularization

Regularized LLS: add penalty term on $\mathbf{w}$ with suitable norm:

$$\min \|\mathbf{y} - \mathbf{X}\mathbf{w}\|_2^2 + \lambda \|\mathbf{w}\| .$$

Purpose:

- **Statistical**: reduce expressiveness of model by reducing fluctuations of $\mathbf{w}$. Allows to control the bias-variance tradeoff via λ.
- **Numerical**: regularized solutions often numerically better behaved.
- **Structural**: e.g., induce sparsity in solution.

Regularization method depends on penalty type:

Regularization type	Penalty term	Prior	Solution methods
Tikhonov regularization Ridge regression	$\|\mathbf{w}\|_2^2$	Normal	Closed form
Lasso regression	$\|\mathbf{w}\|_1$	Laplace	Proximal gradient descent
l_0 regularization	$\|\mathbf{w}\|_0$	-	Forward selection, Backward elimination
Elastic nets	$(1-\alpha)\|\mathbf{w}\|_1 +$ $\alpha \|\mathbf{w}\|_2$	-	Proximal gradient descent

L2 (Ridge) Regularization

We would like to work in high-dimensional feature spaces

$$\mathbf{r}_i \to \mathbf{x}_i = (\phi_1(\mathbf{r}_i), ..., \phi_n(\mathbf{r}_i))^\top .$$

However, this leads to danger of overfitting. To avoid overfitting, we penalize the norm of the solution:

$$\min \|\mathbf{y} - \mathbf{X}\mathbf{w}\|_2^2 + \lambda \|\mathbf{w}\|_2^2,$$

where λ is a hyperparameter.
Taking derivatives and setting them to zero yields the solution:

$$\begin{aligned}\mathbf{w} &= \left(\lambda \mathbf{I} + \mathbf{X}^\top \mathbf{X}\right)^{-1} \mathbf{X}^\top \mathbf{y} \\ &= \tilde{\mathbf{C}}_{XX}^{-1} \mathbf{C}_{XY}\end{aligned}$$

This is equal to the direct solution of the normal equations, only that we use the so-called shrinkage estimator for the covariance matrix:

$$\tilde{\mathbf{C}}_{XX} = \lambda \mathbf{I} + \mathbf{X}^\top \mathbf{X}$$

L2 (Ridge) Regularization

We would like to work in high-dimensional feature spaces

$$\mathbf{r}_i \to \mathbf{x}_i = (\phi_1(\mathbf{r}_i), ..., \phi_n(\mathbf{r}_i))^\top .$$

However, this leads to danger of overfitting. To avoid overfitting, we penalize the norm of the solution:

$$\min \|\mathbf{y} - \mathbf{X}\mathbf{w}\|_2^2 + \lambda \|\mathbf{w}\|_2^2 ,$$

where λ is a hyperparameter.

Taking derivatives and setting them to zero yields the solution:

$$\begin{aligned} \mathbf{w} &= \left(\lambda \mathbf{I} + \mathbf{X}^\top \mathbf{X}\right)^{-1} \mathbf{X}^\top \mathbf{y} \\ &= \tilde{\mathbf{C}}_{XX}^{-1} \mathbf{C}_{XY} \end{aligned}$$

This is equal to the direct solution of the normal equations, only that we use the so-called shrinkage estimator for the covariance matrix:

$$\tilde{\mathbf{C}}_{XX} = \lambda \mathbf{I} + \mathbf{X}^\top \mathbf{X}$$

L2 (Ridge) Regularization

We would like to work in high-dimensional feature spaces

$$\mathbf{r}_i \to \mathbf{x}_i = (\phi_1(\mathbf{r}_i), ..., \phi_n(\mathbf{r}_i))^\top .$$

However, this leads to danger of overfitting. To avoid overfitting, we penalize the norm of the solution:

$$\min \|\mathbf{y} - \mathbf{X}\mathbf{w}\|_2^2 + \lambda \|\mathbf{w}\|_2^2 ,$$

where λ is a hyperparameter.
Taking derivatives and setting them to zero yields the solution:

$$\begin{aligned} \mathbf{w} &= \left(\lambda \mathbf{I} + \mathbf{X}^\top \mathbf{X}\right)^{-1} \mathbf{X}^\top \mathbf{y} \\ &= \tilde{\mathbf{C}}_{XX}^{-1} \mathbf{C}_{XY} \end{aligned}$$

This is equal to the direct solution of the normal equations, only that we use the so-called shrinkage estimator for the covariance matrix:

$$\tilde{\mathbf{C}}_{XX} = \lambda \mathbf{I} + \mathbf{X}^\top \mathbf{X}$$

L2 (Ridge) Regularization

We would like to work in high-dimensional feature spaces

$$\mathbf{r}_i \to \mathbf{x}_i = (\phi_1(\mathbf{r}_i), ..., \phi_n(\mathbf{r}_i))^\top .$$

However, this leads to danger of overfitting. To avoid overfitting, we penalize the norm of the solution:

$$\min \|\mathbf{y} - \mathbf{X}\mathbf{w}\|_2^2 + \lambda \|\mathbf{w}\|_2^2,$$

where λ is a hyperparameter.
Taking derivatives and setting them to zero yields the solution:

$$\begin{aligned} \mathbf{w} &= \left(\lambda \mathbf{I} + \mathbf{X}^\top \mathbf{X}\right)^{-1} \mathbf{X}^\top \mathbf{y} \\ &= \tilde{\mathbf{C}}_{XX}^{-1} \mathbf{C}_{XY} \end{aligned}$$

This is equal to the direct solution of the normal equations, only that we use the so-called shrinkage estimator for the covariance matrix:

$$\tilde{\mathbf{C}}_{XX} = \lambda \mathbf{I} + \mathbf{X}^\top \mathbf{X}$$

Sparsity-inducing Regularization

- **L0 regularization**

$$\min \|\mathbf{y} - \mathbf{X}\mathbf{w}\|_2^2 + \lambda \|\mathbf{w}\|_0 ,$$

Most extreme way to enforce sparsity. Magnitude of the coefficients of $\mathbf{w}$ does not matter, we only want to minimize the number of non-zero entries. This regularization function is not commonly used in practice, as it is very difficult to solve.

- **L1 regularization**, e.g. using the least absolute selection and shrinkage (LASSO) method.

$$\min \|\mathbf{y} - \mathbf{X}\mathbf{w}\|_2^2 + \lambda \|\mathbf{w}\|_1 ,$$

- **Elastic net**

$$\min \|\mathbf{y} - \mathbf{X}\mathbf{w}\|_2^2 + \lambda \left[(1-\alpha) \|\mathbf{w}\|_1 + \alpha \|\mathbf{w}\|_2^2 \right],$$

Where α switches between the two extremes $\alpha = 0$ (L1 regularization) and $\alpha = 1$ (Ridge regression).

Sparsity-inducing Regularization

- **L0 regularization**

$$\min \|\mathbf{y} - \mathbf{X}\mathbf{w}\|_2^2 + \lambda \|\mathbf{w}\|_0 ,$$

  Most extreme way to enforce sparsity. Magnitude of the coefficients of $\mathbf{w}$ does not matter, we only want to minimize the number of non-zero entries. This regularization function is not commonly used in practice, as it is very difficult to solve.

- **L1 regularization**, e.g. using the least absolute selection and shrinkage (LASSO) method.

$$\min \|\mathbf{y} - \mathbf{X}\mathbf{w}\|_2^2 + \lambda \|\mathbf{w}\|_1 ,$$

- **Elastic net**

$$\min \|\mathbf{y} - \mathbf{X}\mathbf{w}\|_2^2 + \lambda \left[(1-\alpha) \|\mathbf{w}\|_1 + \alpha \|\mathbf{w}\|_2^2 \right],$$

  Where α switches between the two extremes $\alpha = 0$ (L1 regularization) and $\alpha = 1$ (Ridge regression).

Sparsity-inducing Regularization

- **L0 regularization**

$$\min \|\mathbf{y} - \mathbf{X}\mathbf{w}\|_2^2 + \lambda \|\mathbf{w}\|_0 ,$$

  Most extreme way to enforce sparsity. Magnitude of the coefficients of $\mathbf{w}$ does not matter, we only want to minimize the number of non-zero entries. This regularization function is not commonly used in practice, as it is very difficult to solve.

- **L1 regularization**, e.g. using the least absolute selection and shrinkage (LASSO) method.

$$\min \|\mathbf{y} - \mathbf{X}\mathbf{w}\|_2^2 + \lambda \|\mathbf{w}\|_1 ,$$

- **Elastic net**

$$\min \|\mathbf{y} - \mathbf{X}\mathbf{w}\|_2^2 + \lambda \left[(1-\alpha) \|\mathbf{w}\|_1 + \alpha \|\mathbf{w}\|_2^2 \right],$$

  Where α switches between the two extremes $\alpha = 0$ (L1 regularization) and $\alpha = 1$ (Ridge regression).

Kernel Regression

- Replace each data point with feature vector: $\mathbf{x}_i \to \phi(\mathbf{x}_i)$.
- $\dim(\phi)$ can be much higher than $\dim(\mathbf{x}_i)$ (even ∞).
  - $\to$ How do we compute or invert $\mathbf{X}^\top\mathbf{X}$?
  - $\to$ How do we avoid overfitting?
- **Kernel trick**: allows us to work with either $\mathbf{X}^\top\mathbf{X}$ or $\mathbf{X}\mathbf{X}^\top$, whichever is more convenient (lower-dimensional).
- **Kernel ridge regression** can be written as:

$$\mathbf{w} = \left(\lambda\mathbf{I} + \mathbf{X}^\top\mathbf{X}\right)^{-1}\mathbf{X}^\top\mathbf{y} = \mathbf{X}^\top\left(\mathbf{X}\mathbf{X}^\top + \lambda\mathbf{I}\right)^{-1}\mathbf{y},$$

or, equivalently:

$$\mathbf{w} = \sum_i \alpha_i \mathbf{x}_i$$

$$\alpha = \left(\mathbf{X}\mathbf{X}^\top + \lambda\mathbf{I}\right)^{-1}\mathbf{y}$$

Kernel Regression

- Replace each data point with feature vector: $\mathbf{x}_i \rightarrow \phi(\mathbf{x}_i)$.
- $\dim(\phi)$ can be much higher than $\dim(\mathbf{x}_i)$ (even ∞).
  - $\rightarrow$ How do we compute or invert $\mathbf{X}^\top\mathbf{X}$?
  - $\rightarrow$ How do we avoid overfitting?
- **Kernel trick**: allows us to work with either $\mathbf{X}^\top\mathbf{X}$ or $\mathbf{X}\mathbf{X}^\top$, whichever is more convenient (lower-dimensional).
- **Kernel ridge regression** can be written as:

$$\mathbf{w} = \left(\lambda\mathbf{I} + \mathbf{X}^\top\mathbf{X}\right)^{-1}\mathbf{X}^\top\mathbf{y} = \mathbf{X}^\top\left(\mathbf{X}\mathbf{X}^\top + \lambda\mathbf{I}\right)^{-1}\mathbf{y},$$

or, equivalently:

$$\mathbf{w} = \sum_i \alpha_i \mathbf{x}_i$$

$$\alpha = \left(\mathbf{X}\mathbf{X}^\top + \lambda\mathbf{I}\right)^{-1}\mathbf{y}$$

Kernel Regression

- Replace each data point with feature vector: $\mathbf{x}_i \rightarrow \phi(\mathbf{x}_i)$.
- $\dim(\phi)$ can be much higher than $\dim(\mathbf{x}_i)$ (even ∞).
  - $\rightarrow$ How do we compute or invert $\mathbf{X}^\top\mathbf{X}$?
  - $\rightarrow$ How do we avoid overfitting?
- **Kernel trick**: allows us to work with either $\mathbf{X}^\top\mathbf{X}$ or $\mathbf{X}\mathbf{X}^\top$, whichever is more convenient (lower-dimensional).
- **Kernel ridge regression** can be written as:

$$\mathbf{w} = \left(\lambda\mathbf{I} + \mathbf{X}^\top\mathbf{X}\right)^{-1}\mathbf{X}^\top\mathbf{y} = \mathbf{X}^\top\left(\mathbf{X}\mathbf{X}^\top + \lambda\mathbf{I}\right)^{-1}\mathbf{y},$$

or, equivalently:

$$\mathbf{w} = \sum_i \alpha_i \mathbf{x}_i$$

$$\alpha = \left(\mathbf{X}\mathbf{X}^\top + \lambda\mathbf{I}\right)^{-1}\mathbf{y}$$

Kernel Regression

- Replace each data point with feature vector: $\mathbf{x}_i \rightarrow \phi(\mathbf{x}_i)$.
- $\dim(\phi)$ can be much higher than $\dim(\mathbf{x}_i)$ (even ∞).
  - $\rightarrow$ How do we compute or invert $\mathbf{X}^\top\mathbf{X}$?
  - $\rightarrow$ How do we avoid overfitting?
- **Kernel trick**: allows us to work with either $\mathbf{X}^\top\mathbf{X}$ or $\mathbf{X}\mathbf{X}^\top$, whichever is more convenient (lower-dimensional).
- **Kernel ridge regression** can be written as:

$$\mathbf{w} = \left(\lambda\mathbf{I} + \mathbf{X}^\top\mathbf{X}\right)^{-1}\mathbf{X}^\top\mathbf{y} = \mathbf{X}^\top\left(\mathbf{X}\mathbf{X}^\top + \lambda\mathbf{I}\right)^{-1}\mathbf{y},$$

or, equivalently:

$$\mathbf{w} = \sum_i \alpha_i \mathbf{x}_i$$

$$\alpha = \left(\mathbf{X}\mathbf{X}^\top + \lambda\mathbf{I}\right)^{-1}\mathbf{y}$$

Kernel Regression

- **Kernel ridge regression**

$$\mathbf{w} = \sum_i \alpha_i \mathbf{x}_i$$

$$\alpha = \left(\mathbf{X}\mathbf{X}^\top + \lambda \mathbf{I}\right)^{-1} \mathbf{y}$$

  - Solution $\mathbf{w}$ lies in $\text{span}\{\mathbf{x}_i\}_{i=1\ldots N}$, even if $\dim(\phi) = \infty$.

- Predicted value $\tilde{y}$ for test point $\tilde{\mathbf{x}}$:

$$\tilde{y} = \mathbf{y}(\mathbf{X}\mathbf{X}^\top + \lambda \mathbf{I})^{-1}\mathbf{X}\tilde{\mathbf{x}}$$

- **Key ideas**:
  - Never perform the feature transformation $\mathbf{X}_i = (\phi_1(\mathbf{x}_i), \ldots, \phi_1(\mathbf{x}_i))$ explicitly. Instead, define kernel function that models scalar product between feature vectors:

$$k(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{X}_i^\top \mathbf{X}_j = \langle \mathbf{X}_i, \mathbf{X}_j \rangle$$

  - Use the kernel matrix $\mathbf{K} = \mathbf{X}\mathbf{X}^\top \in \mathbb{R}^{N \times N}$, $K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$

Kernel Regression

- **Kernel ridge regression**

$$\mathbf{w} = \sum_i \alpha_i \mathbf{x}_i$$

$$\alpha = \left(\mathbf{X}\mathbf{X}^\top + \lambda \mathbf{I}\right)^{-1} \mathbf{y}$$

  - Solution $\mathbf{w}$ lies in $\text{span}\{\mathbf{x}_i\}_{i=1\ldots N}$, even if $\dim(\phi) = \infty$.
- Predicted value $\tilde{y}$ for test point $\tilde{\mathbf{x}}$:

$$\tilde{y} = \mathbf{y}(\mathbf{X}\mathbf{X}^\top + \lambda \mathbf{I})^{-1}\mathbf{X}\tilde{\mathbf{x}}$$

- **Key ideas**:
  - Never perform the feature transformation $\mathbf{X}_i = (\phi_1(\mathbf{x}_i), \ldots, \phi_1(\mathbf{x}_i))$ explicitly. Instead, define kernel function that models scalar product between feature vectors:

$$k(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{X}_i^\top \mathbf{X}_j = \langle \mathbf{X}_i, \mathbf{X}_j \rangle$$

  - Use the kernel matrix $\mathbf{K} = \mathbf{X}\mathbf{X}^\top \in \mathbb{R}^{N \times N}$, $K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$

Kernel Regression

- **Kernel ridge regression**

$$\mathbf{w} = \sum_i \alpha_i \mathbf{x}_i$$

$$\alpha = \left(\mathbf{X}\mathbf{X}^\top + \lambda \mathbf{I}\right)^{-1} \mathbf{y}$$

  - Solution $\mathbf{w}$ lies in $\text{span}\{\mathbf{x}_i\}_{i=1...N}$, even if $\dim(\phi) = \infty$.
- Predicted value $\tilde{y}$ for test point $\tilde{\mathbf{x}}$:

$$\tilde{y} = \mathbf{y}(\mathbf{X}\mathbf{X}^\top + \lambda \mathbf{I})^{-1}\mathbf{X}\tilde{\mathbf{x}}$$

- **Key ideas**:
  - Never perform the feature transformation $\mathbf{X}_i = (\phi_1(\mathbf{x}_i), ..., \phi_1(\mathbf{x}_i))$ explicitly. Instead, define kernel function that models scalar product between feature vectors:

$$k(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{X}_i^\top \mathbf{X}_j = \langle \mathbf{X}_i, \mathbf{X}_j \rangle$$

  - Use the kernel matrix $\mathbf{K} = \mathbf{X}\mathbf{X}^\top \in \mathbb{R}^{N \times N}$, $K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$

Kernel Regression

- **Kernel ridge regression**

$$\mathbf{w} = \sum_i \alpha_i \mathbf{x}_i$$

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

- Predicted value $\tilde{y}$ for test point $\tilde{\mathbf{x}}$:

$$\tilde{y} = \mathbf{y}(\mathbf{K} + \lambda \mathbf{I})^{-1} \kappa(\tilde{\mathbf{x}})$$

with $\kappa(\tilde{\mathbf{x}}) = [k(\mathbf{x}_1, \mathbf{x}), ..., k(\mathbf{x}_N, \mathbf{x})]^\top$

- **Properties**:
  - Useful when the feature space is huge or even infinite. Many relatively simple choices have huge or infinite feature spaces associated to them.
  - Kernel trick defines very powerful feature transformations and thus solve very nonlinear problems without having to carry out the feature transformation.
  - Computing full kernel matrix can be prohitively expensive. Many tricks can reduce the complexity, e.g. Nyström approximation.
  - Regularization is very important: Kernel methods are prone to overfitting.

Kernel Regression

- **Kernel ridge regression**

$$\mathbf{w} = \sum_i \alpha_i \mathbf{x}_i$$

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

- Predicted value $\tilde{y}$ for test point $\tilde{\mathbf{x}}$:

$$\tilde{y} = \mathbf{y}(\mathbf{K} + \lambda \mathbf{I})^{-1} \kappa(\tilde{\mathbf{x}})$$

  with $\kappa(\tilde{\mathbf{x}}) = [k(\mathbf{x}_1, \mathbf{x}), ..., k(\mathbf{x}_N, \mathbf{x})]^\top$

- **Properties**:
  - Useful when the feature space is huge or even infinite. Many relatively simple choices have huge or infinite feature spaces associated to them.
  - Kernel trick defines very powerful feature transformations and thus solve very nonlinear problems without having to carry out the feature transformation.
  - Computing full kernel matrix can be prohitively expensive. Many tricks can reduce the complexity, e.g. Nyström approximation.
  - Regularization is very important: Kernel methods are prone to overfitting.

Kernel Regression

- **Kernel ridge regression**

$$\mathbf{w} = \sum_i \alpha_i \mathbf{x}_i$$

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

- Predicted value $\tilde{y}$ for test point $\tilde{\mathbf{x}}$:

$$\tilde{y} = \mathbf{y}(\mathbf{K} + \lambda \mathbf{I})^{-1} \kappa(\tilde{\mathbf{x}})$$

  with $\kappa(\tilde{\mathbf{x}}) = [k(\mathbf{x}_1, \mathbf{x}), ..., k(\mathbf{x}_N, \mathbf{x})]^\top$

- **Properties**:
  - Useful when the feature space is huge or even infinite. Many relatively simple choices have huge or infinite feature spaces associated to them.
  - Kernel trick defines very powerful feature transformations and thus solve very nonlinear problems without having to carry out the feature transformation.
  - Computing full kernel matrix can be prohitively expensive. Many tricks can reduce the complexity, e.g. Nyström approximation.
  - Regularization is very important: Kernel methods are prone to overfitting.

Kernel Regression

- **Kernel ridge regression**

$$\mathbf{w} = \sum_i \alpha_i \mathbf{x}_i$$

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

- Predicted value $\tilde{y}$ for test point $\tilde{\mathbf{x}}$:

$$\tilde{y} = \mathbf{y}(\mathbf{K} + \lambda \mathbf{I})^{-1} \kappa(\tilde{\mathbf{x}})$$

  with $\kappa(\tilde{\mathbf{x}}) = [k(\mathbf{x}_1, \mathbf{x}), ..., k(\mathbf{x}_N, \mathbf{x})]^\top$

- **Properties**:
  - Useful when the feature space is huge or even infinite. Many relatively simple choices have huge or infinite feature spaces associated to them.
  - Kernel trick defines very powerful feature transformations and thus solve very nonlinear problems without having to carry out the feature transformation.
  - Computing full kernel matrix can be prohitively expensive. Many tricks can reduce the complexity, e.g. Nyström approximation.
  - Regularization is very important: Kernel methods are prone to overfitting.

Kernel Regression

- **Kernel ridge regression**

$$\mathbf{w} = \sum_i \alpha_i \mathbf{x}_i$$

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

- Predicted value $\tilde{y}$ for test point $\tilde{\mathbf{x}}$:

$$\tilde{y} = \mathbf{y}(\mathbf{K} + \lambda \mathbf{I})^{-1} \kappa(\tilde{\mathbf{x}})$$

with $\kappa(\tilde{\mathbf{x}}) = [k(\mathbf{x}_1, \mathbf{x}), ..., k(\mathbf{x}_N, \mathbf{x})]^\top$

- **Properties**:
  - Useful when the feature space is huge or even infinite. Many relatively simple choices have huge or infinite feature spaces associated to them.
  - Kernel trick defines very powerful feature transformations and thus solve very nonlinear problems without having to carry out the feature transformation.
  - Computing full kernel matrix can be prohitively expensive. Many tricks can reduce the complexity, e.g. Nyström approximation.
  - Regularization is very important: Kernel methods are prone to overfitting.

Kernel Regression

- **Kernel ridge regression**

$$\mathbf{w} = \sum_i \alpha_i \mathbf{x}_i$$

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

- Predicted value $\tilde{y}$ for test point $\tilde{\mathbf{x}}$:

$$\tilde{y} = \mathbf{y}(\mathbf{K} + \lambda \mathbf{I})^{-1} \kappa(\tilde{\mathbf{x}})$$

  with $\kappa(\tilde{\mathbf{x}}) = [k(\mathbf{x}_1, \mathbf{x}), ..., k(\mathbf{x}_N, \mathbf{x})]^\top$

- **Properties**:
  - Useful when the feature space is huge or even infinite. Many relatively simple choices have huge or infinite feature spaces associated to them.
  - Kernel trick defines very powerful feature transformations and thus solve very nonlinear problems without having to carry out the feature transformation.
  - Computing full kernel matrix can be prohitively expensive. Many tricks can reduce the complexity, e.g. Nyström approximation.
  - Regularization is very important: Kernel methods are prone to overfitting.

Example: polynomial kernels

- A kernel function corresponds to an inner product in a feature space based on the feature mapping $\phi(\cdot)$, without having to execute this feature mapping explicitly:

$$k(\mathbf{x}_1, \mathbf{x}_2) = \langle \phi(\mathbf{x}_1), \phi(\mathbf{x}_2) \rangle.$$

- For degree-d polynomials, the polynomial kernel is defined as

$$k(\mathbf{x}_1, \mathbf{x}_2) = (\mathbf{x}_1^\top \mathbf{x}_2 + c)^d$$

- For $d = 2$ its feature mapping is given by:

$$\phi_2(\mathbf{x}) = \left[x_1^2, ..., x_n^2, \sqrt{2}x_1x_2, ..., \sqrt{2}x_{n-1}x_n, \sqrt{2c}x_1, ..., \sqrt{2c}x_n, c \right]^\top$$

- When the input features are binary-valued (booleans), then the features correspond to logical conjunctions of input features[1].

[1] Yoav Goldberg and Michael Elhadad (2008). splitSVM: Fast, Space-Efficient, non-Heuristic, Polynomial Kernel Computation for NLP Applications. Proc. ACL-08: HLT.

Example: polynomial kernels

- A kernel function corresponds to an inner product in a feature space based on the feature mapping $\phi(\cdot)$, without having to execute this feature mapping explicitly:

$$k(\mathbf{x}_1, \mathbf{x}_2) = \langle \phi(\mathbf{x}_1), \phi(\mathbf{x}_2) \rangle.$$

- For degree-d polynomials, the polynomial kernel is defined as

$$k(\mathbf{x}_1, \mathbf{x}_2) = (\mathbf{x}_1^\top \mathbf{x}_2 + c)^d$$

- For $d = 2$ its feature mapping is given by:

$$\phi_2(\mathbf{x}) = \left[x_1^2, ..., x_n^2, \sqrt{2}x_1x_2, ..., \sqrt{2}x_{n-1}x_n, \sqrt{2c}x_1, ..., \sqrt{2c}x_n, c \right]^\top$$

- When the input features are binary-valued (booleans), then the features correspond to logical conjunctions of input features[1].

[1] Yoav Goldberg and Michael Elhadad (2008). splitSVM: Fast, Space-Efficient, non-Heuristic, Polynomial Kernel Computation for NLP Applications. Proc. ACL-08: HLT.

Example: polynomial kernels

- A kernel function corresponds to an inner product in a feature space based on the feature mapping $\phi(\cdot)$, without having to execute this feature mapping explicitly:

$$k(\mathbf{x}_1, \mathbf{x}_2) = \langle \phi(\mathbf{x}_1), \phi(\mathbf{x}_2) \rangle.$$

- For degree-d polynomials, the polynomial kernel is defined as

$$k(\mathbf{x}_1, \mathbf{x}_2) = (\mathbf{x}_1^\top \mathbf{x}_2 + c)^d$$

- For $d = 2$ its feature mapping is given by:

$$\phi_2(\mathbf{x}) = \left[x_1^2, ..., x_n^2, \sqrt{2}x_1x_2, ..., \sqrt{2}x_{n-1}x_n, \sqrt{2c}x_1, ..., \sqrt{2c}x_n, c \right]^\top$$

- When the input features are binary-valued (booleans), then the features correspond to logical conjunctions of input features[1].

[1] Yoav Goldberg and Michael Elhadad (2008). splitSVM: Fast, Space-Efficient, non-Heuristic, Polynomial Kernel Computation for NLP Applications. Proc. ACL-08: HLT.

Example: polynomial kernels

- A kernel function corresponds to an inner product in a feature space based on the feature mapping $\phi(\cdot)$, without having to execute this feature mapping explicitly:

$$k(\mathbf{x}_1, \mathbf{x}_2) = \langle \phi(\mathbf{x}_1), \phi(\mathbf{x}_2) \rangle.$$

- For degree-d polynomials, the polynomial kernel is defined as

$$k(\mathbf{x}_1, \mathbf{x}_2) = (\mathbf{x}_1^\top \mathbf{x}_2 + c)^d$$

- For $d = 2$ its feature mapping is given by:

$$\phi_2(\mathbf{x}) = \left[x_1^2, ..., x_n^2, \sqrt{2}x_1x_2, ..., \sqrt{2}x_{n-1}x_n, \sqrt{2c}x_1, ..., \sqrt{2c}x_n, c \right]^\top$$

- When the input features are binary-valued (booleans), then the features correspond to logical conjunctions of input features[1].

[1] Yoav Goldberg and Michael Elhadad (2008). splitSVM: Fast, Space-Efficient, non-Heuristic, Polynomial Kernel Computation for NLP Applications. Proc. ACL-08: HLT.