Introduction to Machine Learning

Chap II: Regression

Julien Donini
LPC/Université Clermont Auvergne

Outline

Linear regression

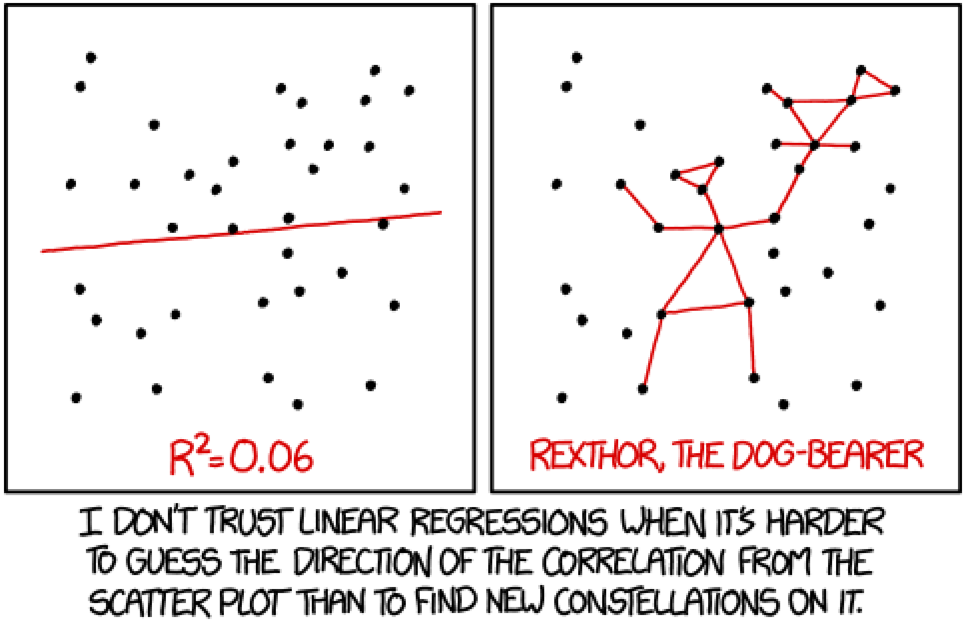
- 1D and multidimensional data
- Basis functions
- Example: polynomial curve fitting
- Regularization (*)

Likelihood and regression (*)

Notations

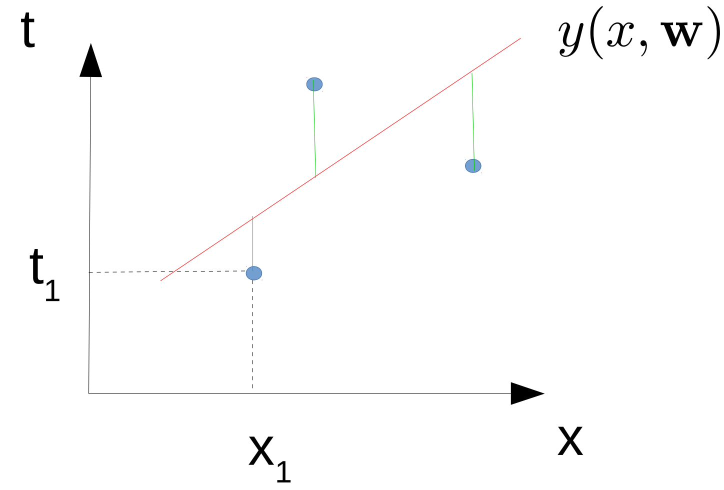
- Variable/feature: x, weight: w
- Vector: variables $\mathbf{x} = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}$; weights $\mathbf{w} = \begin{pmatrix} w_1 \\ \vdots \\ w_n \end{pmatrix}$
- Vector (transpose): $\mathbf{x}^T = (x_1, \cdots, x_n)$
- Dot product : $\mathbf{w} \cdot \mathbf{x} = \mathbf{w}^\mathbf{T}\mathbf{x} = \sum_{i=1}^{n} w_i x_i = w_1 x_1 + w_2 x_2 + ... w_n x_n.$
- Sequence of p vectors: $\{\mathbf{x}_j\}_{(j=1..p)}$
- Matrix (size $n \times m$): $\mathbf{M} = \begin{pmatrix} a_{11} & \cdots & a_{1m} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nm} \end{pmatrix}$

Linear regression



[xkcd.com]

Simple case: 1 dimensional data



Training dataset

- N observations of **feature** x = {x_1, …, x_N}
- N **Target** values t = {t_1, …, t_N}

Prediction model: straight line

$$y(x, \mathbf{w}) = y(x; w_0, w_1) = w_0 + w_1\, x$$

Weights determined by minimizing an **Error function E**

- also called **Cost function** or **Loss function** (i.e what is the consequence of your error !)

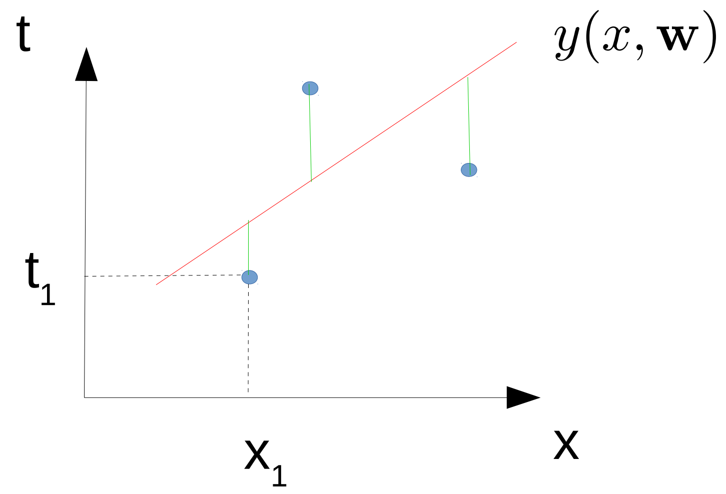
Common choice: sum of **square distance** between function and target:

$$\boxed{E(w_0, w_1) = \sum_{i=1}^{N} \{y(x_i; w_0, w_1) - t_i\}^2}$$

Simple case: 1 dimensional data

Training dataset

- N observations of **feature** $x = \{x_1, \dots, x_N\}$
- N **Target** values $t = \{t_1, \dots, t_N\}$



Prediction model: straight line

$$y(x, \mathbf{w}) = y(x; w_0, w_1) = w_0 + w_1\, x$$

Here **optimal weights** can be calculated analytically (not always possible !)

$$E(w_0, w_1) = \sum_{i=1}^{N} \{y(x_i; w_0, w_1) - t_i\}^2$$

$$\begin{cases} \dfrac{\partial E(w_0, w_1)}{\partial w_0} = 0 \\[2ex] \dfrac{\partial E(w_0, w_1)}{\partial w_1} = 0 \end{cases} \Leftrightarrow \begin{cases} w_1 = \dfrac{\mathrm{cov}(x,t)}{\mathrm{var}(x)} = r\,\dfrac{\sigma(t)}{\sigma(x)} \\[2ex] w_0 = \bar{t} - r\,\dfrac{\sigma(t)}{\sigma(x)}\,\bar{x} \end{cases}$$

(r: correlation factor between x and t)

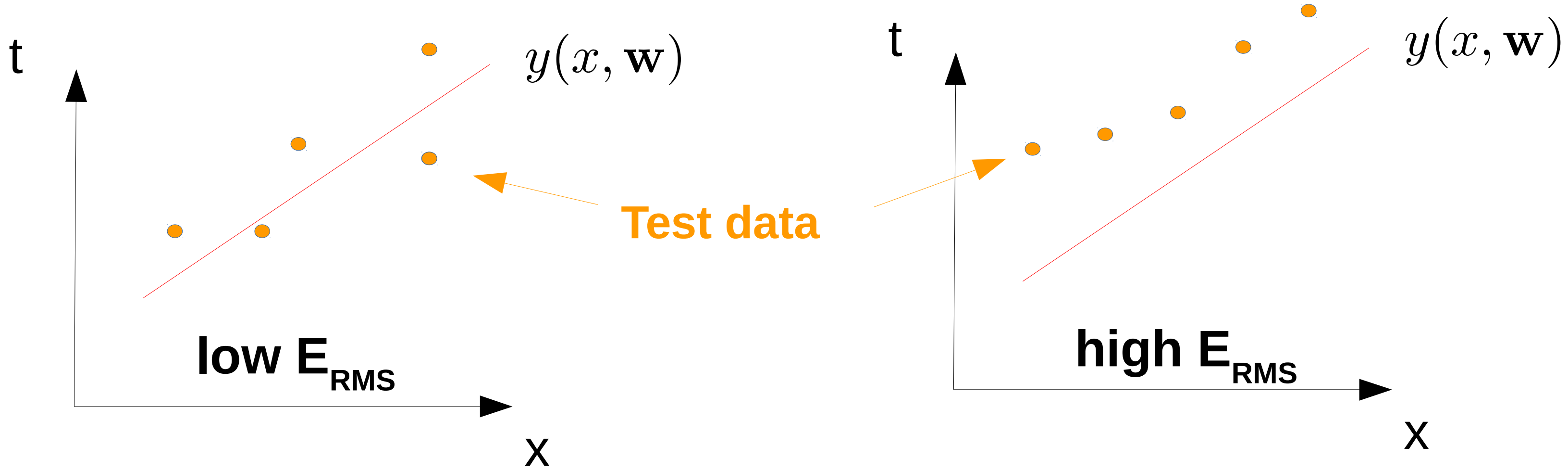
Simple case: 1 dimensional data

Training and testing

- **Training**: use dataset to determine **weights w_0 and w_1**
- **Testing**: check compatibility of y(x,**w**) on a new dataset

Measure of **compatibility**: root mean squared error (RMS)

$$E_{RMS} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\{y(x_i,\mathbf{w}) - t_i\}^2} = \sqrt{\frac{E(\mathbf{w})}{N}}$$



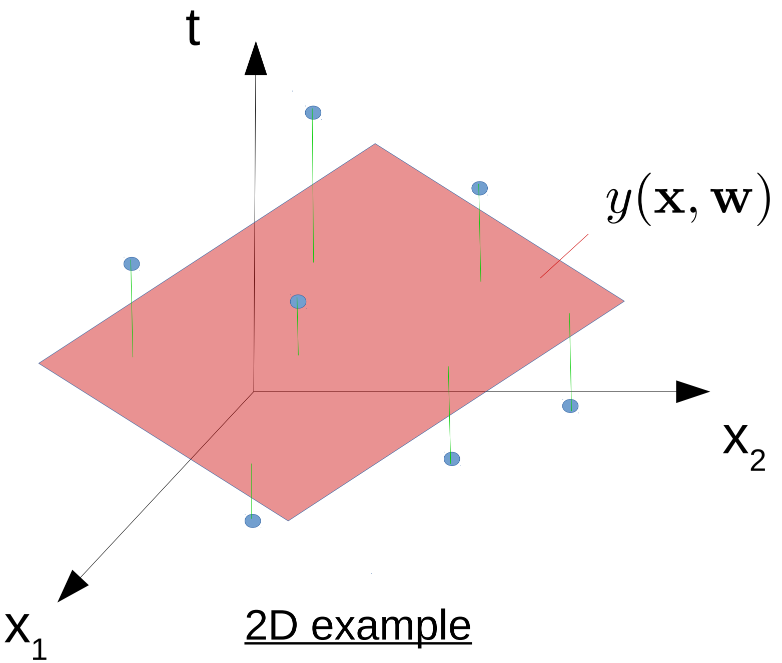
Generalization: multidimensional data

Dataset (p x 1 data)

- **N** observations of ***p***-dimensions **features**

$$\{\mathbf{x_i}\}_{i=1..N} = \{\mathbb{R}^p\} = \left\{ \begin{pmatrix} x_1 \\ \vdots \\ x_p \end{pmatrix} \right\}$$

- N **target** values t = {t_1, …, t_N}



2D example

Fit function: multidimensional plane

- Linear function with p+1 weights: **w**

$$y(\mathbf{x}, \mathbf{w}) = w_0 + \mathbf{w}^{\mathbf{T}}\mathbf{x} = w_0 + w_1 x_1 + w_2 x_2 + ... w_p x_p.$$

bias term

Generalization: multidimensional data

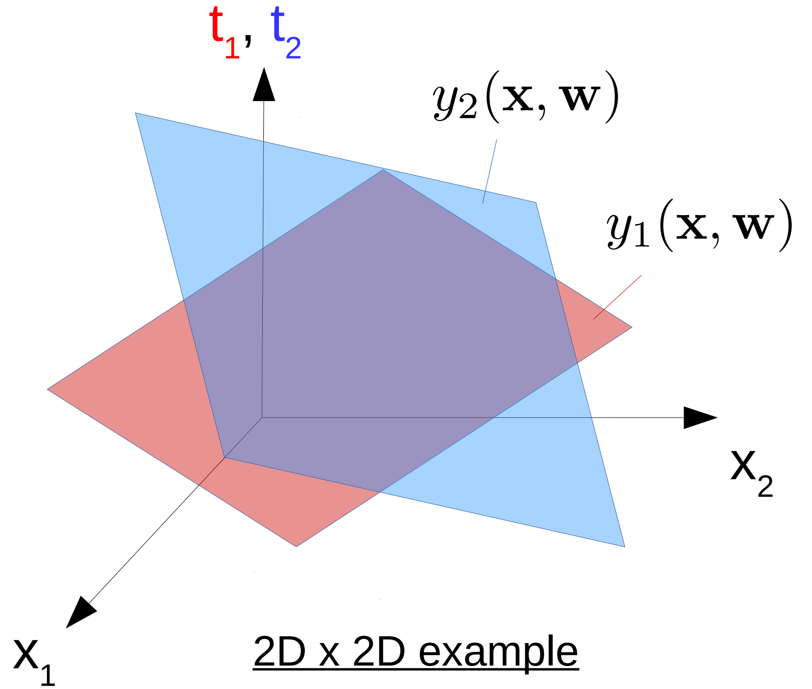
Dataset (p x q data)

- **N** observations of ***p***-dimensions **features**

$$\{\mathbf{x_i}\}_{i=1..N} = \{\mathbb{R}^p\} = \left\{ \begin{pmatrix} x_1 \\ \vdots \\ x_p \end{pmatrix} \right\}$$

- N **target** values of ***q***-dimensions

$$\{\mathbf{t_i}\}_{i=1..N} = \{\mathbb{R}^q\} = \left\{ \begin{pmatrix} t_1 \\ \vdots \\ t_q \end{pmatrix} \right\}$$



2D x 2D example

Fit functions:

$$\begin{pmatrix} y_1(\mathbf{x}, \mathbf{w}) \\ \vdots \\ y_q(\mathbf{x}, \mathbf{w}) \end{pmatrix} = \underbrace{\begin{pmatrix} w_{01} \\ \vdots \\ w_{0q} \end{pmatrix}}_{\text{bias terms}} + \begin{pmatrix} w_{11} & \cdots & w_{1p} \\ \vdots & \ddots & \vdots \\ w_{q1} & \cdots & w_{qp} \end{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ x_p \end{pmatrix}$$

Apply **M** **non-linear basis functions** ϕ to input feature **x**:

$$\mathbf{x} \longrightarrow \begin{pmatrix} \phi_1(\mathbf{x}) \\ \vdots \\ \phi_M(\mathbf{x}) \end{pmatrix}$$

$\phi_j(\mathbf{x})$: **basis function**

The regression function y(**x**, **w**) then become non-linear function of **x:**

$$\boxed{y(\mathbf{x},\mathbf{w}) = w_0 + \sum_{i=1}^{M} w_i\phi_i(\mathbf{x}) = w_0 + w_1\phi_1(\mathbf{x}) + \cdots + w_M\phi_M(\mathbf{x})}$$

These functions are called **linear models** because they are linear in **w.**

For high number of dimensions linear models suffer from **limitations**, and other approaches (as Neural Networks) are more suited.

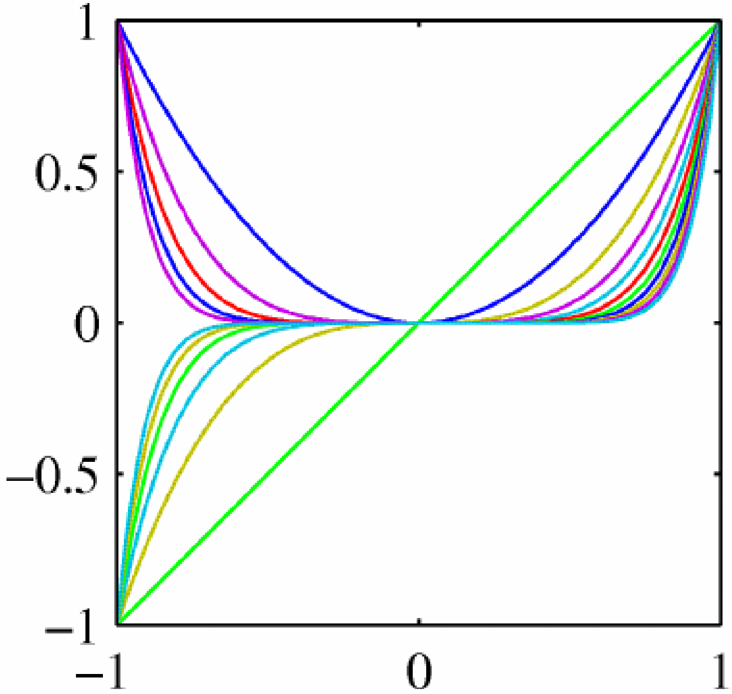
Linear basis function models

Polynomial basis functions (1D)

$$\phi_j(x) = x^j$$

$$y(x, \mathbf{w}) = \sum_{j=0}^{M-1} w_j x^j$$

Global functions of input variable
→ a small change in x affects all basis functions



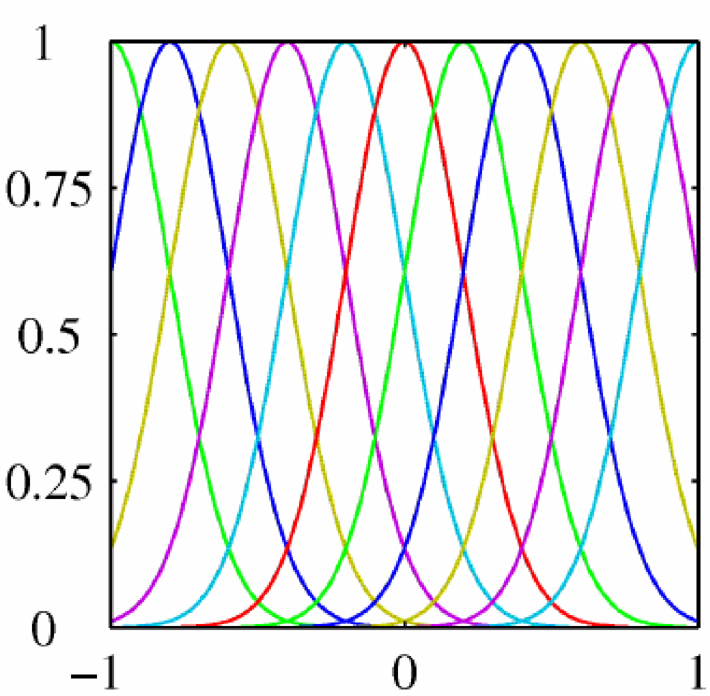
[bishop]

Linear basis function models

<u>Gaussian basis functions (1D)</u>

$$\phi_j(x) = e^{-\frac{(x-\mu_j)^2}{2\sigma^2}}$$

$$y(x, \mathbf{w}) = \sum_{j=0}^{M-1} w_j e^{-\frac{(x-\mu_j)^2}{2\sigma^2}}$$



[bishop]

Parameters:

μ_j (location) and σ (width)

Normalization is not relevant.

local functions of input variable
→ a small change in x mostly affects nearby basis functions

Linear basis function models

Sigmoidal basis functions (1D)

$$\phi_j(x) = \sigma\left(\frac{(x-\mu_j)}{s}\right)$$

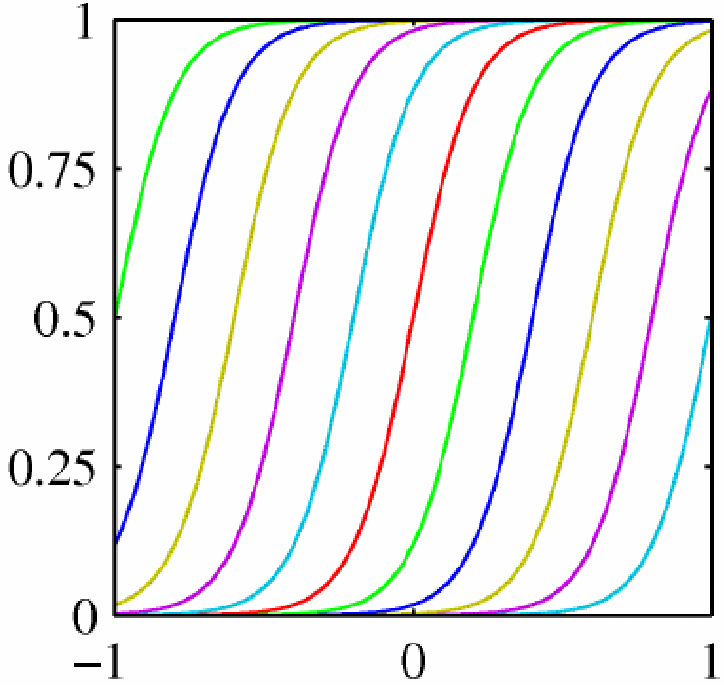
with

$$\sigma(a) = \frac{1}{1+\exp(-a)}$$

Parameters:

μ_j (location) and s (slope)

local functions of input variable
→ a small change in x mostly affects nearby basis functions

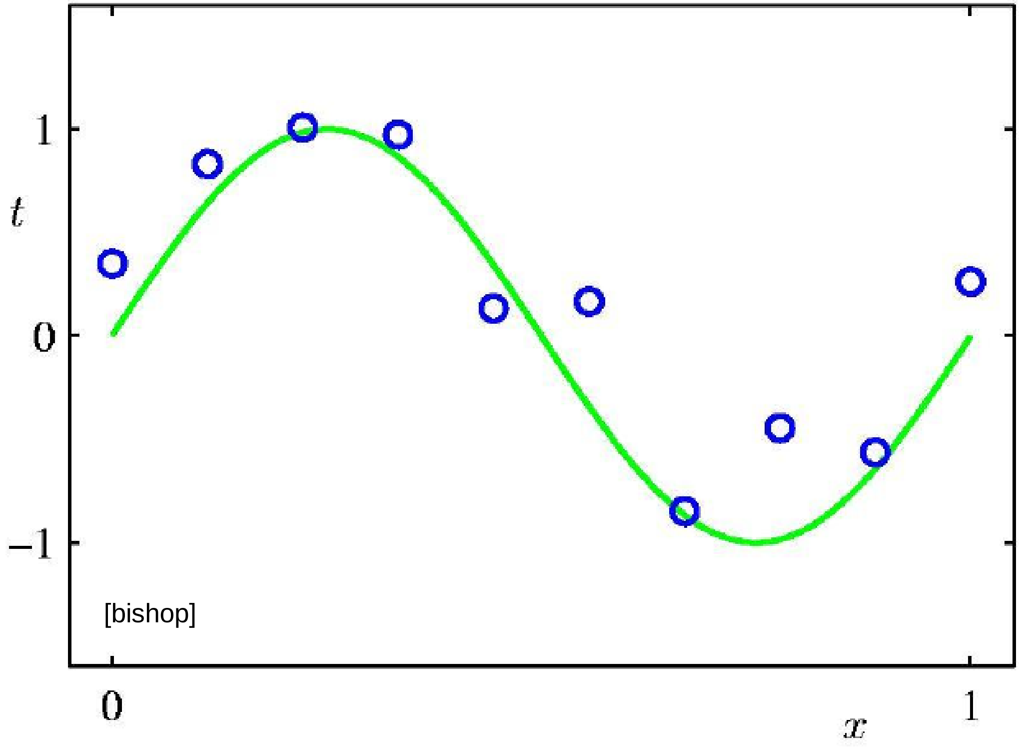


[bishop]

Example: polynomial curve fitting

Training dataset

- N observations of $x = \{x_1, \ldots, x_N\}$: uniformly spaced in [0,1]
- Target values $t = \{t_1, \ldots, t_N\}$: $\sin(2\pi x)$ + Gaussian noise



Dummy example but could be e.g. temperature (t) evolution over 1 day (x)

Polynomial curve fitting

Fit function

- Polynomial function of degree **M**, with coefficients **w** = (w_1, …, w_M)^T

$$y(x, \mathbf{w}) = w_0 + w_1 x + w_2 x^2 + \cdots + w_M x^M = \sum_{j=0}^{M} w_j x^j$$

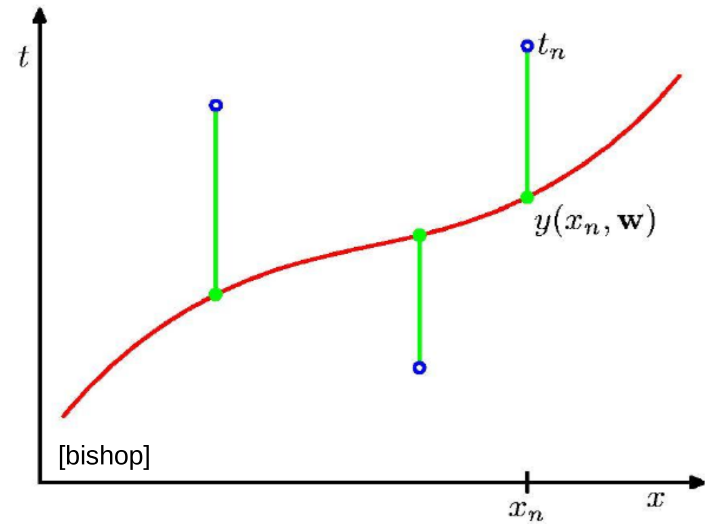
- Non-linear function of x, but linear function of **w** → **linear model**
- Values of coefficient obtained by **minimizing** an **error function**
- **Sum of the square of the errors** E(**w**)

$$E(\mathbf{w}) = \sum_{i=1}^{N} \{y(x_i, \mathbf{w}) - t_i\}^2$$

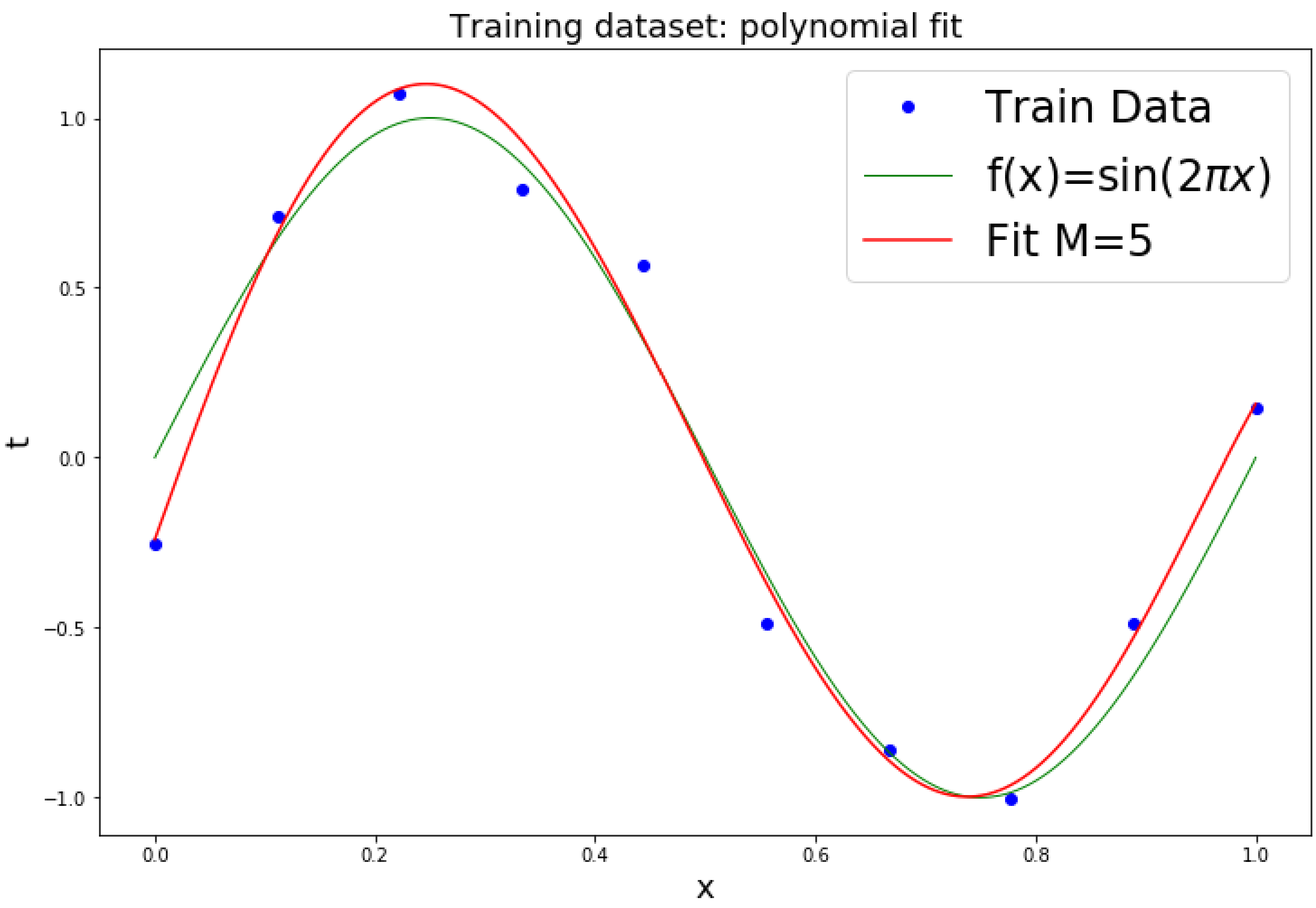
Minimization

Fitted weights **w***

E(**w***)



Tutorial: python code

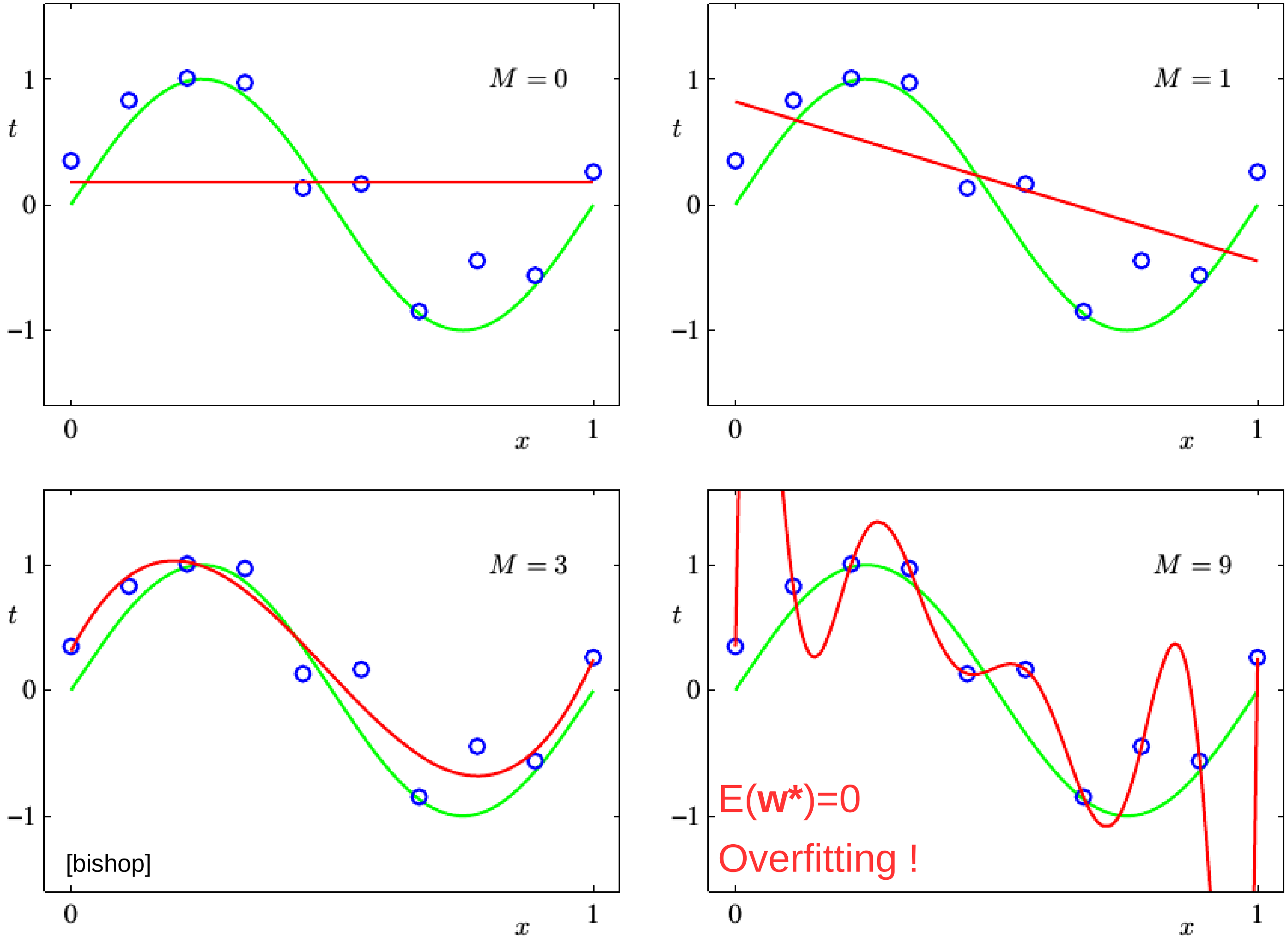


https://mybinder.org/v2/gh/judonini/MLcourses/master binder

→ Polynomial-regression.ipynb

Overfitting

$$y(x, \mathbf{w}) = w_0 + w_1 x + w_2 x^2 + \cdots + w_M x^M = \sum_{j=0}^{M} w_j x^j$$

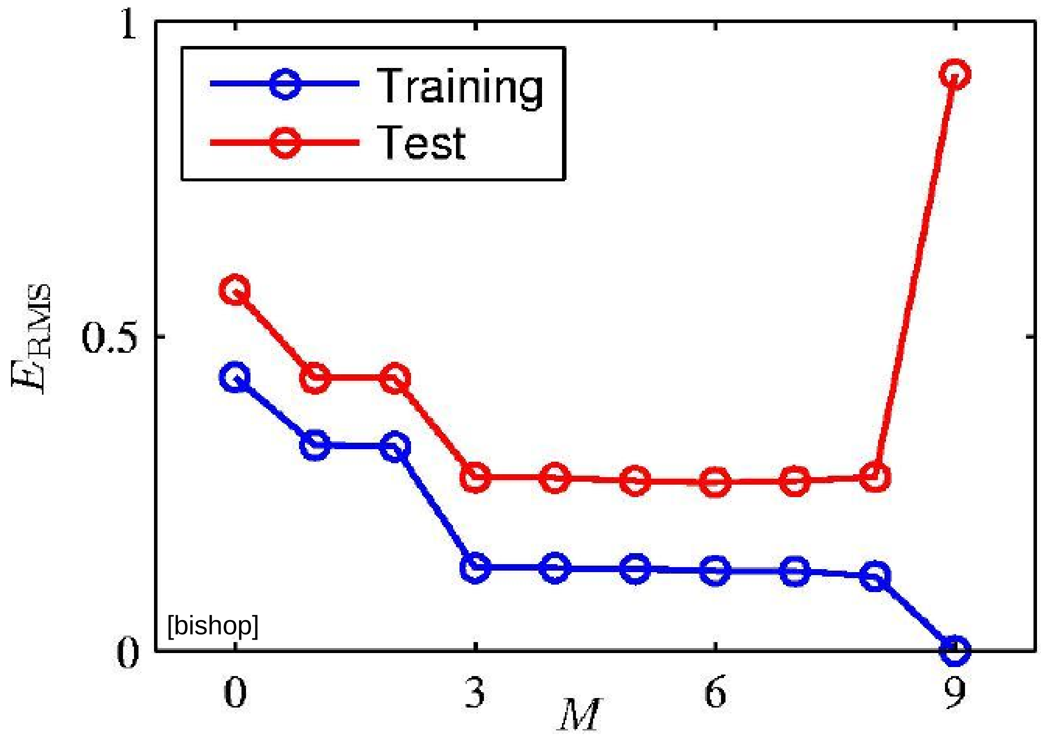


It is instructive to look at the **fitted weights** for various cases: when M increases the coefficient become **fine tuned** to data by developing large positive and negative values.

	$M = 0$	$M = 1$	$M = 3$	$M = 9$
$w_0^\star$	0.19	0.82	0.31	0.35
$w_1^\star$		-1.27	7.99	232.37
$w_2^\star$			-25.43	-5321.83
$w_3^\star$			17.37	48568.31
$w_4^\star$				-231639.30
$w_5^\star$				640042.26
$w_6^\star$				-1061800.52
$w_7^\star$				1042400.18
$w_8^\star$				-557682.99
$w_9^\star$				125201.43

Root mean squared error (RMS)

$$E_{RMS} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\{y(x_i,\mathbf{w}) - t_i\}^2} = \sqrt{\frac{E(\mathbf{w})}{N}}$$



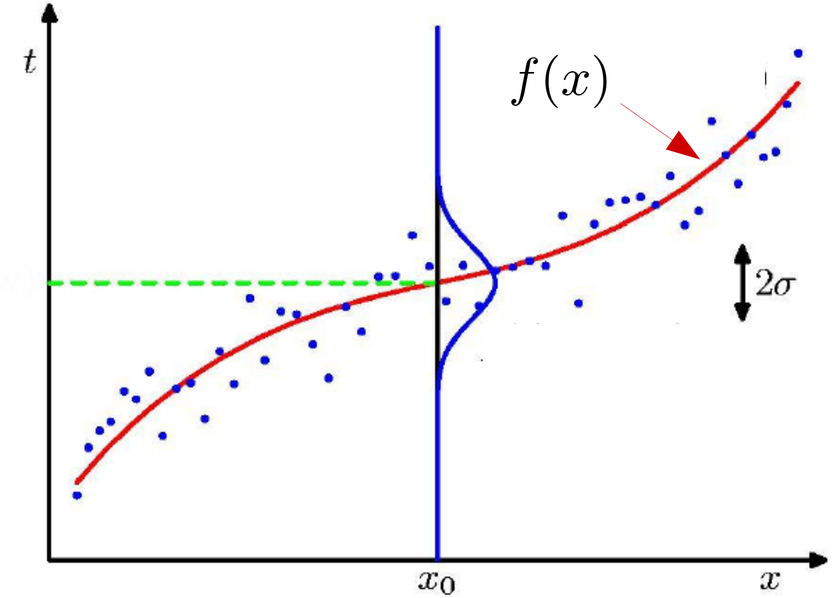
The Bias-Variance decomposition

Training dataset

- N observations of **feature** $x = \{x_1, \ldots, x_N\}$
- N **Target** values $t = \{t_1, \ldots, t_N\}$

We assume that **t** are distributed following a function: $t_i = f(x_i) + \boxed{\epsilon}$

Noise (Mean 0, variance σ²)



→ **We want to find y(x) that approximate true function f(x)**

The Bias-Variance decomposition (*)

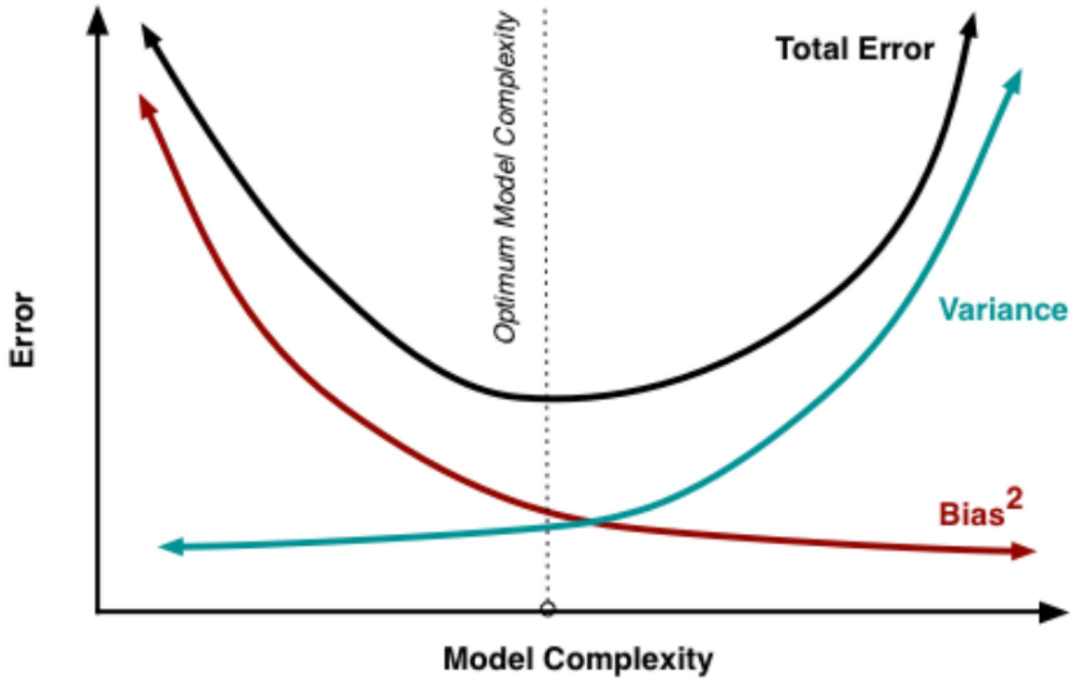
As before we determine y(x) by **minimizing:** $\sum_{i=1}^{N} \{y(x_i, \mathbf{w}) - t_i\}^2$
over the **training** dataset

The **expected error** for a **new test sample x** can be decomposed as:

- Data **noise**: minimal error of the model
- **Bias** in the model: error caused by model assumptions
- **Variance** of model: how much y(x) depends on structure of data

$$\text{squared error on y(x)} = \boxed{\sigma^2} + \boxed{(\bar{y}(x) - f(x))^2} + \boxed{E[(y(x) - \bar{y}(x))^2]}$$

The Bias-Variance decomposition

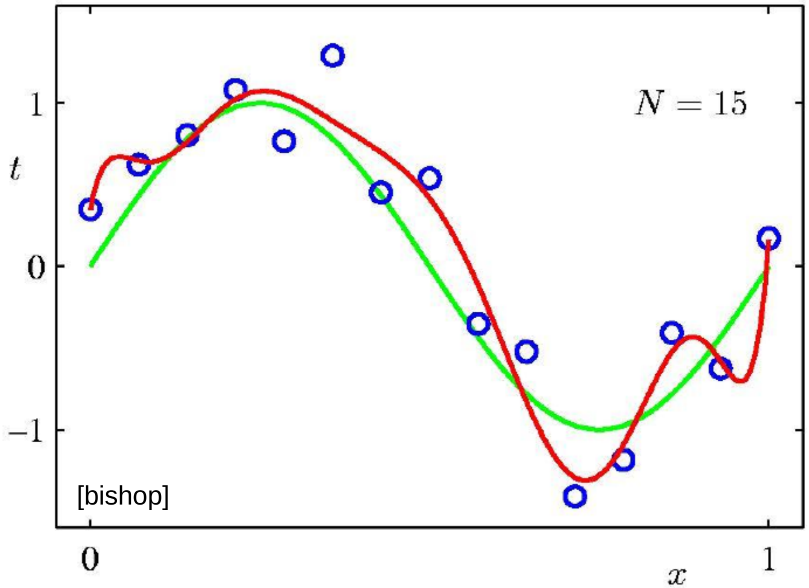


Simple models **under-fit**: deviate from data (high bias) but not influenced by structure of data (low variance)

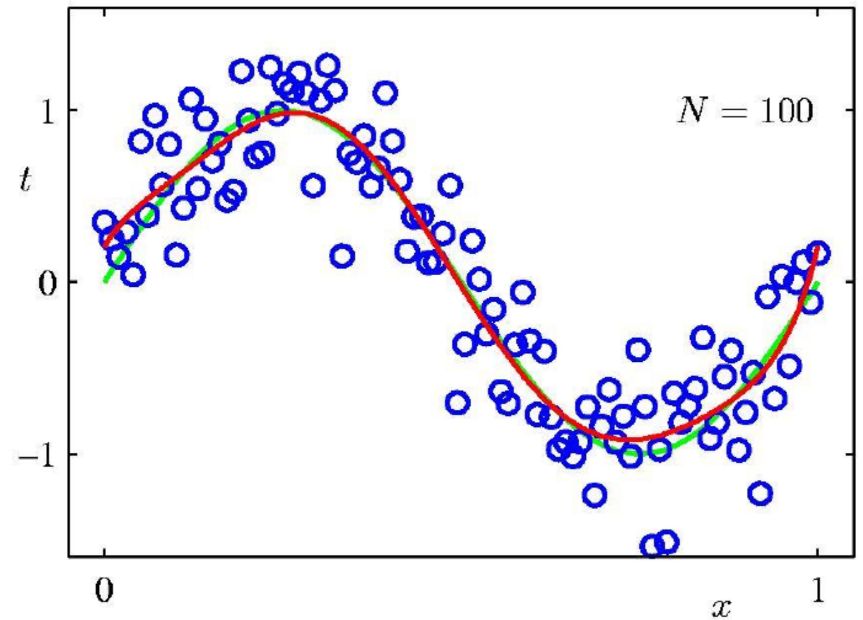
Complex models **over-fit**: small deviation from data (low bias) but very sensitive to data fluctuations (high variance)

Overfitting really depends on **N** data and **M** parameters.

M = 9



M = 9



How can we constrain the fitted parameter into reasonable values ?

→ **Regularization** techniques can be a solution.

Regularization

Add **penalization term** to error function in order to **constrain** parameters **w**.

→ Simple penalization: **ridge regression** (L2 norm)

Constrains weight to be not too large .

$$\tilde{E}(\mathbf{w}) = \sum_{i=1}^{N} \{y(x_i, \mathbf{w}) - t_i\}^2 + \boxed{\lambda ||\mathbf{w}||^2}$$

where $||\mathbf{w}||^2 = \mathbf{w}^{\mathbf{T}}\mathbf{w} = w_0^2 + \cdots + w_M^2$

and λ : parameter that governs the importance of regularization

Other choices

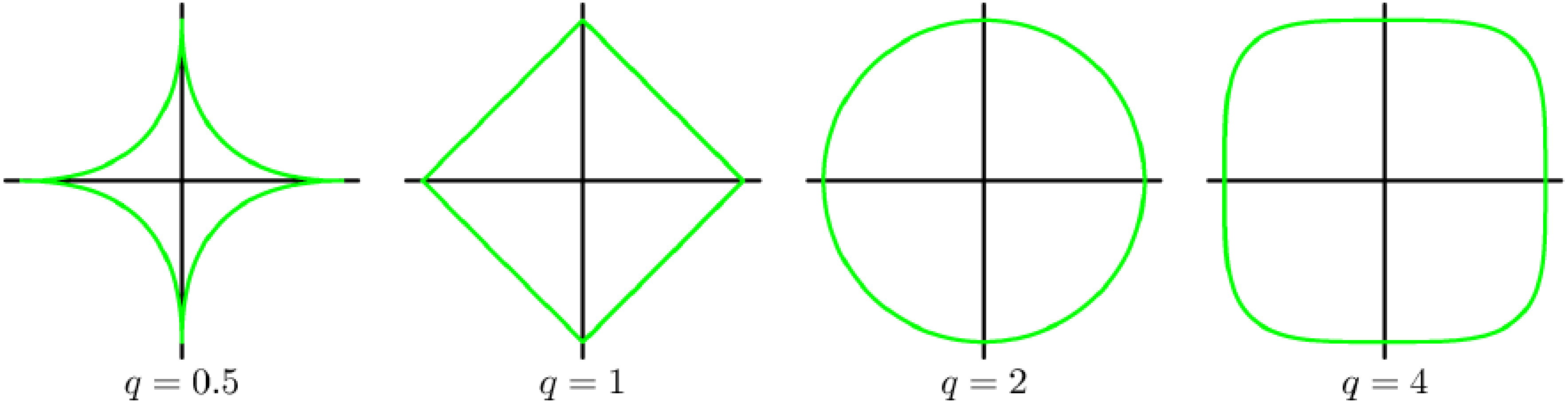
- **Lasso** regression (L1 norm): $||\mathbf{w}|| = |w_0| + ... + |w_M|$
  Reduce number of weights (set some of them to 0)
- **Elastic net**: L1 + L2 norm

Regularization

General regularization term is of the form:

$$\tilde{E}(\mathbf{w}) = \sum_{i=1}^{N} \{y(x_i, \mathbf{w}) - t_i\}^2 + \boxed{\lambda \sum_{j=1}^{M} |w_j|^q}$$

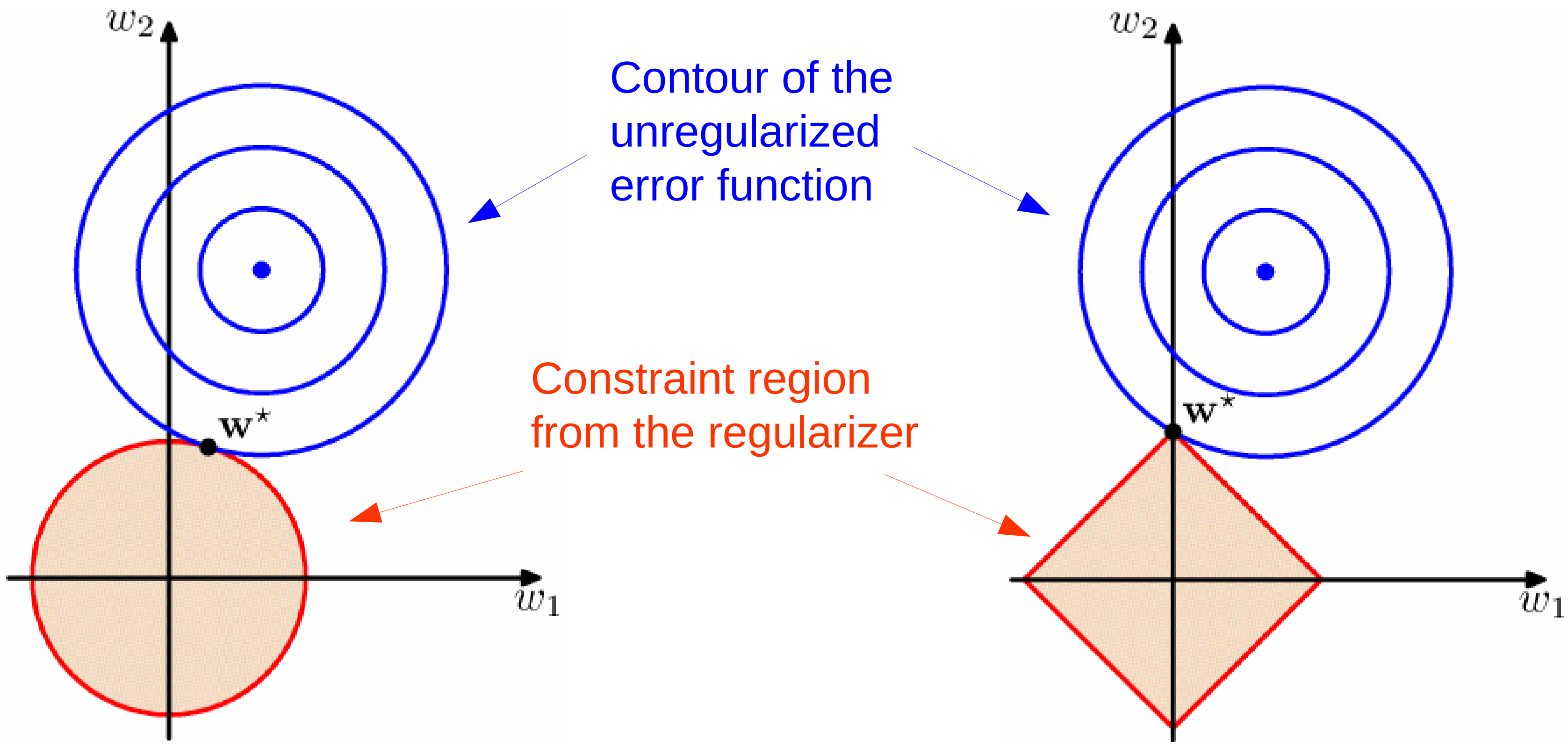
Minimizing this error function is equivalent to minimizing the unregularized sum-of-square error with the constraint $\rightarrow \boxed{\sum_{j=1}^{M} |w_j|^q \leq \eta}$



Lasso
(L1 norm)

Ridge
(L2 norm)

Regularization

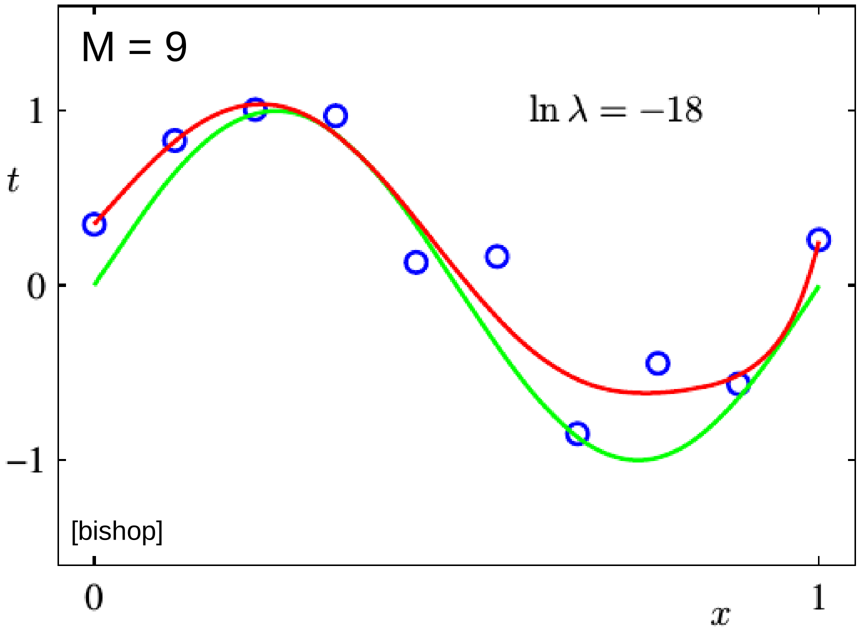


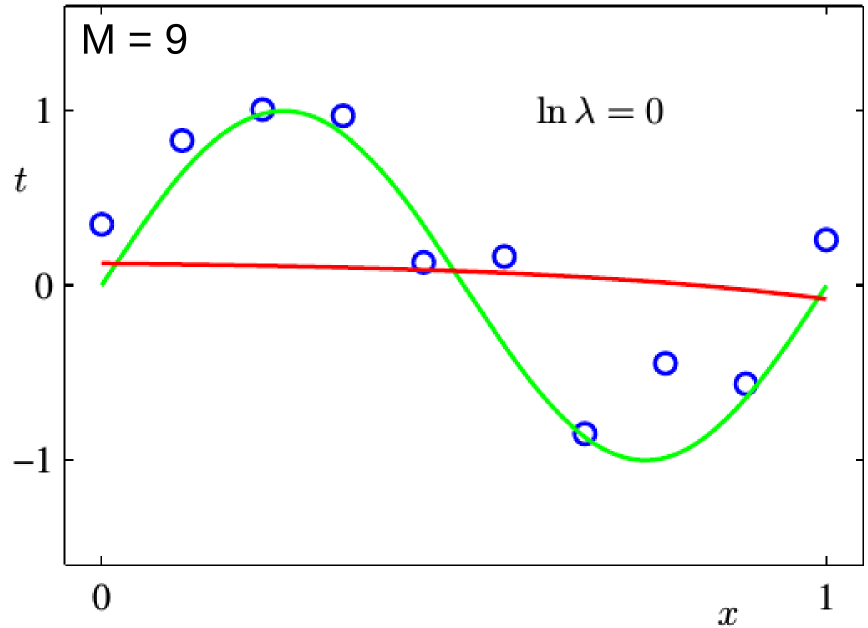
Ridge
(L2 norm)

Lasso
(L1 norm)

The optimum value for the parameter vector w is denoted by w^*.

The lasso gives a sparse solution in which $w_1^* = 0$.



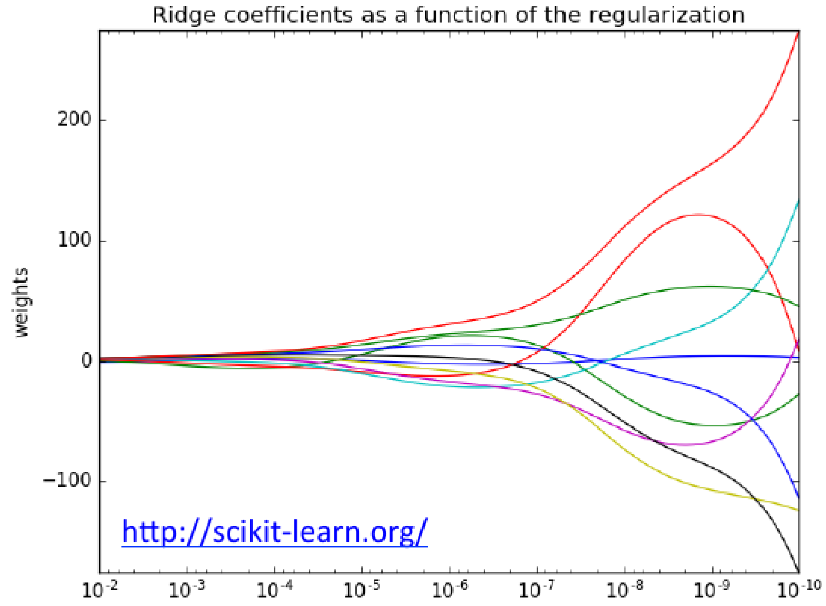


	$\ln \lambda = -\infty$	$\ln \lambda = -18$	$\ln \lambda = 0$
$w_0^\star$	0.35	0.35	0.13
$w_1^\star$	232.37	4.74	-0.05
$w_2^\star$	-5321.83	-0.77	-0.06
$w_3^\star$	48568.31	-31.97	-0.05
$w_4^\star$	-231639.30	-3.89	-0.03
$w_5^\star$	640042.26	55.28	-0.02
$w_6^\star$	-1061800.52	41.32	-0.01
$w_7^\star$	1042400.18	-45.95	-0.00
$w_8^\star$	-557682.99	-91.53	0.00
$w_9^\star$	125201.43	72.68	0.01

Effect of L2 norm regularization

- ln λ = -inf : no regularization
- ln λ = -18 : suppressed overfitting
- ln λ = 0: fit too constrained

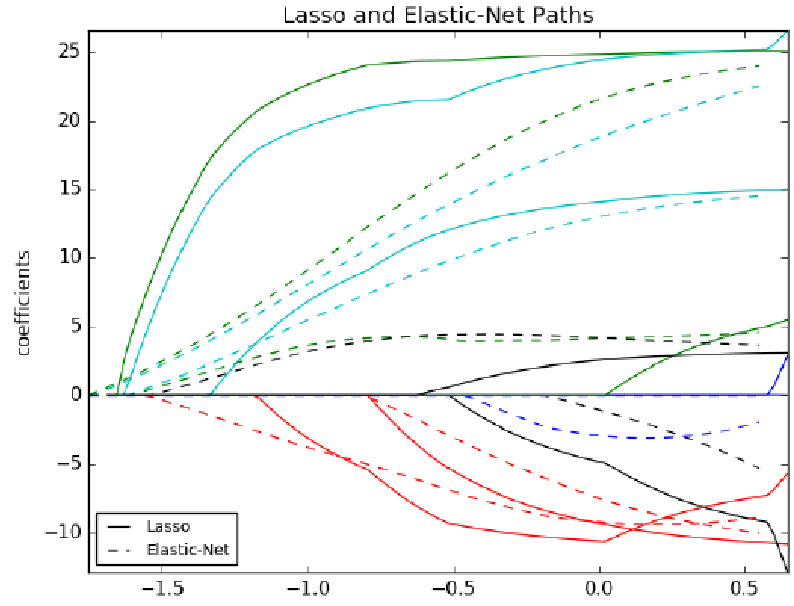
L2 norm: $\lambda \, ||\mathbf{w}||^2$



More constraints

Less constraints

L1 norm: $\lambda \, ||\mathbf{w}||$



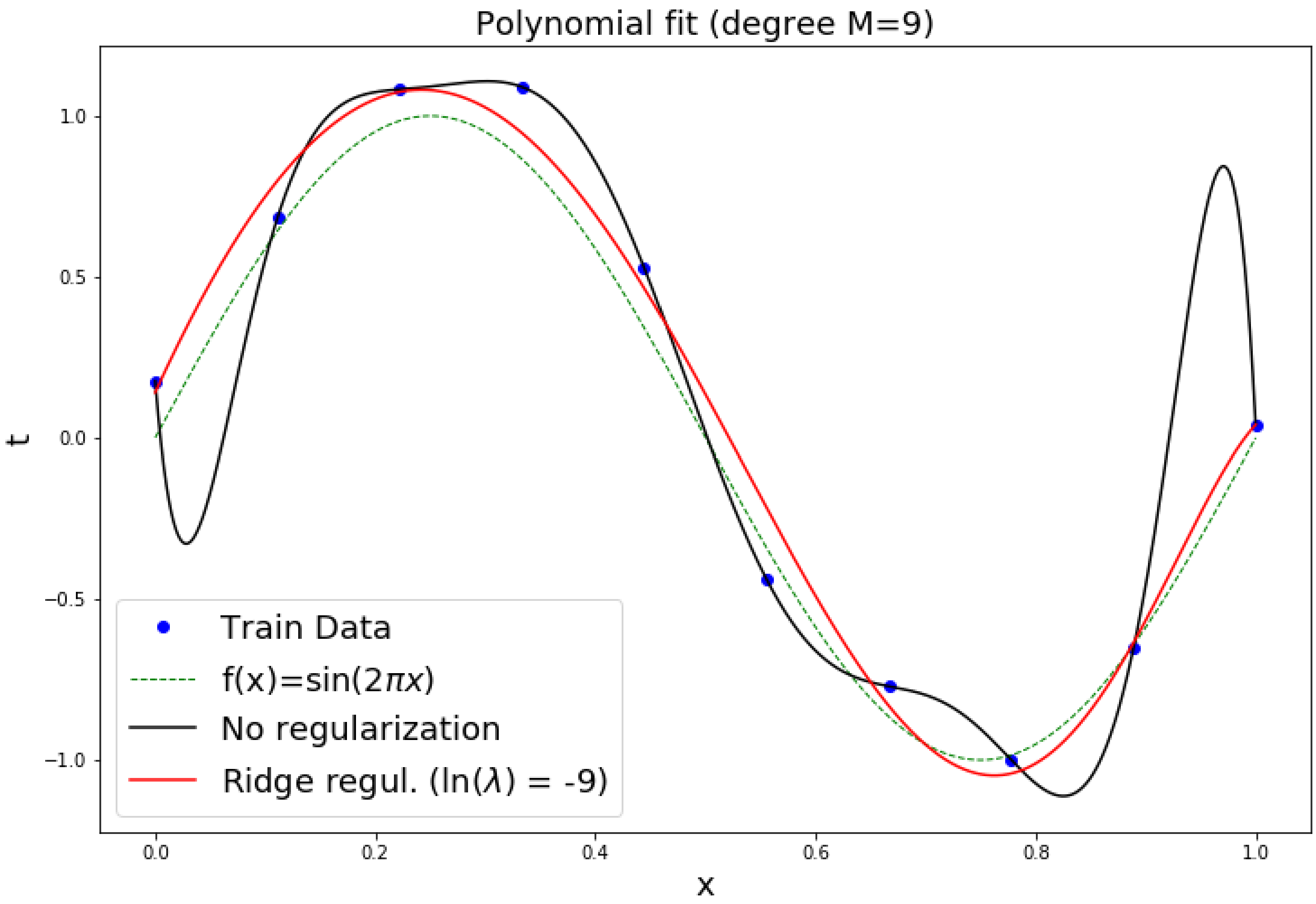
More constraints

Less constraints

Affects value of coefficients (shrinkage)

Affects number of coefficients (sparsity)

Tutorial: Python code



https://mybinder.org/v2/gh/judonini/MLcourses/master binder

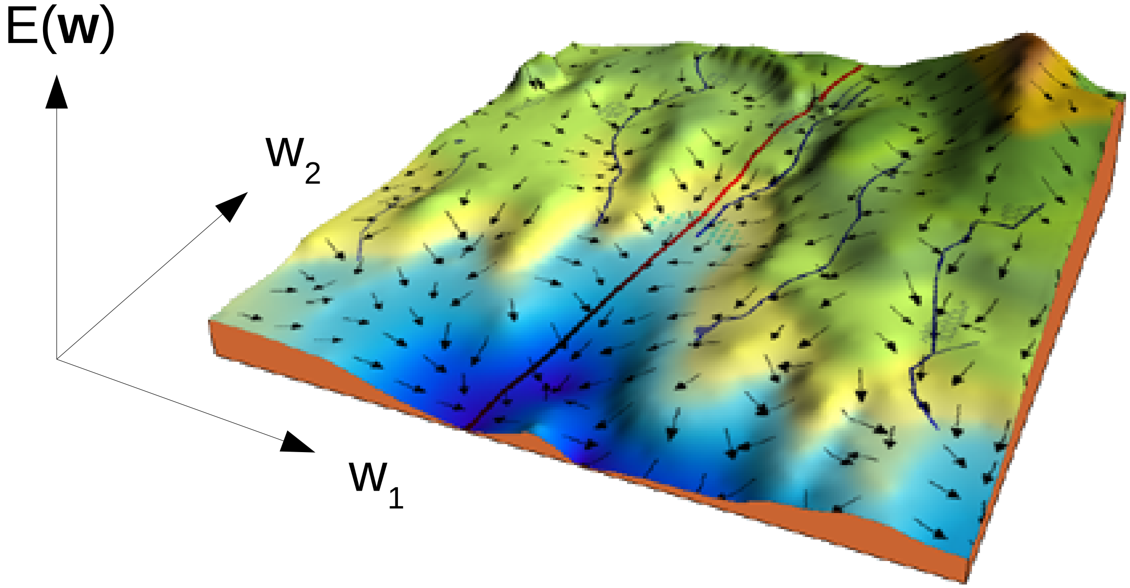
→ Polynomial-regression-regularization.ipynb

Gradient descent

How can we **minimize** the error function for complex cases (ex: when there is no analytic solution) ?

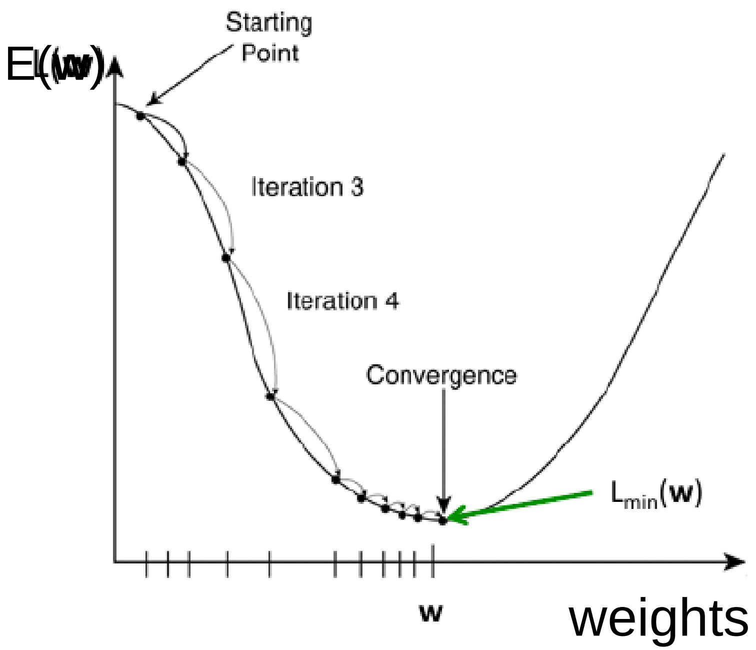
→ **Solution: Gradient descent**

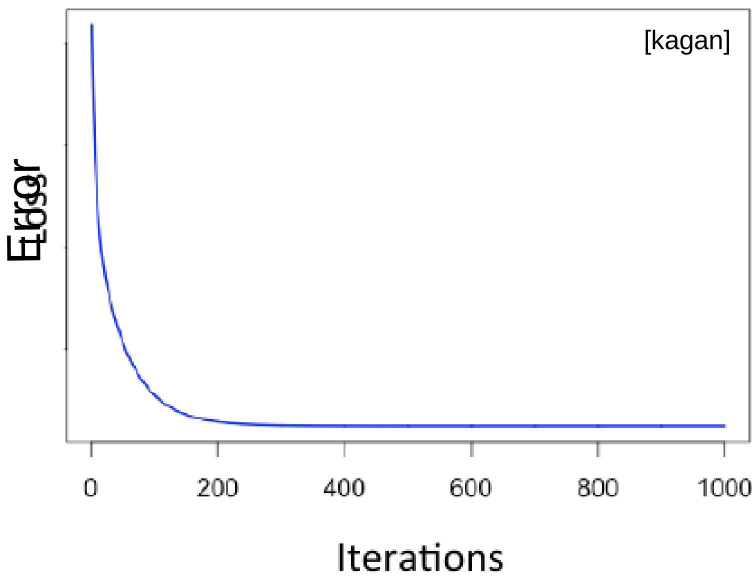
Iteratively move in the direction of steepest descent as defined by the negative of the gradient of the error function



Gradient descent

Descend along the error function to find a (local) minimum:





Direction of descent:

→ (negative of the **gradient** of the error function) × (**learning rate**)

Gradient descent

Simple example: fit N data points with linear function: $y(x, \mathbf{w}) = w_0 + w_1 x$

Error function and its **derivatives**

$$E(w_0, w_1) = \sum_{i=1}^{N} \{y(x_i, \mathbf{w}) - t_i\}^2 = \sum_{i=1}^{N} \{(w_0 + w_1 x_i) - t_i\}^2$$

$$\longrightarrow \begin{cases} \frac{\partial E(w_0, w_1)}{\partial w_0} = \sum_{i=1}^{N} 2 \{(w_0 + w_1 x_i) - t_i\} \\ \frac{\partial E(w_0, w_1)}{\partial w_1} = \sum_{i=1}^{N} 2x_i \{(w_0 + w_1 x_i) - t_i\} \end{cases}$$

Iterative update **rule**:

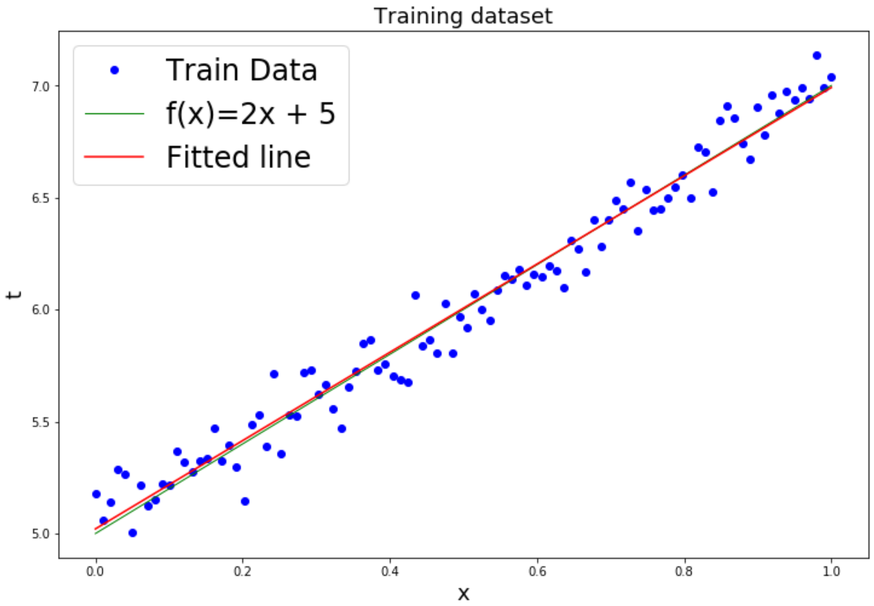
$$\boxed{\begin{aligned} \mathrm{w}_0^{(k)} &\rightarrow w_0^{(k+1)} = w_0^{(k)} - \frac{\partial E(w_0, w_1)}{\partial w_0} \times \eta \\ \mathrm{w}_1^{(k)} &\rightarrow w_1^{(k+1)} = w_1^{(k)} - \frac{\partial E(w_0, w_1)}{\partial w_1} \times \eta \end{aligned}}$$

k: iteration number

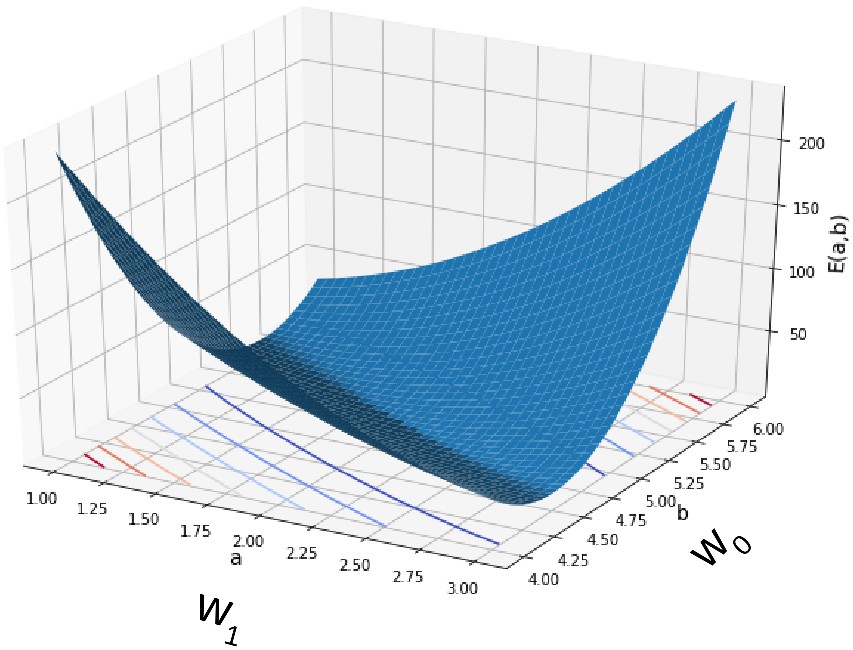
η: learning rate

Repeat until convergence

Input data: $\{x_i, t_i\}$



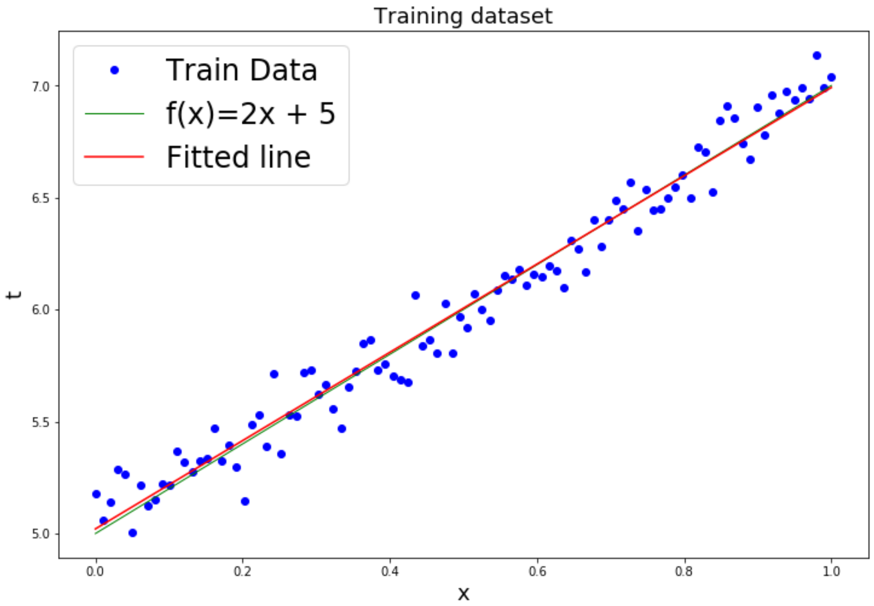
Error function



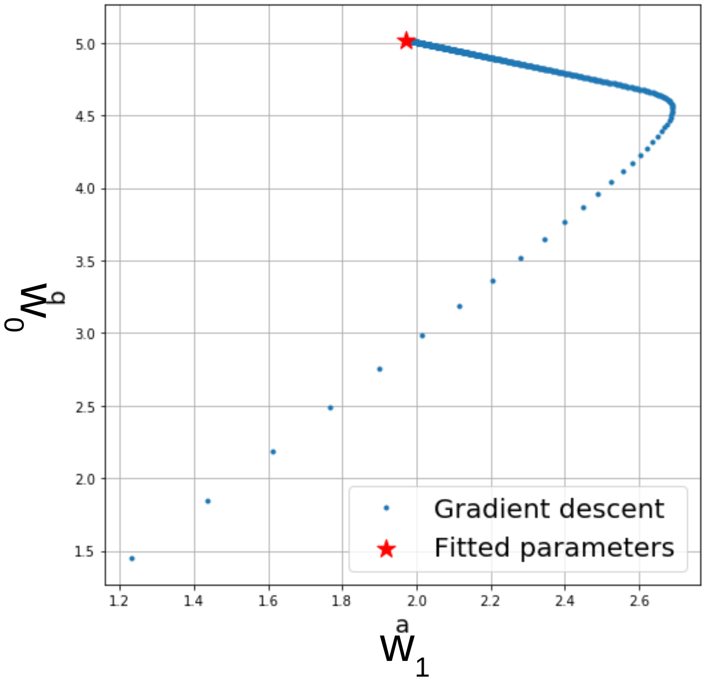
$$E(w_0, w_1) = \sum_{i=1}^{N} \{(w_0 + w_1 x_i) - t_i\}^2$$

Gradient descent

Input data: $\{x_i, t_i\}$



Gradient descent



$$\mathrm{w}_0^{(k)} \rightarrow w_0^{(k+1)} = w_0^{(k)} - \frac{\partial E(w_0, w_1)}{\partial w_0} \times \eta$$

$$\mathrm{w}_1^{(k)} \rightarrow w_1^{(k+1)} = w_1^{(k)} - \frac{\partial E(w_0, w_1)}{\partial w_1} \times \eta$$

1000 iterations

learning rate η = 0.05

Stochastic gradient descent

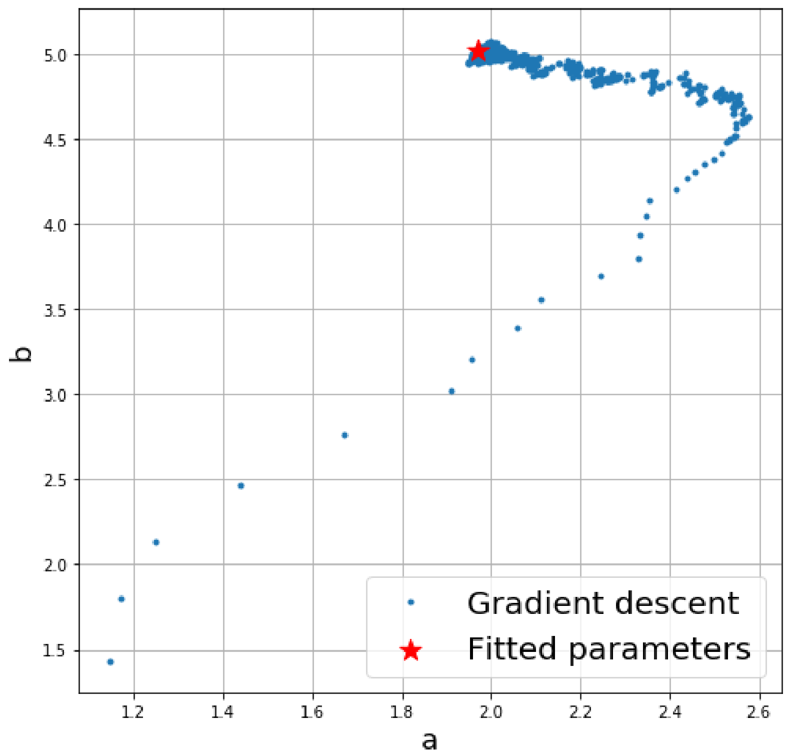
Gradient descent can be **computationally costly** for large N since the gradient is calculated over full training set.

→ **Solution: Stochastic gradient descent**

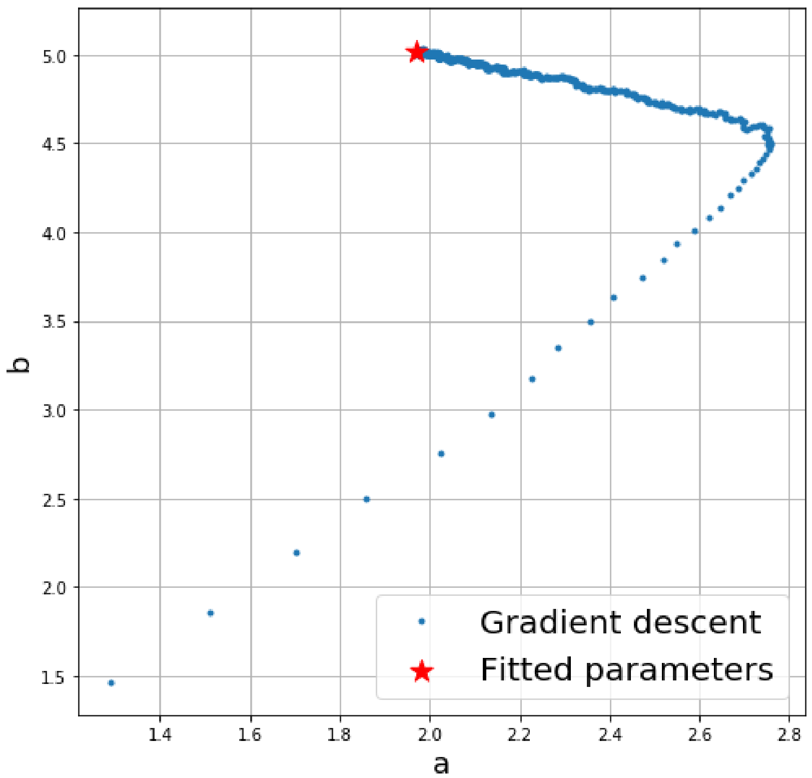
Compute gradient on a small **batch** of events (can be 1 event):

$$\begin{cases} \frac{\partial E(w_0,w_1)}{\partial w_0} = \sum_{i \subset N} 2\left\{(w_0 + w_1 x_i) - t_i\right\} \\ \frac{\partial E(w_0,w_1)}{\partial w_1} = \sum_{i \subset N} 2x_i\left\{(w_0 + w_1 x_i) - t_i\right\} \end{cases}$$

Stochastic gradient descent



Gradient calculated on 1 (random) event at each step



Gradient calculated on 10 (random) events at each step

Going further (*)

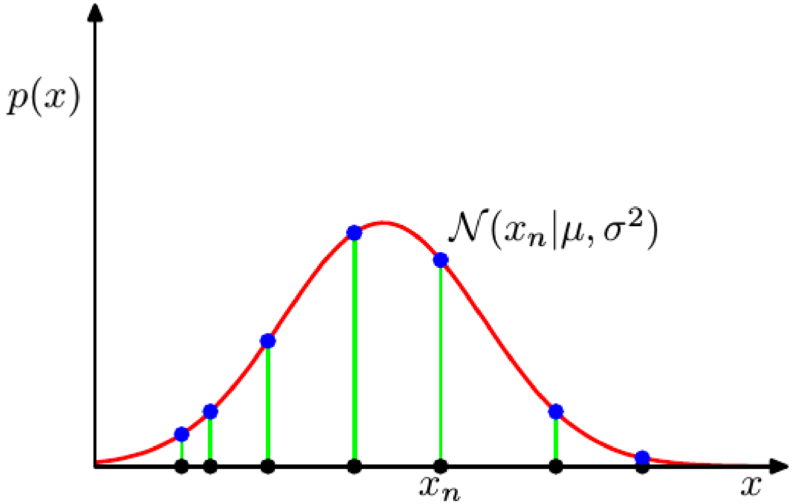
Likelihood

Consider **N** measurements of x distributed along a given probability law p(x).

$$\mathbf{x} = (x_1, \ldots, x_N)^T$$

where values x_i are **independent and identically distributed** (i.i.d).

Ex: Normal (a.k.a Gaussian) law with 2 parameters: mean μ and variance σ^2



[bishop]

$$\mathcal{N}\left(x|\mu,\sigma^2\right) = \frac{1}{(2\pi\sigma^2)^{1/2}} \exp\left\{-\frac{1}{2\sigma^2}(x-\mu)^2\right\}$$

What is the *likelihood* of this set of measurements ?

Can we estimate μ and σ given **x** ?

Likelihood and parameter estimation

Since the variables x are i.i.d we can write the joint probability distribution, therefore the **likelihood** of the dataset, given μ and σ is:

$$\mathcal{L}(\mu, \sigma^2; \mathbf{x}) \equiv p(\mathbf{x}|\mu, \sigma^2) = \prod_{n=1}^{N} \mathcal{N}(x_n|\mu, \sigma^2)$$

To **estimate** μ and σ given **x** one **maximizes** *p* w.r.t these parameters.
In practice often maximize ln(*p*) or minimize -ln(*p*).

$$\ln p(\mathbf{x}|\mu, \sigma^2) = -\frac{1}{2\sigma^2}\sum_{n=1}^{N}(x_n - \mu)^2 - \frac{N}{2}\ln\sigma^2 - \frac{N}{2}\ln(2\pi)$$

$$\begin{cases} \dfrac{\partial(\ln p(\mathbf{x}|\mu, \sigma^2))}{\partial\mu} = 0 \to \mu_{\mathrm{ML}} = \dfrac{1}{N}\displaystyle\sum_{n=1}^{N} x_n \\ \dfrac{\partial(\ln p(\mathbf{x}|\mu, \sigma^2))}{\partial\sigma} = 0 \to \sigma_{\mathrm{ML}} = \dfrac{1}{N}\displaystyle\sum_{n=1}^{N}(x_n - \mu_{\mathrm{ML}})^2 \end{cases}$$

Expected values

$$\begin{aligned} \mathbb{E}[\mu_{\mathrm{ML}}] &= \mu \\ \mathbb{E}[\sigma^2_{\mathrm{ML}}] &= \left(\frac{N-1}{N}\right)\sigma^2 \end{aligned}$$

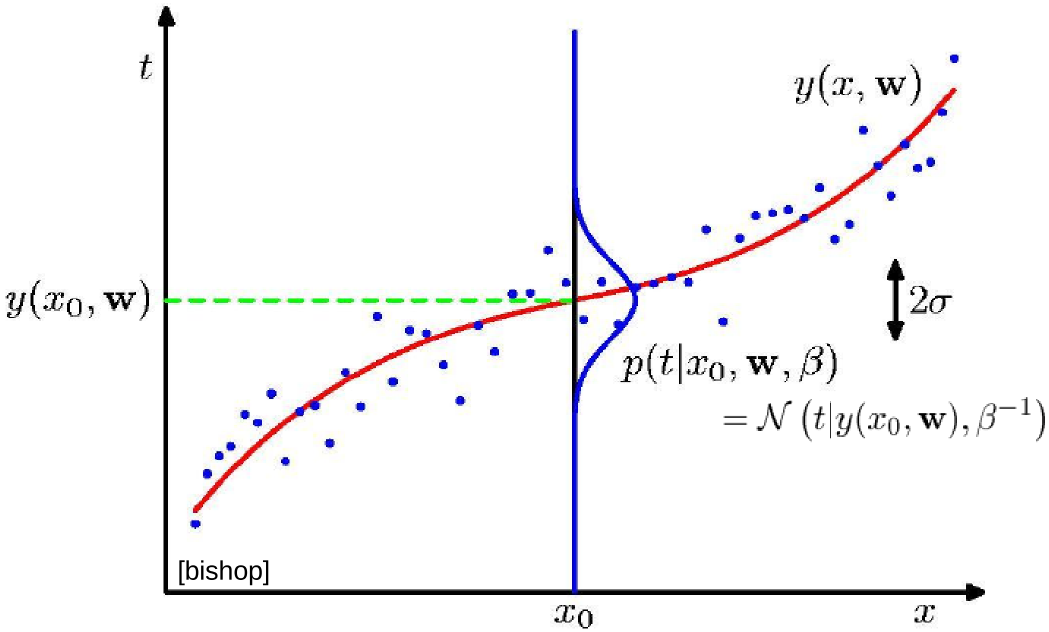
Likelihood and regression (*)

Curve fitting with noise

Assume target variable in training dataset is subject to Gaussian noise

$$p(t|x, \mathbf{w}, \beta) = \mathcal{N}\left(t|y(x, \mathbf{w}), \beta^{-1}\right)$$

where $\beta = \dfrac{1}{\sigma^2}$ is a precision parameter.



Predictive probabilistic model

By maximizing the likelihood on the training dataset we obtain a probabilistic predictive model for *t* (instead of a single point estimate):

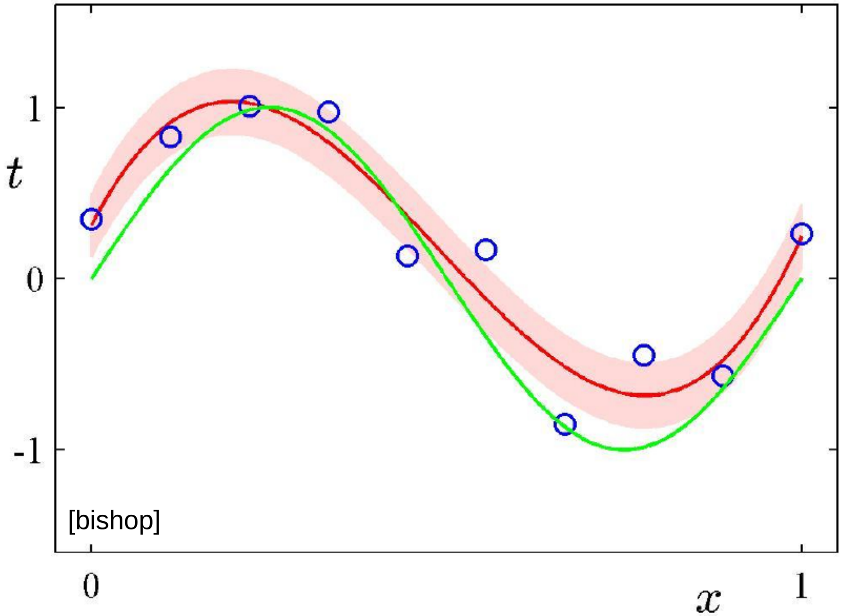
$$\boxed{p(t|x, \mathbf{w}_{\mathrm{ML}}, \beta_{\mathrm{ML}}) = \mathcal{N}\left(t|y(x, \mathbf{w}_{\mathrm{ML}}), \beta_{\mathrm{ML}}^{-1}\right)}$$

where $\mathbf{w}_{\mathrm{ML}}$ is obtained by minimizing the sum of square error E(**w**)

$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\{y(x_n, \mathbf{w}) - t_n\}^2$$

and β_{ML} is given by

$$\frac{1}{\beta_{\mathrm{ML}}} = \frac{1}{N}\sum_{n=1}^{N}\{y(x_n, \mathbf{w}_{\mathrm{ML}}) - t_n\}^2$$



Exercice: show this !

Hints →

$$p(\mathbf{t}|\mathbf{x}, \mathbf{w}, \beta) = \prod_{n=1}^{N} \mathcal{N}\left(t_n | y(x_n, \mathbf{w}), \beta^{-1}\right)$$

$$\ln p(\mathbf{t}|\mathbf{x}, \mathbf{w}, \beta) = \underbrace{-\frac{\beta}{2}\sum_{n=1}^{N}\{y(x_n, \mathbf{w}) - t_n\}^2}_{\beta E(\mathbf{w})} + \frac{N}{2}\ln\beta - \frac{N}{2}\ln(2\pi)$$

$\mathbf{w}_{ML}$ is determined by minimizing sum of square error E(**w**), then:

$$\frac{1}{\beta_{\mathrm{ML}}} = \frac{1}{N}\sum_{n=1}^{N}\{y(x_n, \mathbf{w}_{\mathrm{ML}}) - t_n\}^2$$

Likelihood and regression (*)

Bayes theorem and likelihood

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

Likelihood of observing
these data given a theory

Posterior knowledge
on theory

Prior knowledge
on theory

$$P(\text{theory}|\text{data}) = \frac{P(\text{data}|\text{theory})P(\text{theory})}{P(\text{data})}$$

Usually just a
normalisation factor

Bayesian approach

We assume that unknown parameters **w** follow a Gaussian ***prior*** of the form:

$$p(\mathbf{w}|\alpha) = \mathcal{N}(\mathbf{w}|\mathbf{0}, \alpha^{-1}\mathbf{I}) = \left(\frac{\alpha}{2\pi}\right)^{(M+1)/2} \exp\left\{-\frac{\alpha}{2}\mathbf{w}^{\mathrm{T}}\mathbf{w}\right\}$$

According to **Bayes Theorem** we have the ***posterior***:

$$p(\mathbf{w}|\mathbf{x}, \mathbf{t}, \alpha, \beta) \propto p(\mathbf{t}|\mathbf{x}, \mathbf{w}, \beta)p(\mathbf{w}|\alpha)$$

Taking -ln(p) → the maximum of the posterior is given by the minimum of:

$$\frac{\beta}{2}\sum_{n=1}^{N}\{y(x_n, \mathbf{w}) - t_n\}^2 + \boxed{\frac{\alpha}{2}\mathbf{w}^{\mathrm{T}}\mathbf{w}}$$

Regularization parameter with λ=α/β

Backup slides

Likelihood and regression

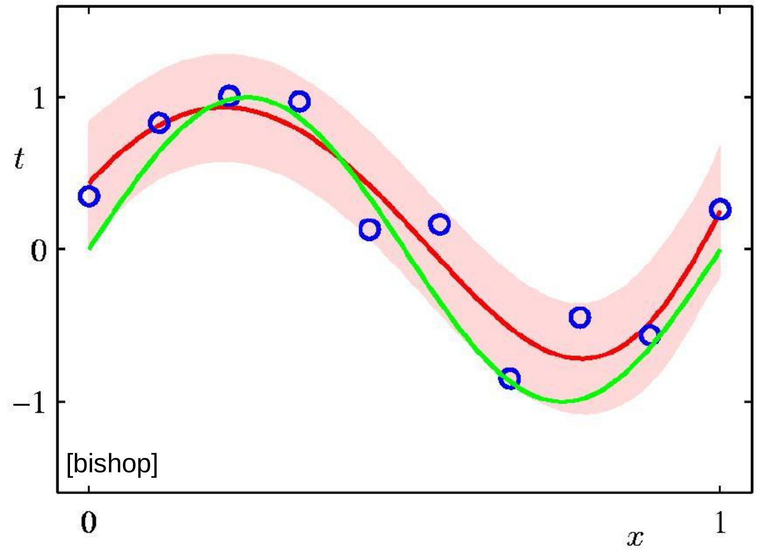
Bayesian curve fitting

$$p(t|x, \mathbf{x}, \mathbf{t}) = \int p(t|x, \mathbf{w}) p(\mathbf{w}|\mathbf{x}, \mathbf{t}) \, \mathrm{d}\mathbf{w} = \mathcal{N}\left(t|m(x), s^2(x)\right)$$

$$m(x) = \beta \boldsymbol{\phi}(x)^{\mathrm{T}} \mathbf{S} \sum_{n=1}^{N} \boldsymbol{\phi}(x_n) t_n \qquad s^2(x) = \beta^{-1} + \boldsymbol{\phi}(x)^{\mathrm{T}} \mathbf{S} \boldsymbol{\phi}(x)$$

$$\mathbf{S}^{-1} = \alpha \mathbf{I} + \beta \sum^{N} \boldsymbol{\phi}(x_n) \boldsymbol{\phi}(x_n)^{\mathrm{T}} \qquad \boldsymbol{\phi}(x_n) = \left(x_n^0, \ldots, x_n^M\right)^{\mathrm{T}}$$

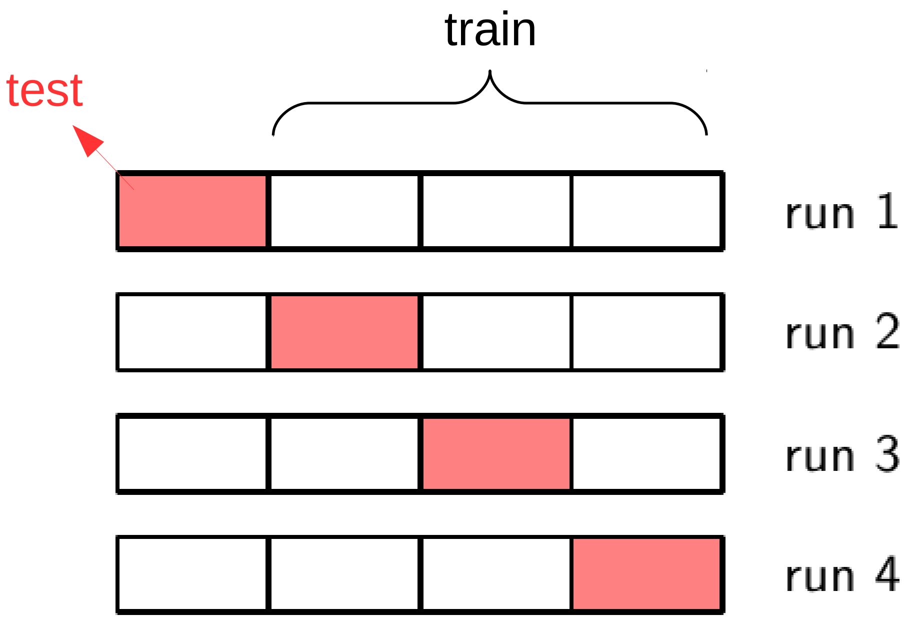
$$p(t|x, \mathbf{x}, \mathbf{t}) = \mathcal{N}\left(t|m(x), s^2(x)\right)$$



S-fold cross-validation

Divide data in S groups, use S-1 for training and test on left-over group

Rinse and repeat S times



$$E_1 = \sum_{i=1}^{N} (y_i - y(x_i, \mathbf{w}^*))^2$$

$$E_2 = \sum_{i=1}^{N} (y_i - y(x_i, \mathbf{w}^*))^2$$

...

Cross-validation error: $\mathrm{CV} = \frac{1}{S} \sum E_i$

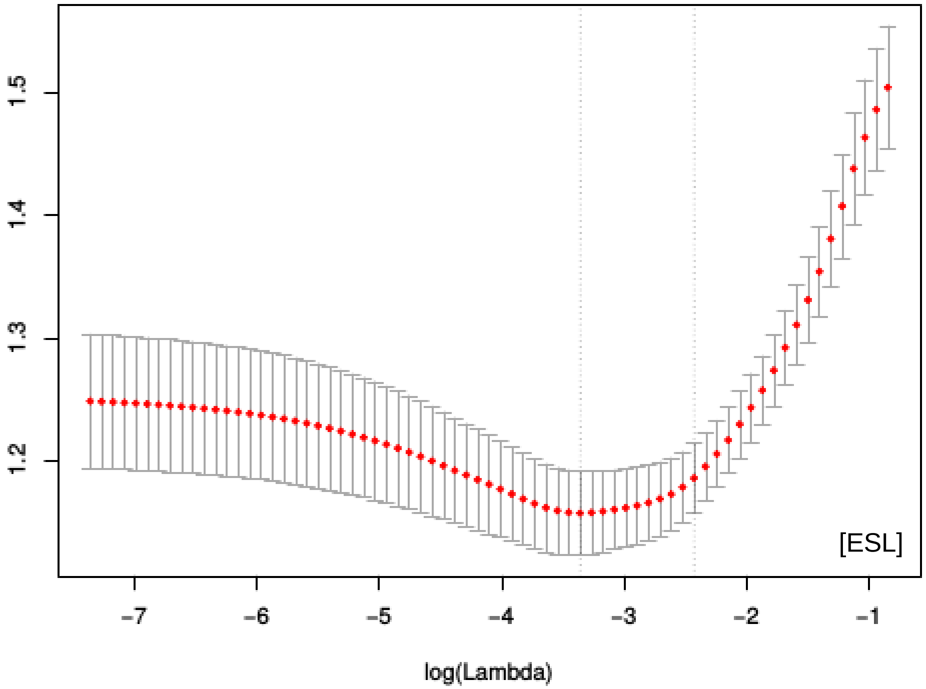
Choose the set of parameters **w** that give the smallest CV.

Drawback: can be very **time consuming** ...

S-fold cross-validation

Divide data in S groups, use S-1 for training and test on left-over group

Rinse and repeat S times



Cross-validation curve