Introduction to Machine Learning
Basic concepts, regression

ml@cezeaux - 18/05/2021

Julien Donini - LPC/Université Clermont Auvergne

Outline of the ML course

Lectures

I: Introduction to Machine Learning (18/05)

Hands-on: linear regression, regularization

II: Introduction to Neural Networks (June)

Hands-on: anomaly detection (fraud detection)

Practice sessions

Code on Git:

https://github.com/judonini/MLcourses

Hands-on: MLcourses/exercices/2020

Outline

Machine learning:

- Basic concepts: regression and classification

Linear regression

- Non linear data and basis functions

Model optimization

- How to control and optimize your model

Model minimization

- Gradient descent

Basic concepts

Machine Learning

Based on mathematics, statistics and algorithmics + computer power

- Determine complex **models** from data
- Used for **classification, inference, generation, ...**

Machine Learning is not recent

- Artificial **Neural Network** (theory 40’s, first functional networks 60’s)
- Decision **Trees** (~80’s)

Renaissance of the field since ~10 years

- **Deep Learning**
- **G**raphics **P**rocessing **U**nits for fast and scalable calculations
- New **recent** algorithms: GAN (2014), Adam minimization (2014), ...

Data

weights

Output

$$y(x,\mathbf{w}) = t$$

Target

Data

Output

weights

$$y(x,\color{blue}{w}) = \color{blue}{t}$$

Target

Examples

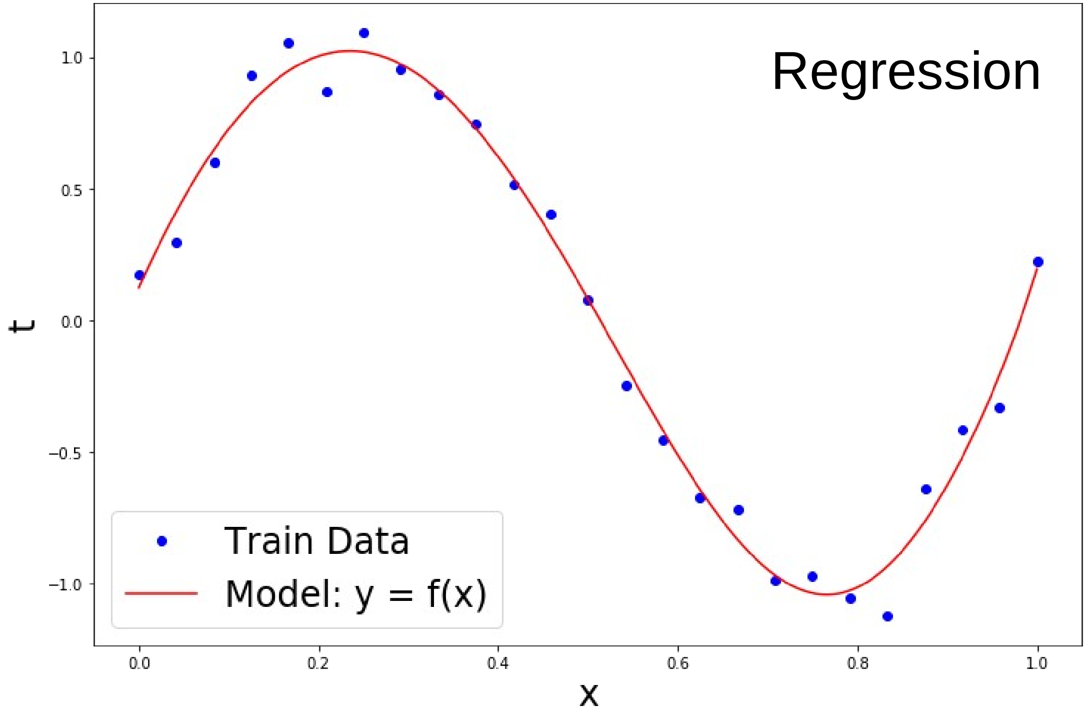
- **X** = {age, year, education, …} → **t**: income
- **X** = {image pixel values} → **t**: face recognition
- **X** = {list of words} → **t**: spam detection
- **X** = {E, p, …} → **t**: particle detection

...

Data
(features **x**)

Algorithm
y = f(x)

Objective
(target **t**)



Data

(features **x**)

Algorithm

y = f(x)

Objective

(target **t**)

Classification

Dog

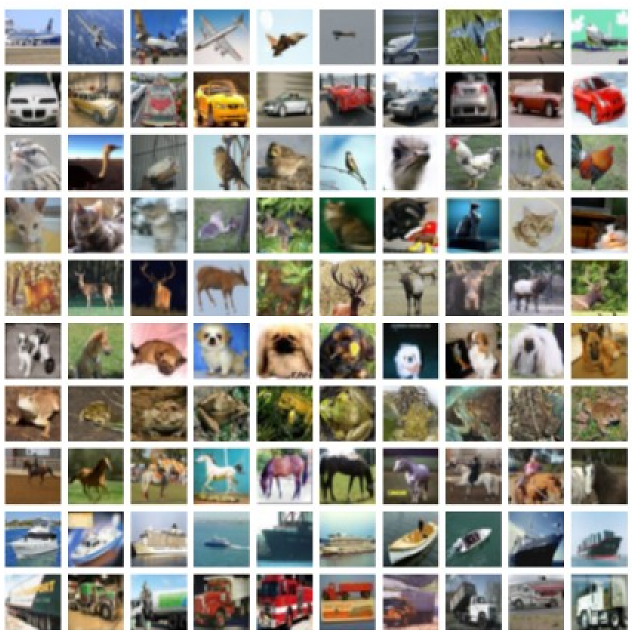
```
[[[ 7.4280e-02,  1.4022e-01, -2.2258e-02,  ..., -2.0172e-01,
    1.6240e-01,  5.5748e-02],
  [-1.1771e-02, -1.1327e-01,  3.0360e-01,  ...,  4.6299e-01,
    3.4765e-02,  2.2633e-02],
  [ 2.2252e-02,  2.1568e-01, -3.5726e-01,  ..., -7.4589e-02,
    7.0776e-02,  1.3573e-01],
  ...,
  [ 1.1035e-01, -2.4609e-01,  1.9962e-01,  ...,  2.4133e-01,
   -2.1069e-01,  1.9942e-01],
  [ 2.9337e-02,  2.4997e-01,  1.0341e-02,  ..., -3.1368e-01,
   -1.6878e-01, -1.4741e-02],
  [ 4.4006e-02,  5.1292e-02,  5.0462e-02,  ..., -8.1194e-02,
    1.6043e-01, -5.7106e-03]]],
```

Machine Learning Basics

Data
(features **x**) →

Algorithm
y = f(x)

→ **Objective**
(target **t**)



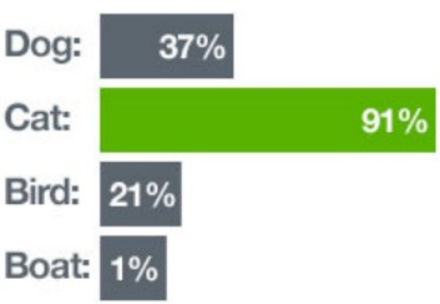
Training →

airplane
automobile
bird
cat
deer
dog
frog
horse
ship
truck

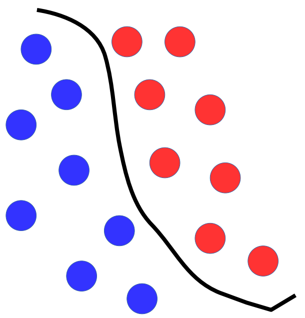
Machine Learning Basics

Data
(features **x**) → **Algorithm** **y = f(x)** → **Objective** (target **t**)

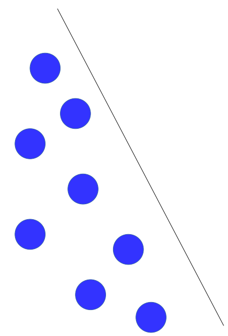
Test →



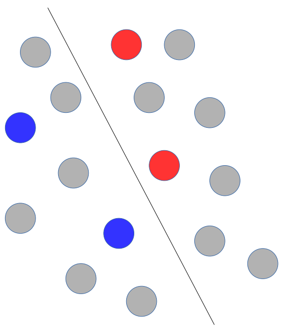
Supervised (labels are known)



Unsupervised (no labels)



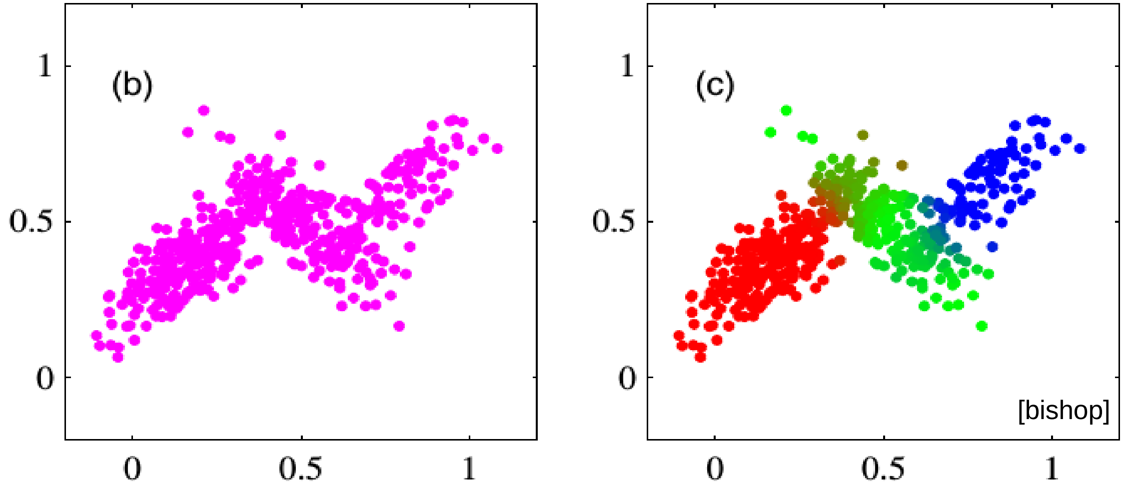
Semi-supervised (few labels)



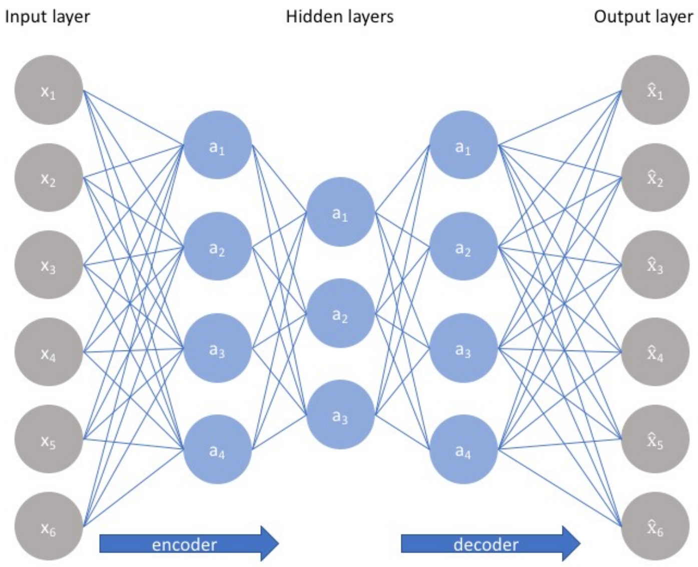
- labels of class 1
- labels of class 2
- unknown class
- decision boundary

Unsupervised learning = no labels

Clustering

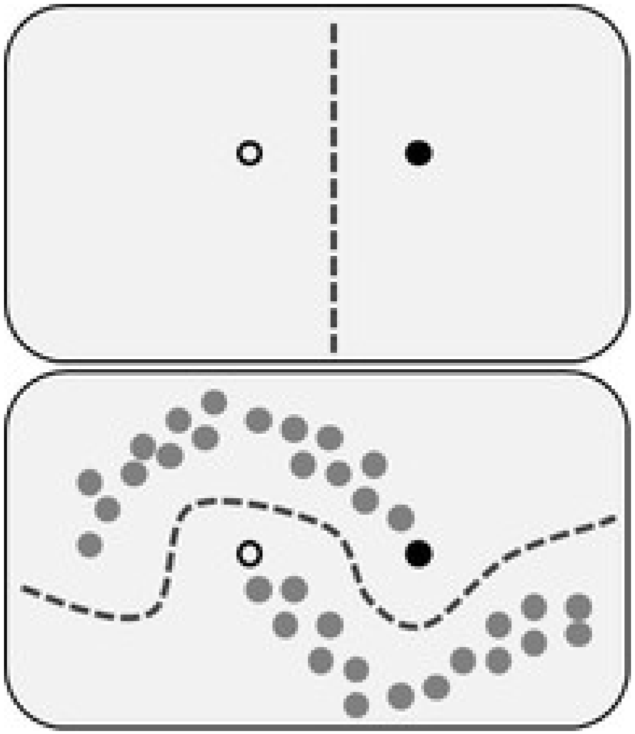


Dimensionality reduction



Semi-supervised learning

Semi-supervised learning = unlabelled data + few labels

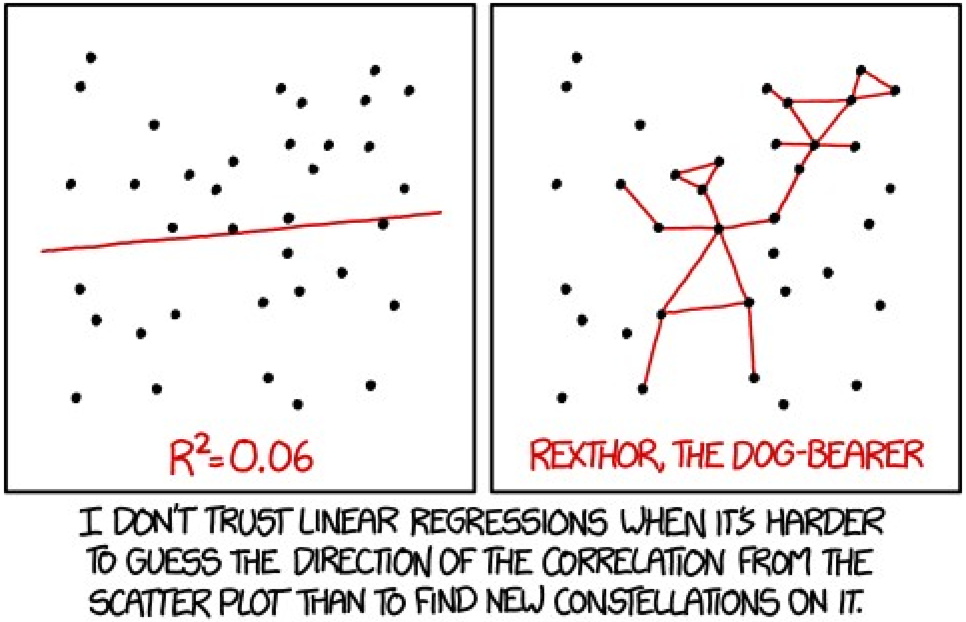


[wikipedia]

Example of the influence of unlabelled data in semi-supervised learning.

The unlabelled data (grey dots) influence the separation of the two classes (decision surface)

Linear regression

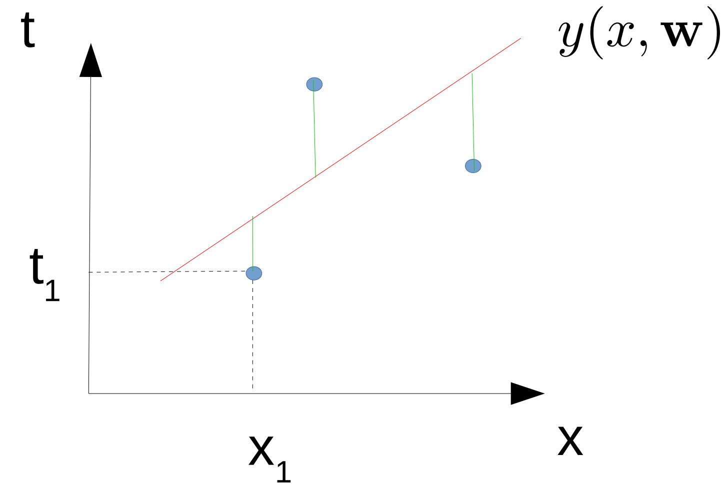


[xkcd.com]

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**

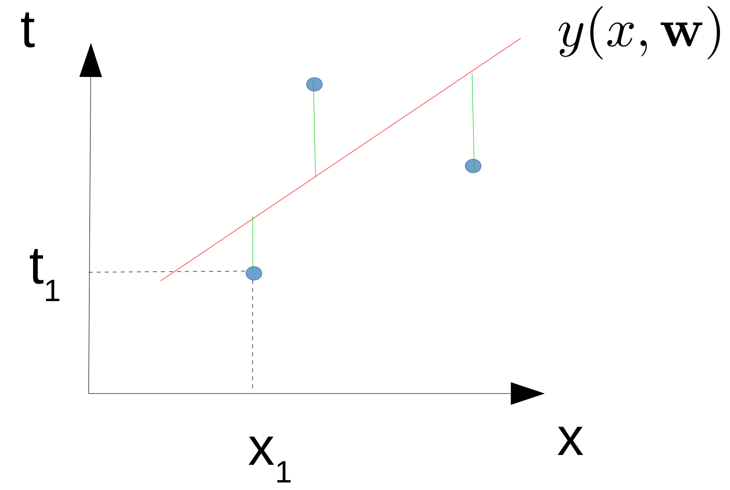
Common choice: sum of **square distance** between function and target:

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

Simple case: 1 dimensional data

Training dataset

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



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 !)

$$\boxed{\begin{gathered} E(w_0, w_1) = \sum_{i=1}^{N} \{y(x_i; w_0, w_1) - t_i\}^2 \\ \begin{cases} \dfrac{\partial E(w_0,w_1)}{\partial w_0} = 0 \\ \dfrac{\partial E(w_0,w_1)}{\partial w_1} = 0 \end{cases} \Leftrightarrow \begin{cases} w_1 = \dfrac{\mathrm{cov}(x,t)}{\mathrm{var}(x)} = r\,\dfrac{\sigma(t)}{\sigma(x)} \\ w_0 = \bar{t} - r\,\dfrac{\sigma(t)}{\sigma(x)}\,\bar{x} \end{cases} \end{gathered}}$$

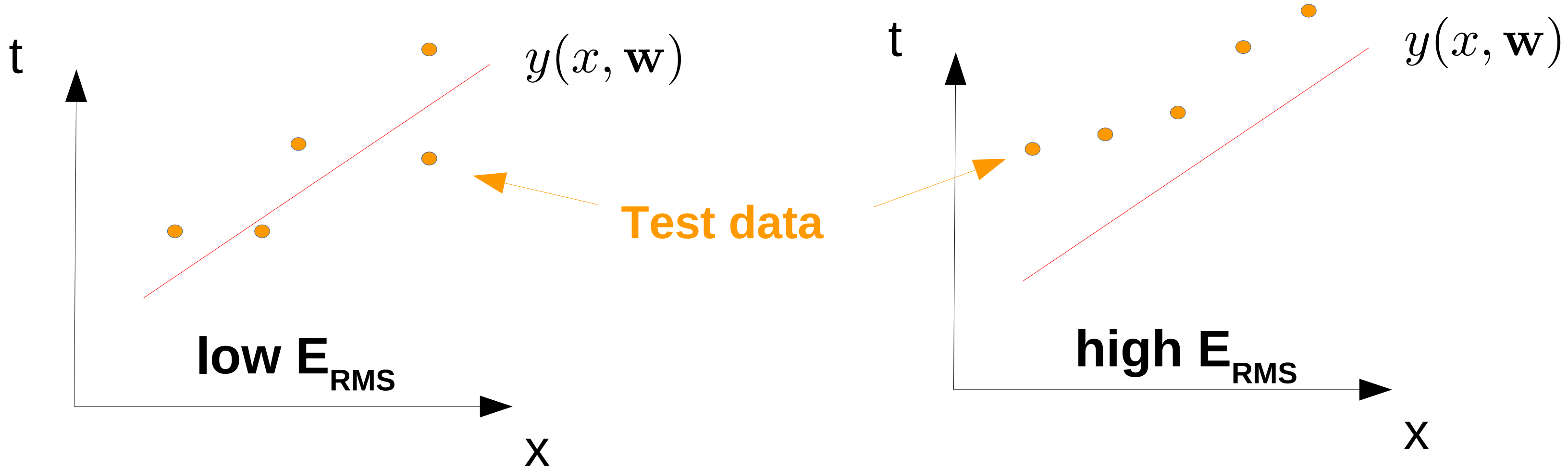
(r: correlation factor between x and t)

Training and testing

- **Training**: use dataset to determine **weights $\mathbf{w}_0$ and $\mathbf{w}_1$**
- **Testing**: check compatibility of y(x,**w**) on a new dataset

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

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



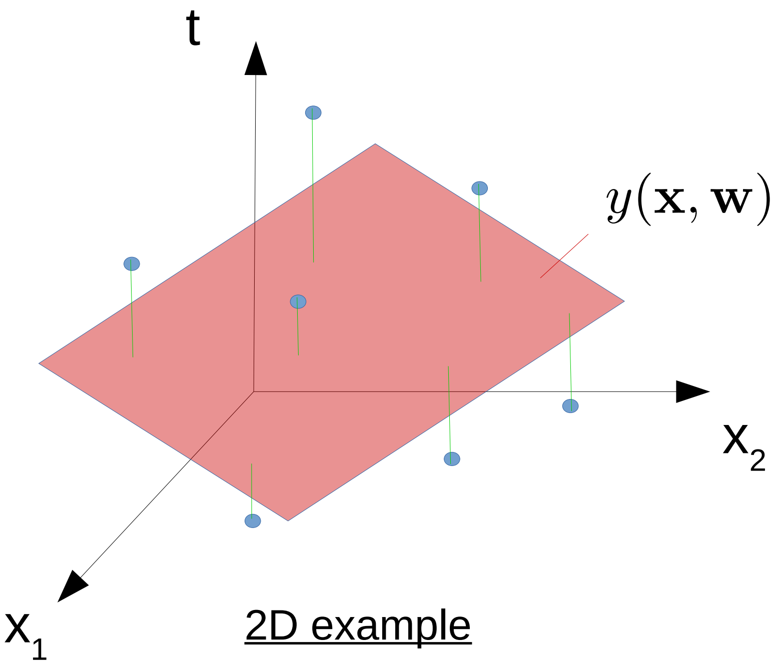
Generalization: multidimensional data

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 $t = \{t_1, \ldots, t_N\}$



2D example

Fit function: multidimensional plane

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

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

bias term

Generalization: multidimensional data

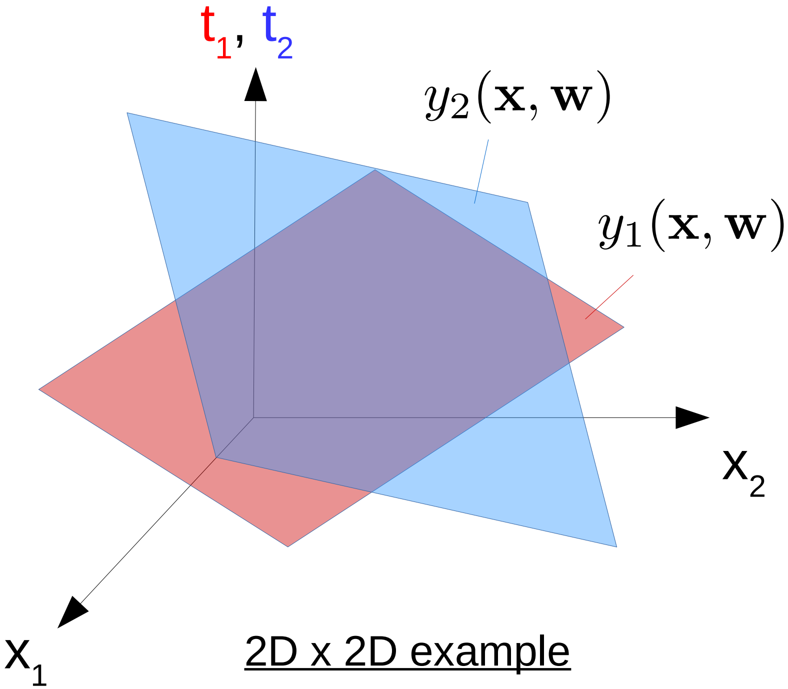
Dataset (p x q data)

- **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

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

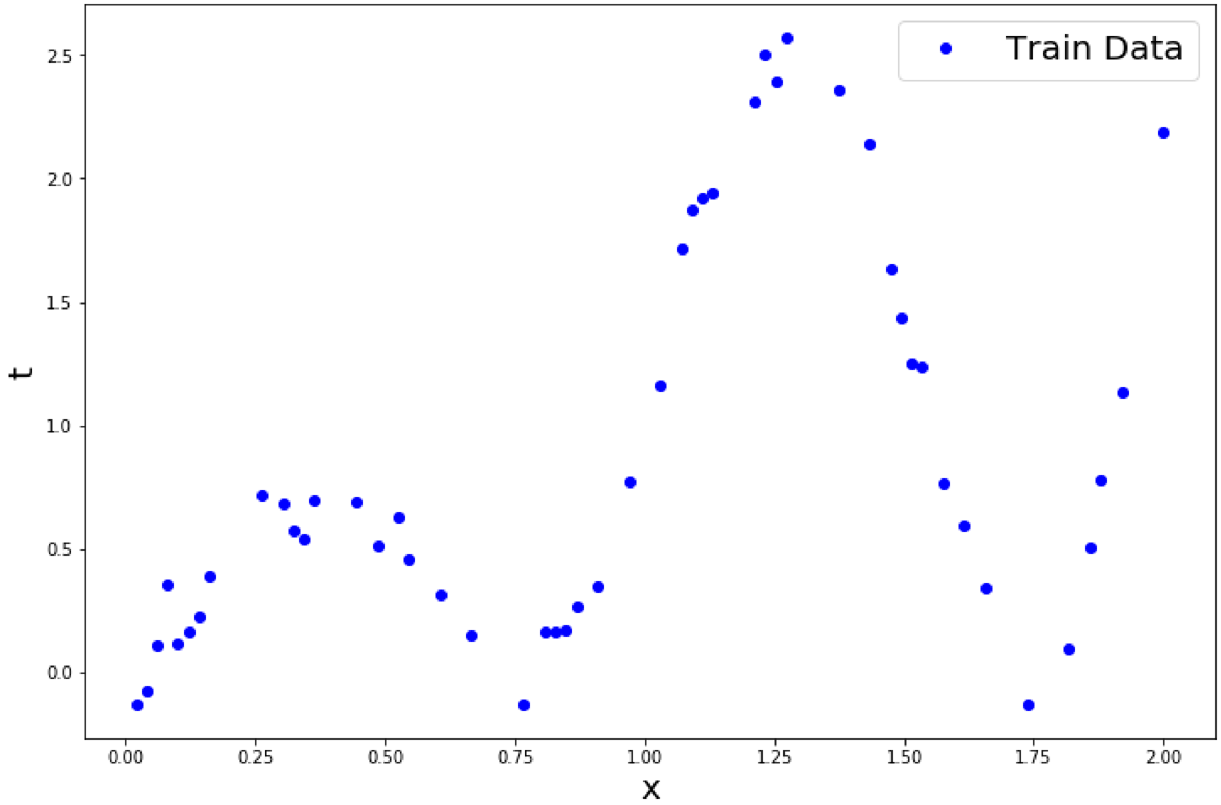


2D x 2D example

Fit functions:

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

What if your data is not linear ?



→ use basis functions

Linear basis function models

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.

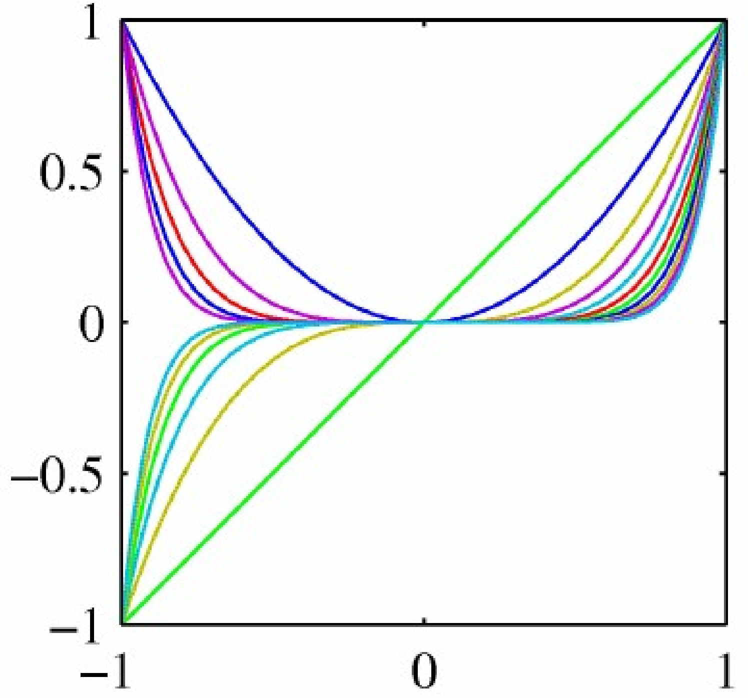
Linear basis function models

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

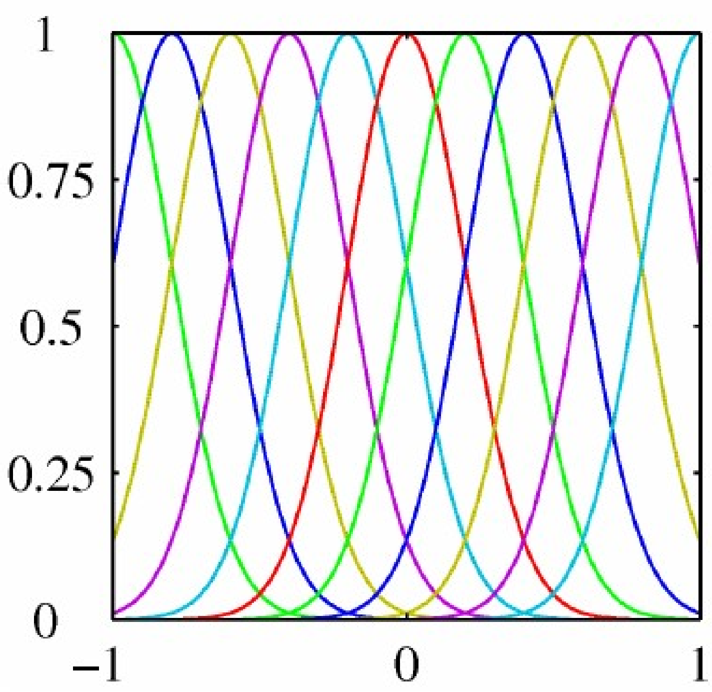


[bishop]

Gaussian basis functions (1D)

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

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



[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

Sigmoidal basis functions (1D)

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

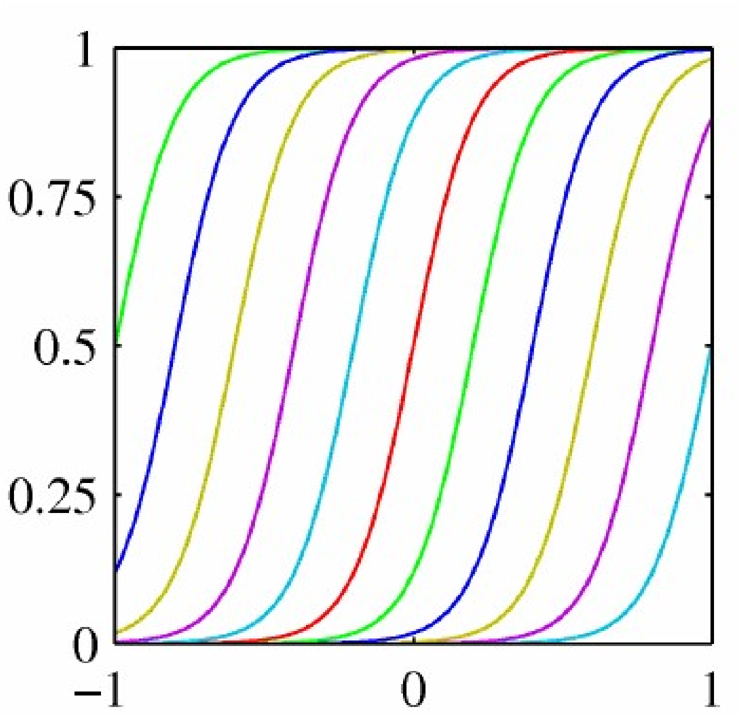
with

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

Parameters:

μ_j (location) and s (slope)

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

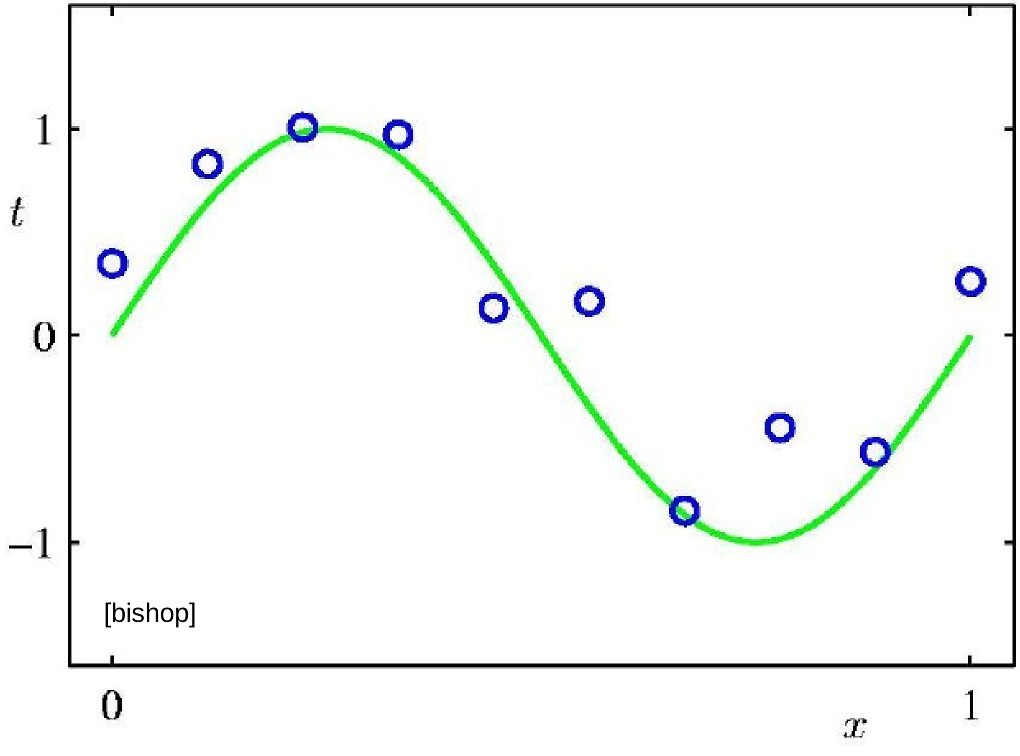


[bishop]

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)

Fit function

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

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

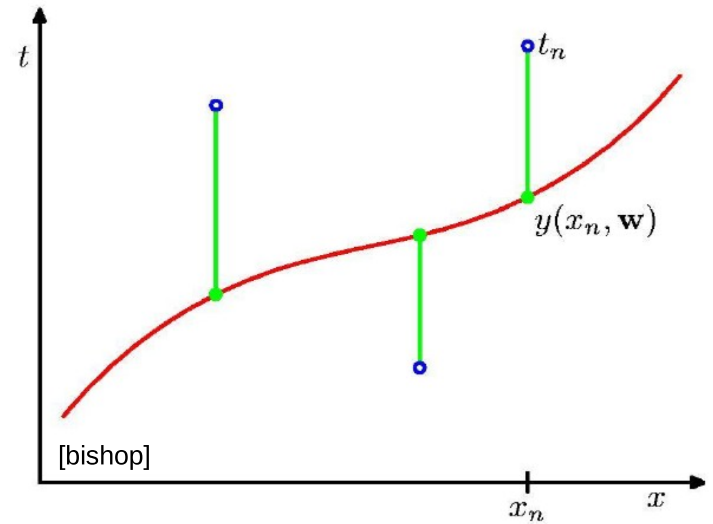
- Non-linear function of x, but linear function of **w** → **linear model**
- Values of coefficient obtained by **minimizing** an **error function**
- **Sum of the square of the errors** E(**w**)

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

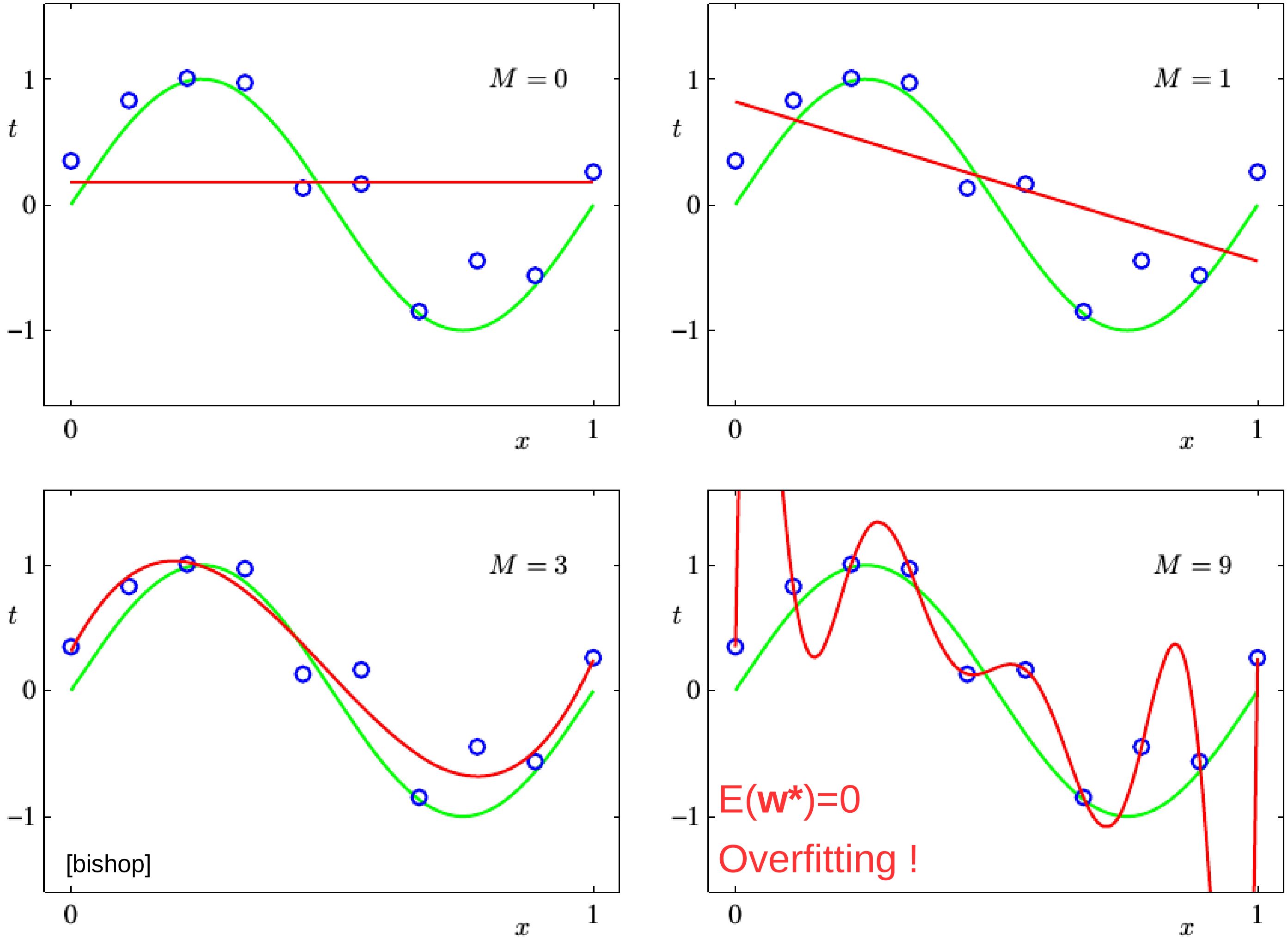
Minimization

Fitted weights **w***

E(**w***)



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

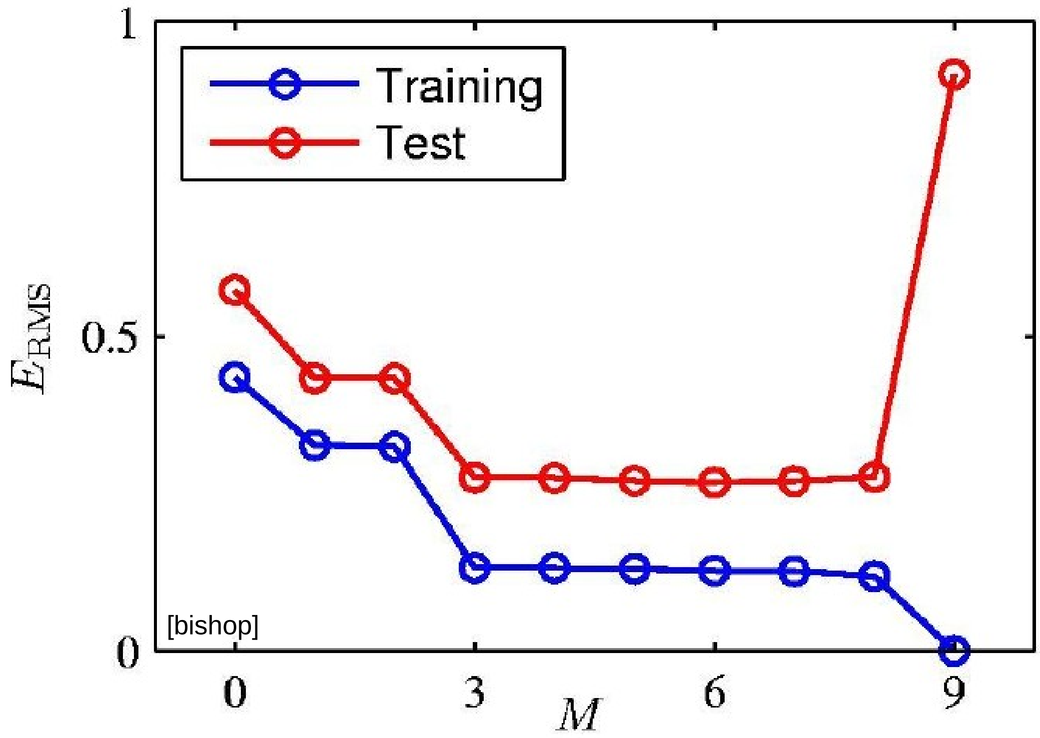


It is instructive to look at the **fitted weights** for various cases: when M increases the coefficient become **fine tuned** to data by developing large positive and negative values.

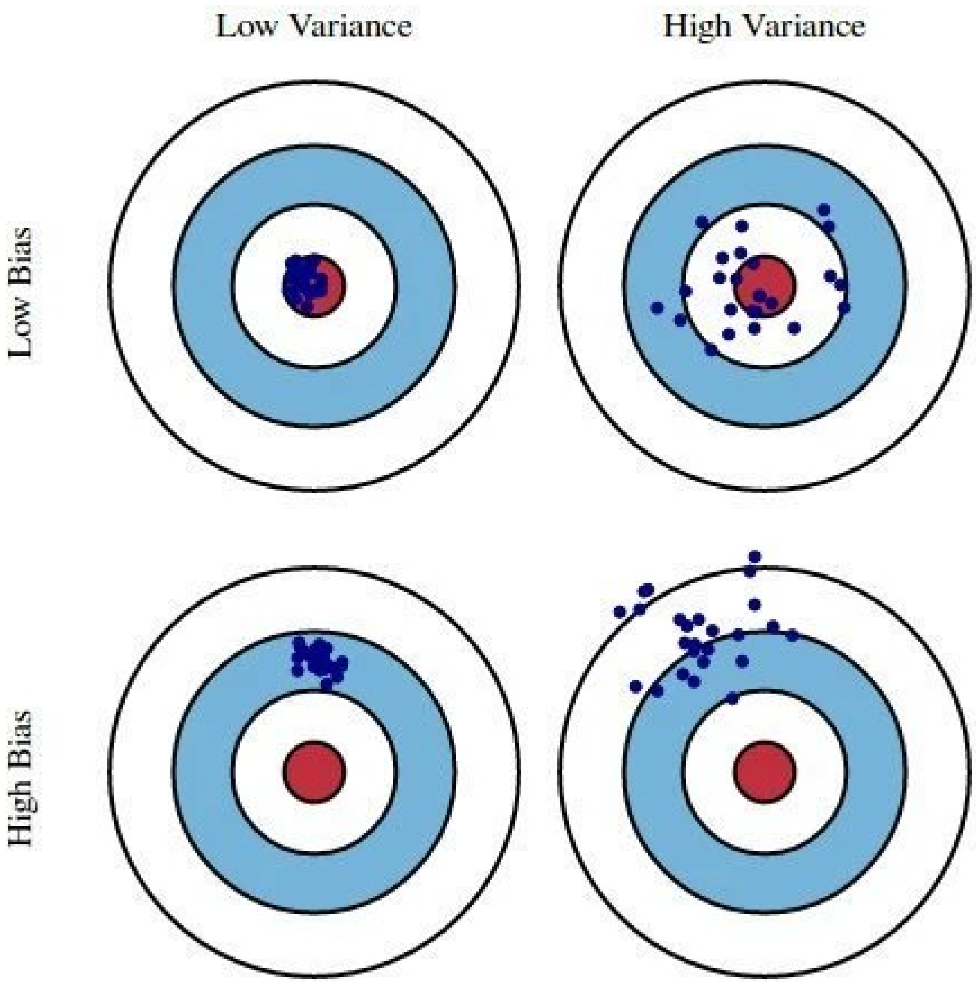
	$M=0$	$M=1$	$M=3$	$M=9$
$w_0^\star$	0.19	0.82	0.31	0.35
$w_1^\star$		-1.27	7.99	232.37
$w_2^\star$			-25.43	-5321.83
$w_3^\star$			17.37	48568.31
$w_4^\star$				-231639.30
$w_5^\star$				640042.26
$w_6^\star$				-1061800.52
$w_7^\star$				1042400.18
$w_8^\star$				-557682.99
$w_9^\star$				125201.43

Root mean squared error (RMS)

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



What is a good model ?



[figure: kdnuggets.com]

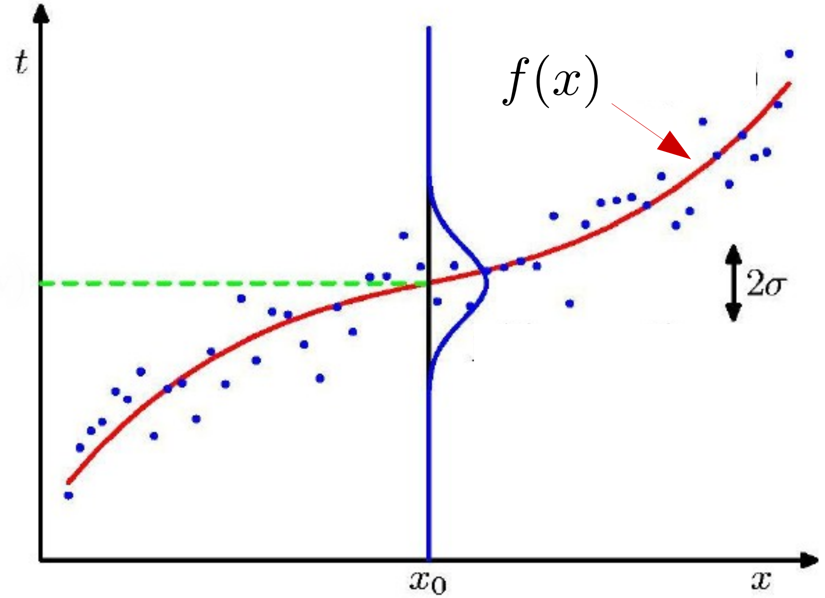
The Bias-Variance decomposition

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 approximates true function f(x)**

The Bias-Variance decomposition (*)

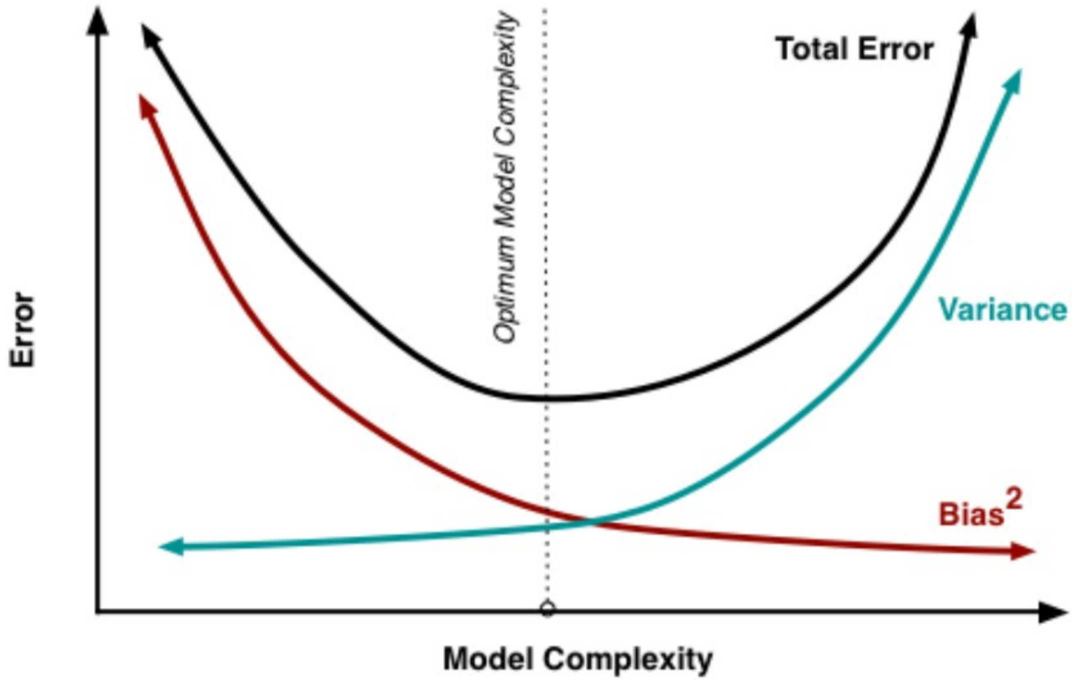
As before we determine y(x) by **minimizing:** $\sum_{i=1}^{N} \{y(x_i, \mathbf{w}) - t_i\}^2$
over the **training** dataset

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

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

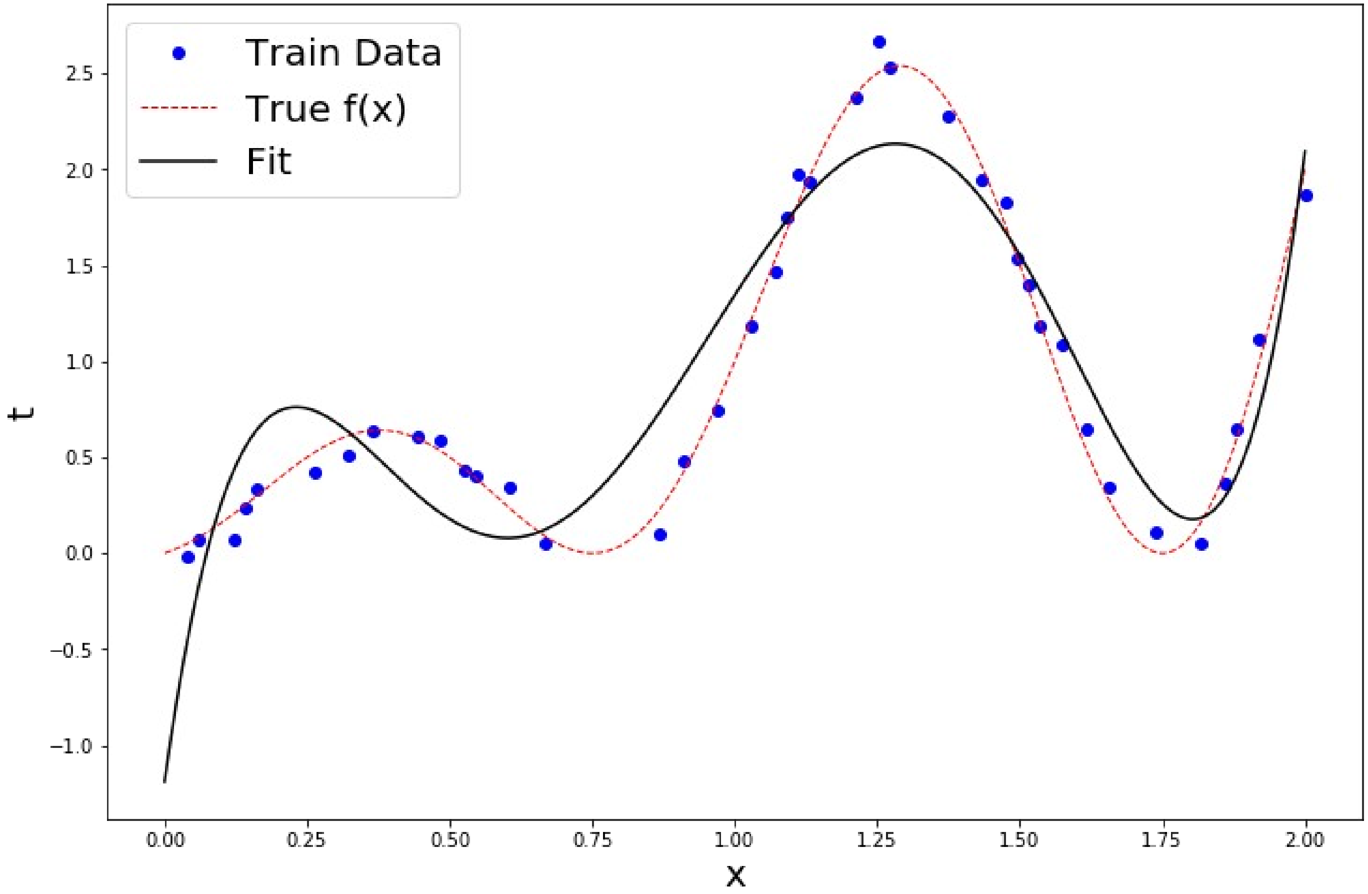
The Bias-Variance decomposition



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)

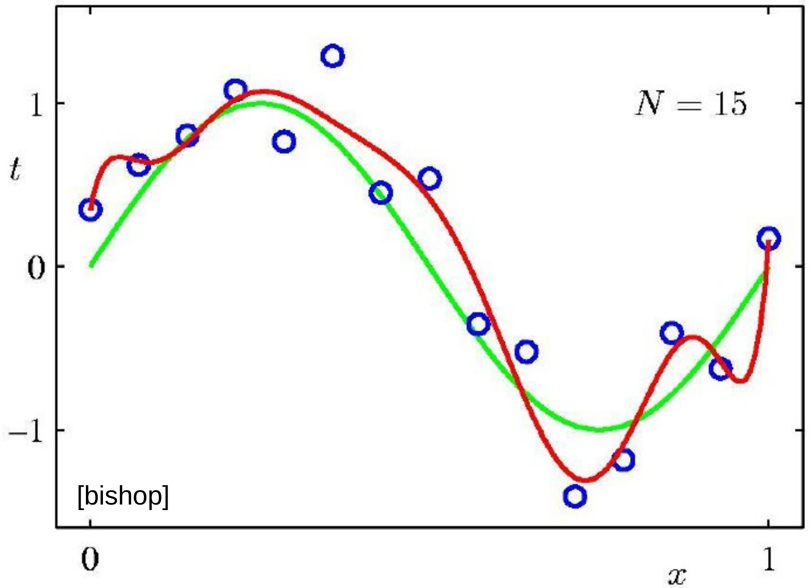
What if your model completely fails ?



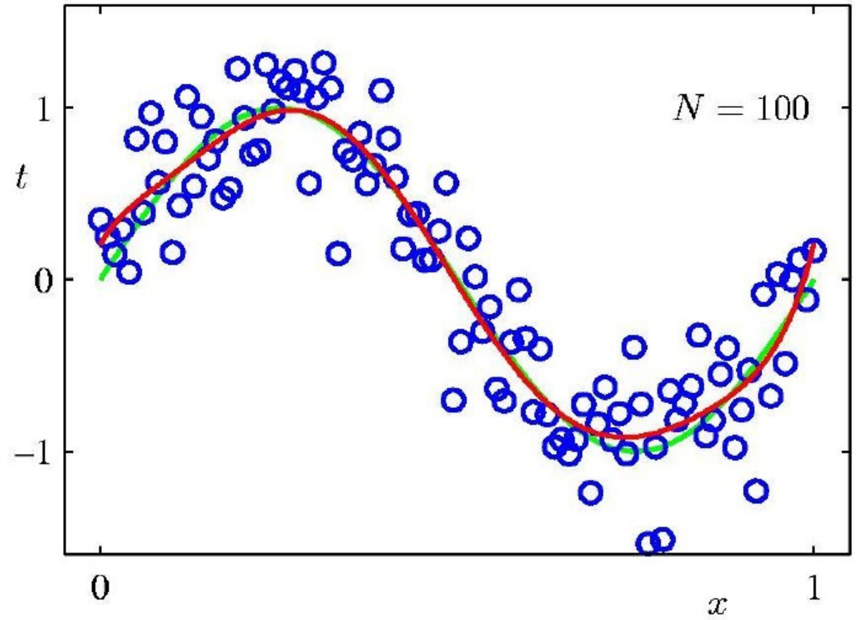
→ try to regularize your model

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.

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

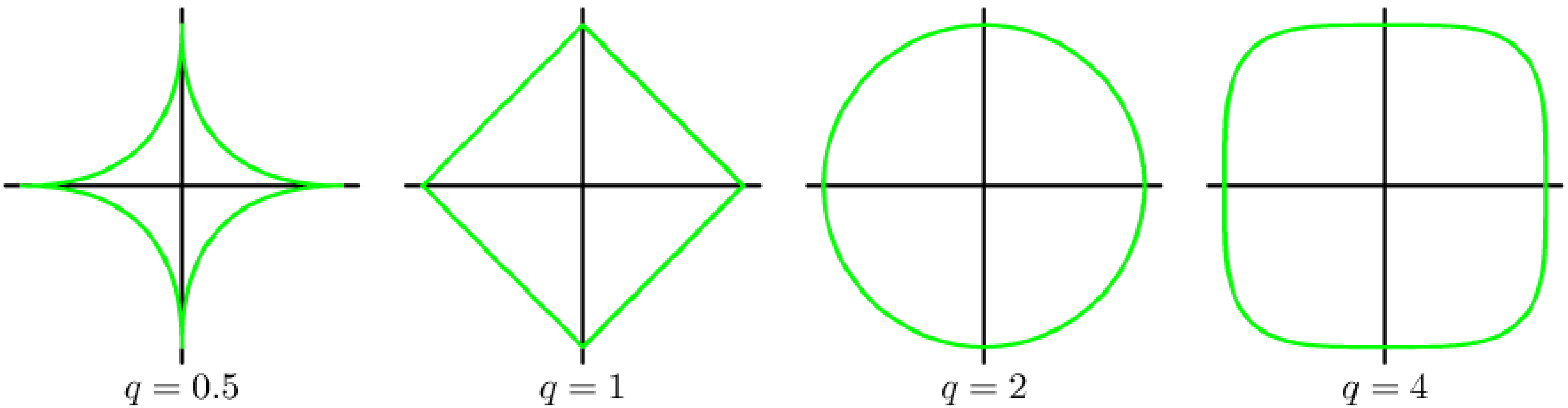
Other choices

- **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

General regularization term is of the form:

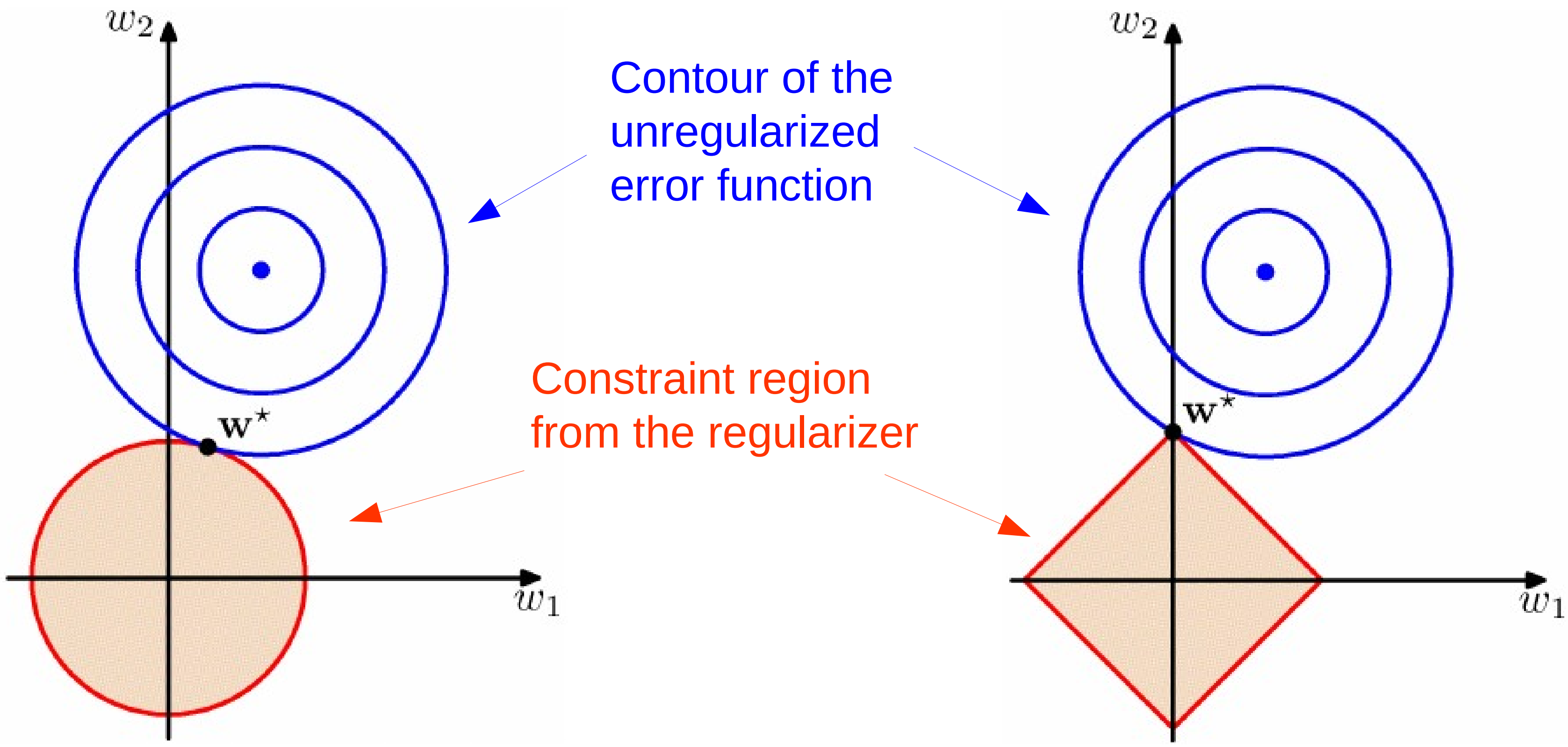
$$\tilde{E}(\mathbf{w}) = \sum_{i=1}^{N} \{y(x_i, \mathbf{w}) - t_i\}^2 + \boxed{\lambda \sum_{j=1}^{M} |w_j|^q}$$

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)



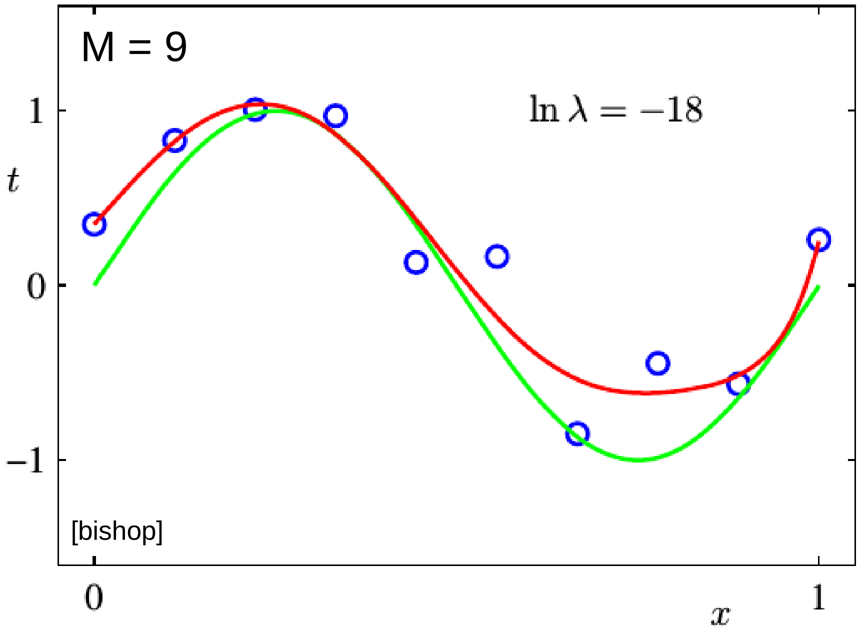
Ridge
(L2 norm)

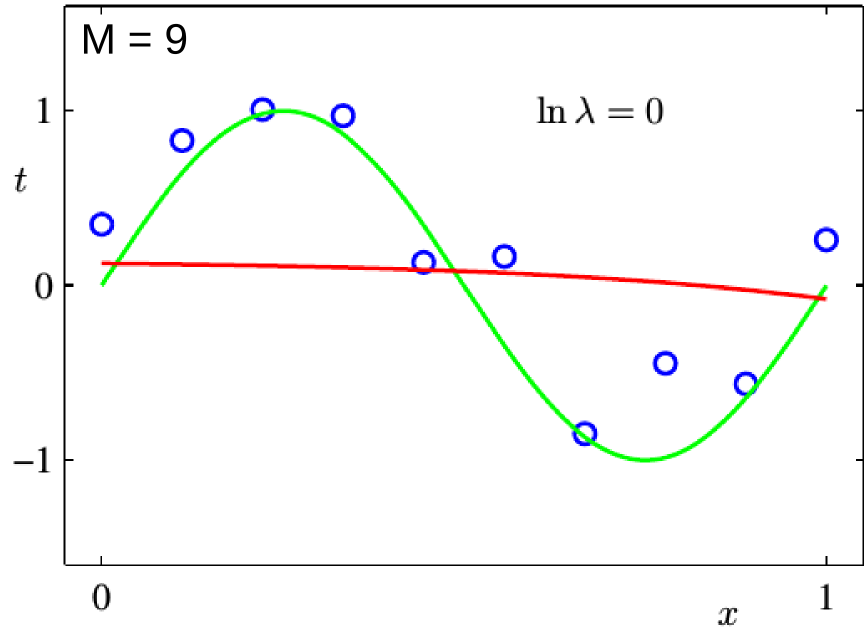
Lasso
(L1 norm)

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

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

Regularization





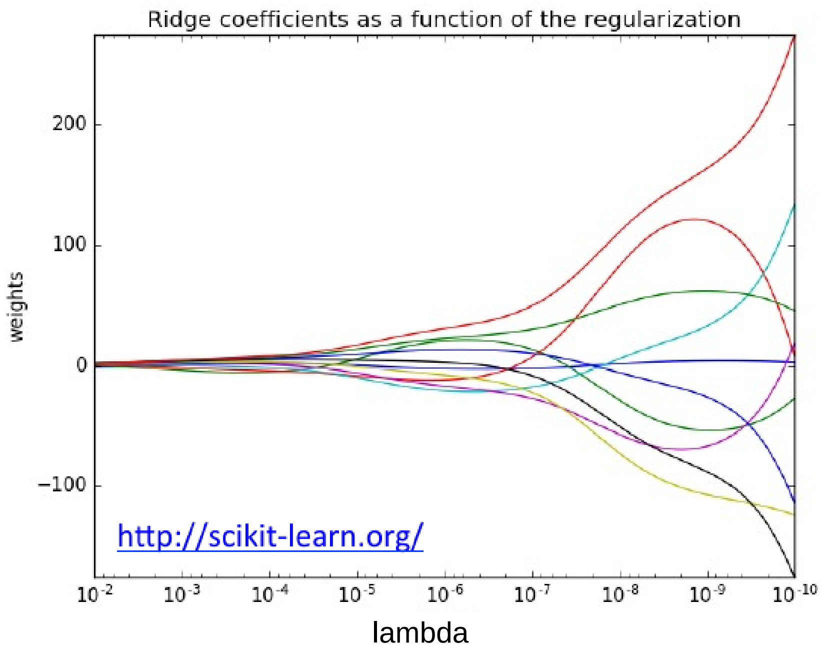
	$\ln \lambda = -\infty$	$\ln \lambda = -18$	$\ln \lambda = 0$
$w_0^\star$	0.35	0.35	0.13
$w_1^\star$	232.37	4.74	-0.05
$w_2^\star$	-5321.83	-0.77	-0.06
$w_3^\star$	48568.31	-31.97	-0.05
$w_4^\star$	-231639.30	-3.89	-0.03
$w_5^\star$	640042.26	55.28	-0.02
$w_6^\star$	-1061800.52	41.32	-0.01
$w_7^\star$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

Regularization

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

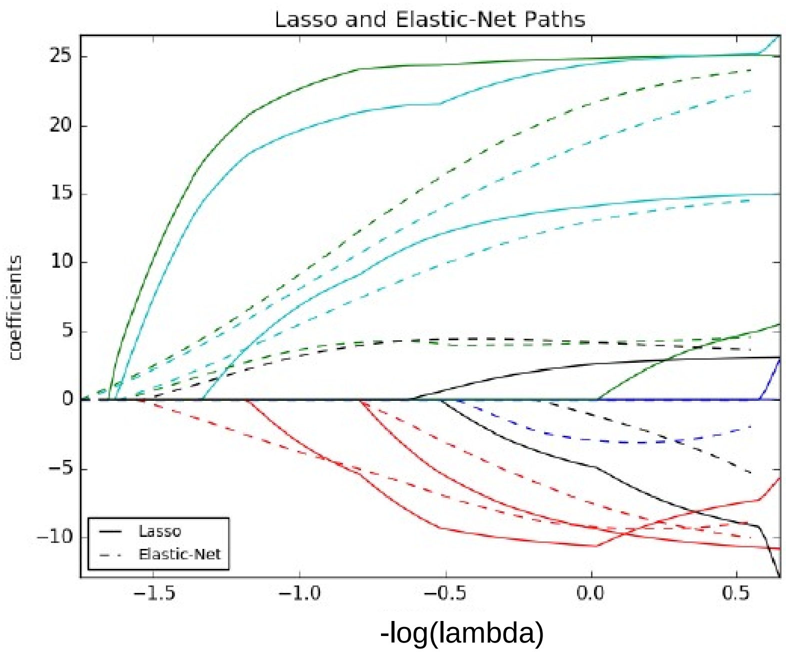


More constraints

Less constraints

Affects value of coefficients (shrinkage)

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



More constraints

Less constraints

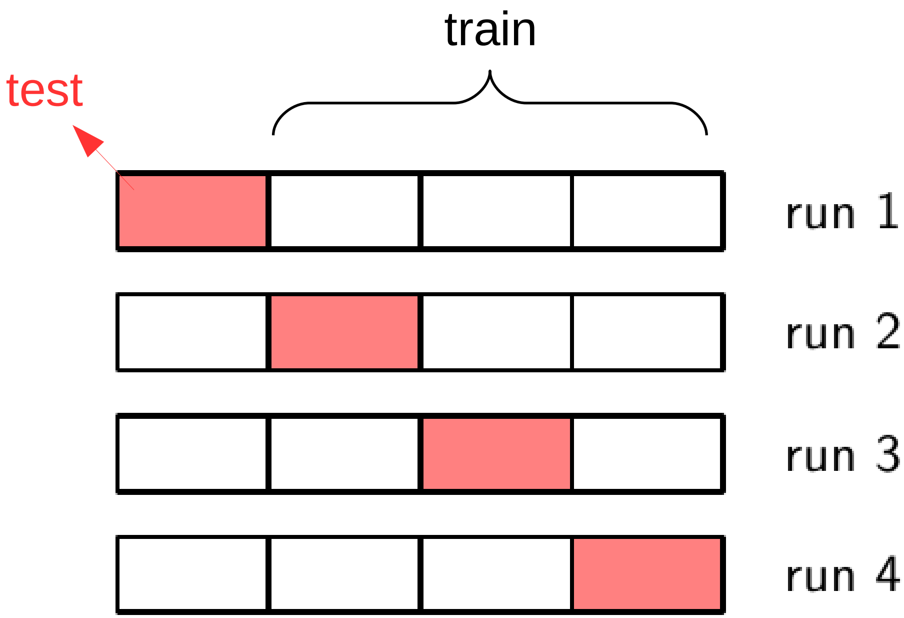
Affects number of coefficients (sparsity)

Model selection

K-fold cross-validation

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

Rinse and repeat K 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: $CV = \frac{1}{K} \sum E_i$

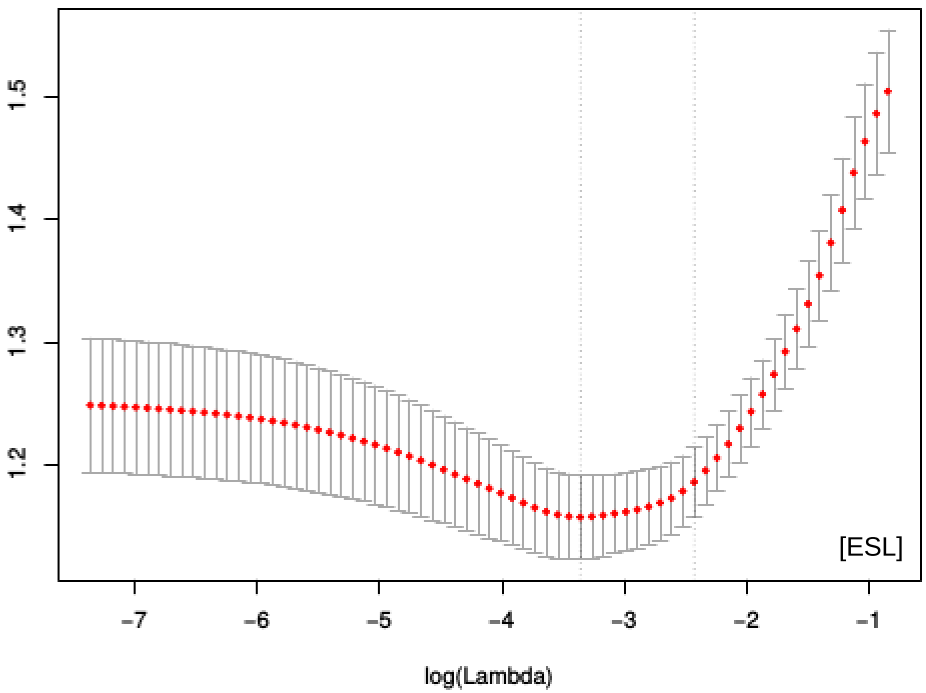
Choose the set of hyper-parameters (ex λ) that give the smallest CV.

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

K-fold cross-validation

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

Rinse and repeat K times



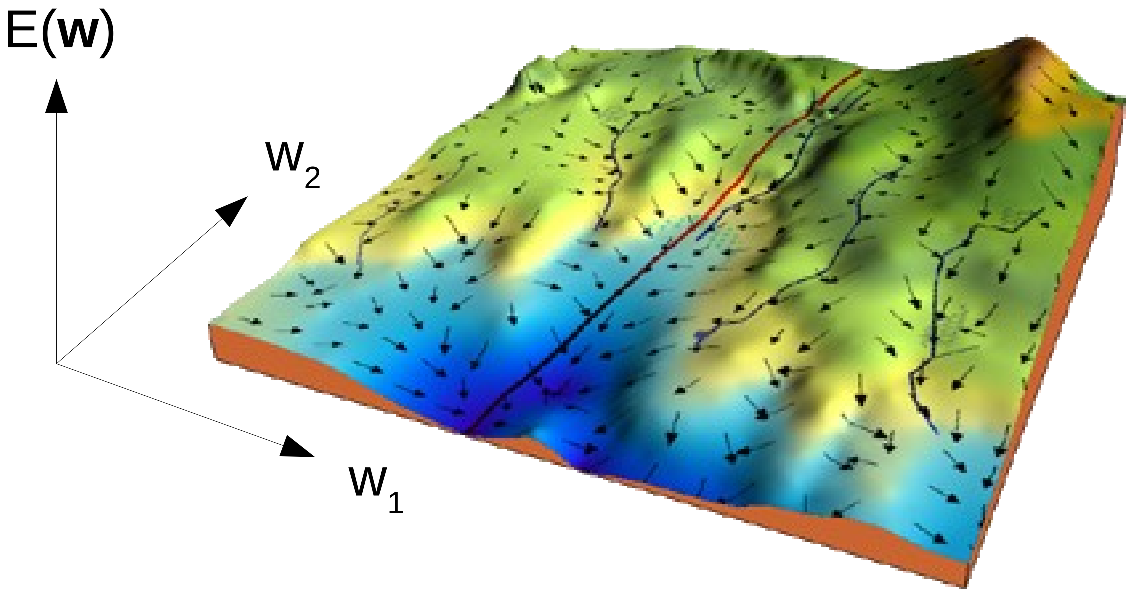
Cross-validation curve

Gradient descent

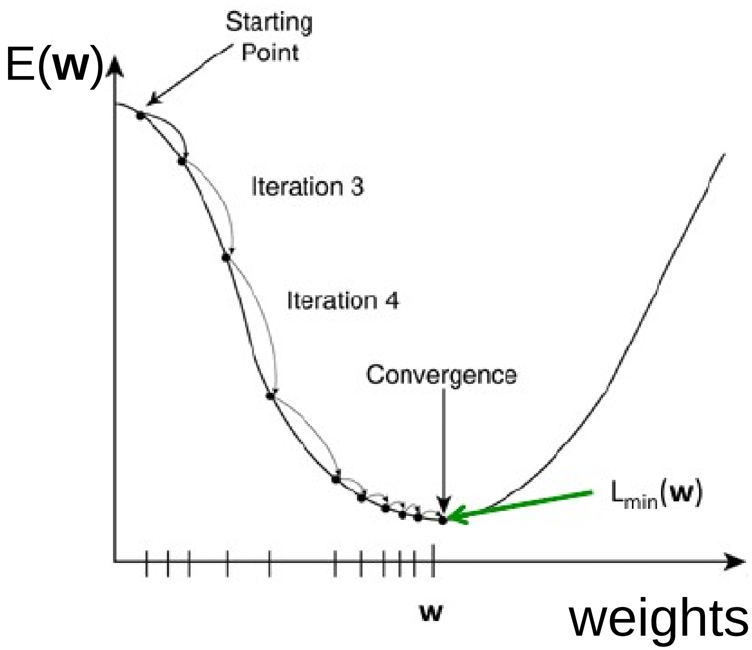
How can we **minimize** the error function for complex cases (ex: when there is no analytic solution) ?

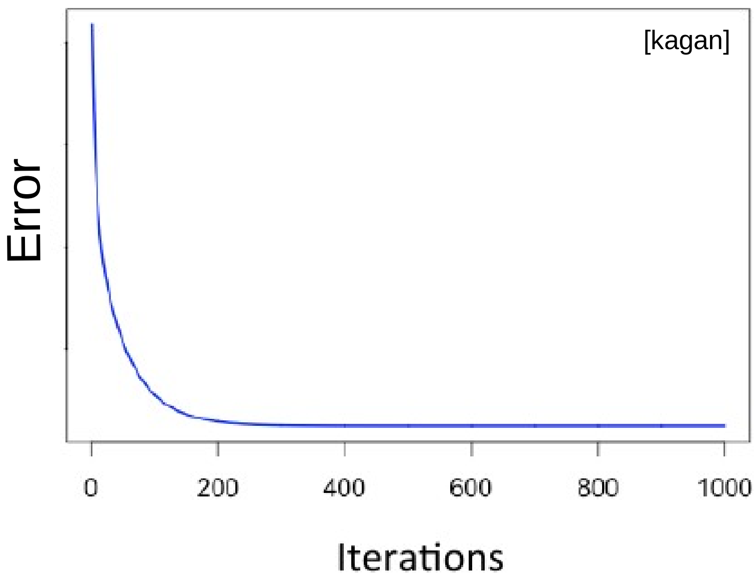
→ **Solution: Gradient descent**

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



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

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**:

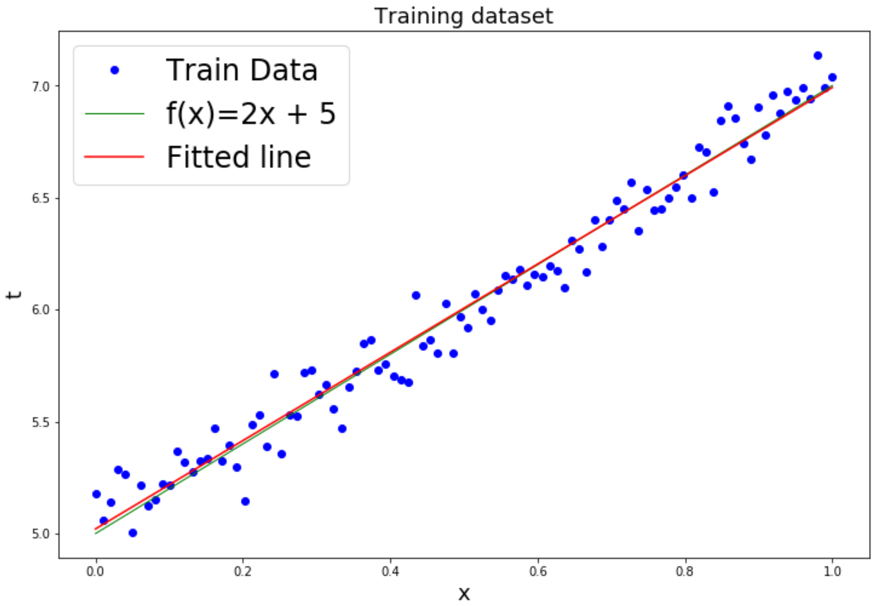
$$\boxed{\begin{aligned} \mathrm{w}_0^{(k)} &\rightarrow w_0^{(k+1)} = w_0^{(k)} - \frac{\partial E(w_0,w_1)}{\partial w_0} \times \eta \\ \mathrm{w}_1^{(k)} &\rightarrow w_1^{(k+1)} = w_1^{(k)} - \frac{\partial E(w_0,w_1)}{\partial w_1} \times \eta \end{aligned}}$$

k: iteration number

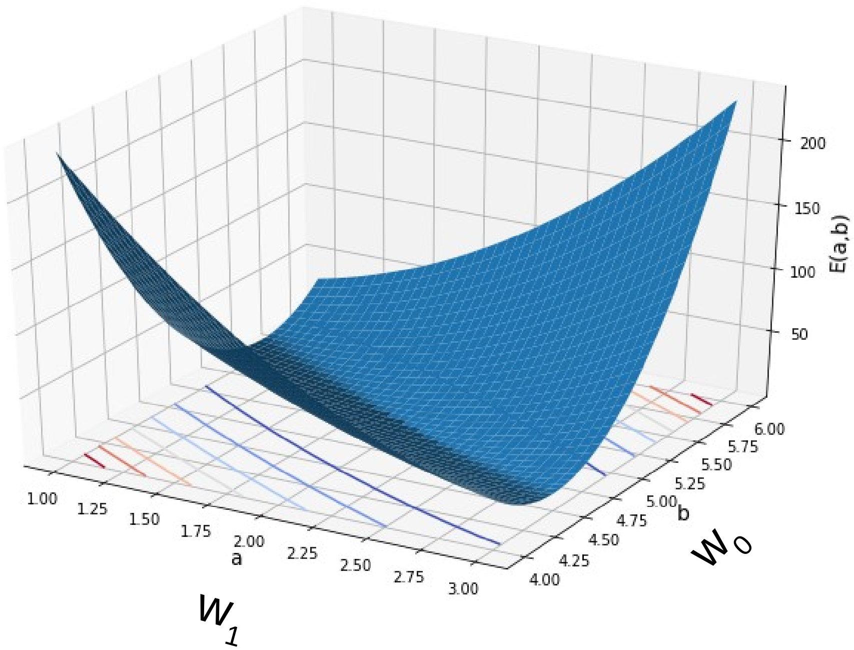
η: learning rate

Repeat until convergence

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

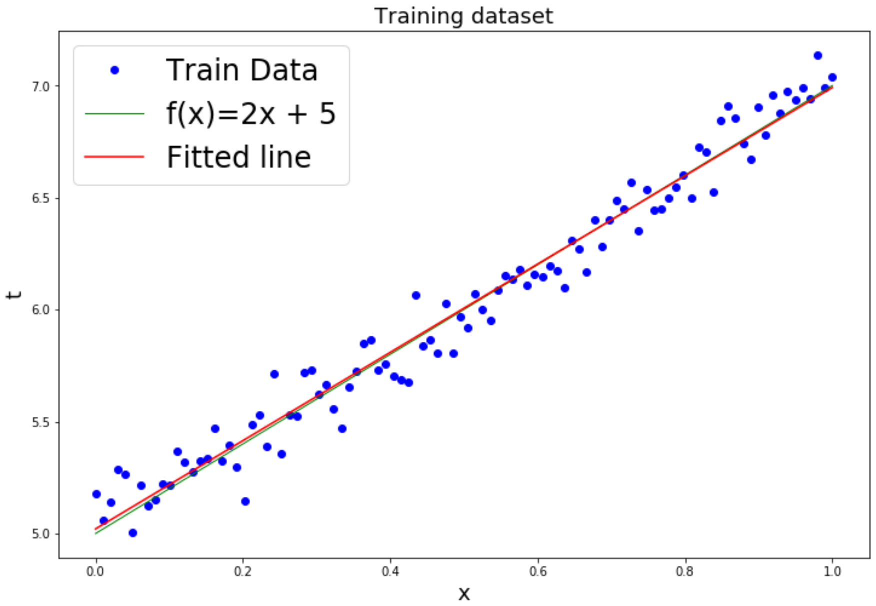


Error function

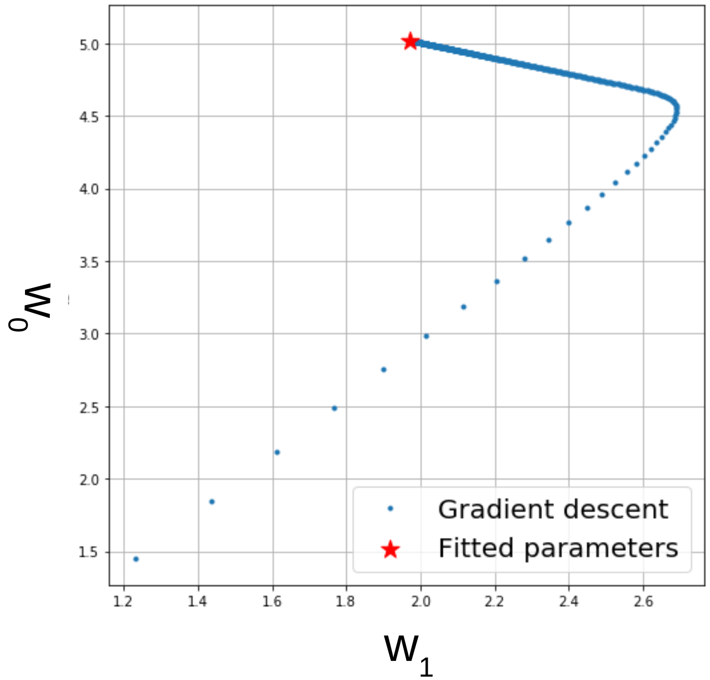


$$E(w_0, w_1) = \sum_{i=1}^{N} \{(w_0 + w_1 x_i) - t_i\}^2$$

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



Gradient descent



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

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

1000 iterations

learning rate η = 0.05

Stochastic gradient descent

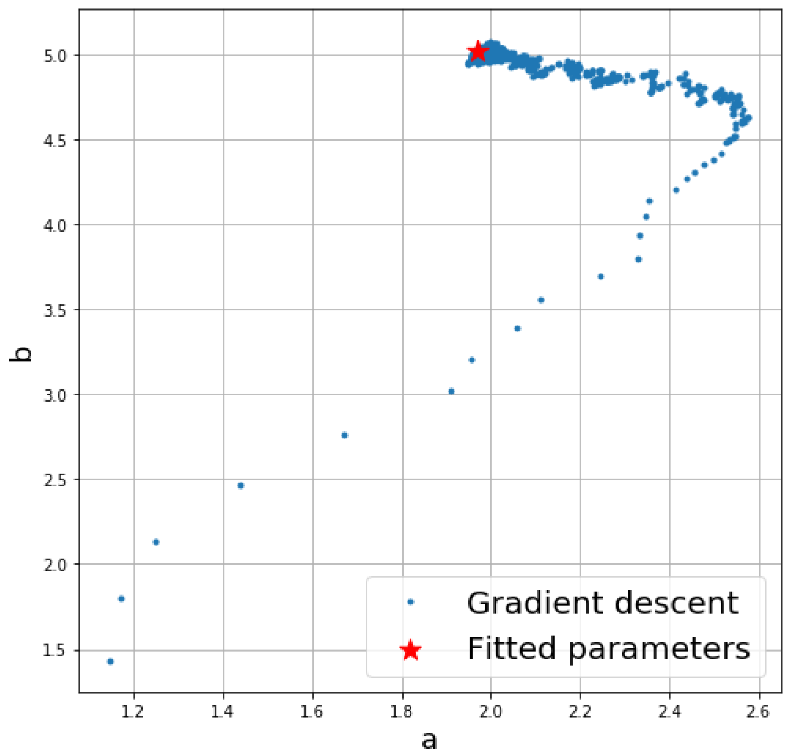
Gradient descent can be **computationally costly** for large N since the gradient is calculated over full training set.

→ **Solution: Stochastic gradient descent**

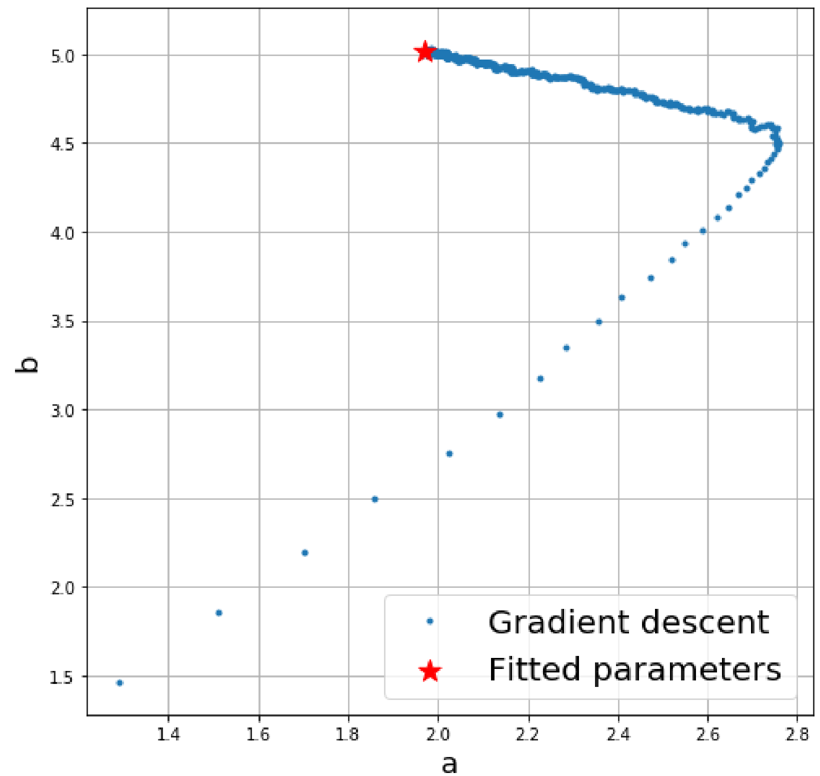
Compute gradient on a small **batch** of events (can be 1 event):

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

Stochastic gradient descent



Gradient calculated on 1 (random) event at each step



Gradient calculated on 10 (random) events at each step

Practise sessions

Git repository: https://github.com/judonini/MLcourses

Go to the Exercices/2020 folder

1) regression-boston-housing.ipynb

2) Gradient-descent.ipynb