An introduction to key concepts in Machine Learning
Part 1

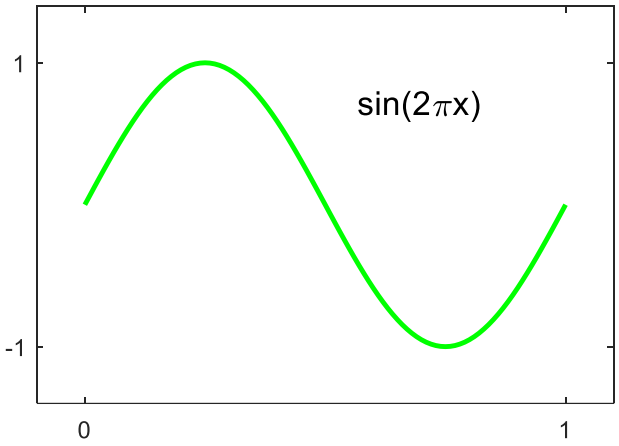
Alberto Paccanaro
EMAp – FGV

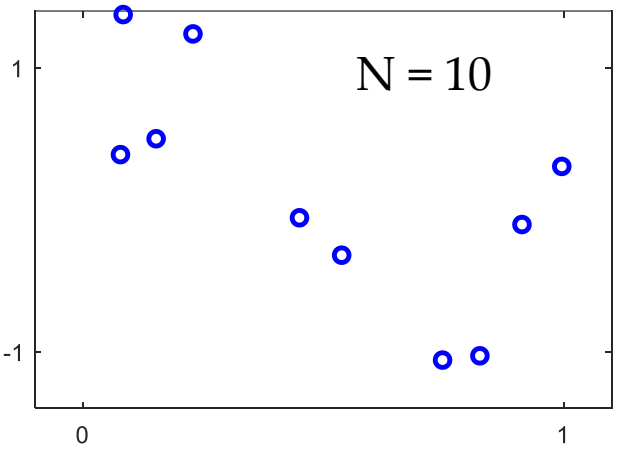
www.paccanarolab.org

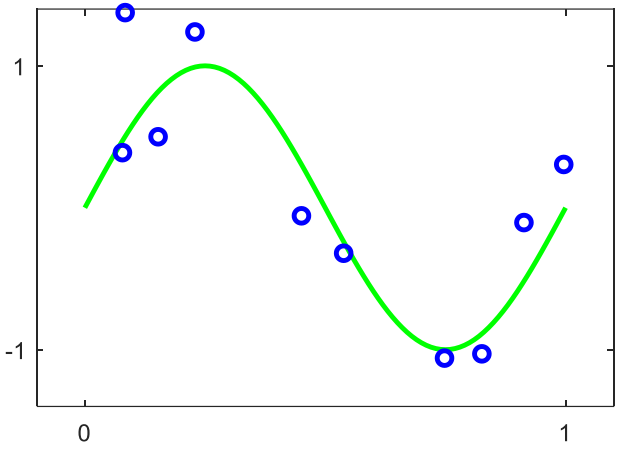
Some material and images are from (or adapted from):
C. Bishop, Pattern Recognition and Machine Learning, Springer, 2006
A. Geron, Hands-on Machine Learning with Scikit-Learn, Keras and TensorFlow, O'Reilly, 2020

The polynomial curve fitting example

A regression problem (supervised)







x	y
0.7482	-1.0592
0.4505	-0.0566
0.0838	1.3751
0.2290	1.2389
0.9133	-0.1043
0.1524	0.5003
0.8258	-1.0292
0.5383	-0.3203
0.9961	0.3052
0.0782	0.3883

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

$$\mathbf{t} \equiv (t_1, \ldots, t_N)^{\mathrm{T}}$$

} dataset

GOAL: use this training set in order to make predictions of the value $\hat{t}$ of the target variable for some new value $\hat{x}$ of the input variable.

Note: Uncertainty !

We will need probability theory + decision theory

So given the dataset:

(1) we choose a model

(2) we learn the parameters of the model

(3) we use the learned model $y(x,\mathbf{w}^*)$ to make predictions

A curve fitting approach

We use a polynomial function as our model:

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

where M is the order of the polynomial.

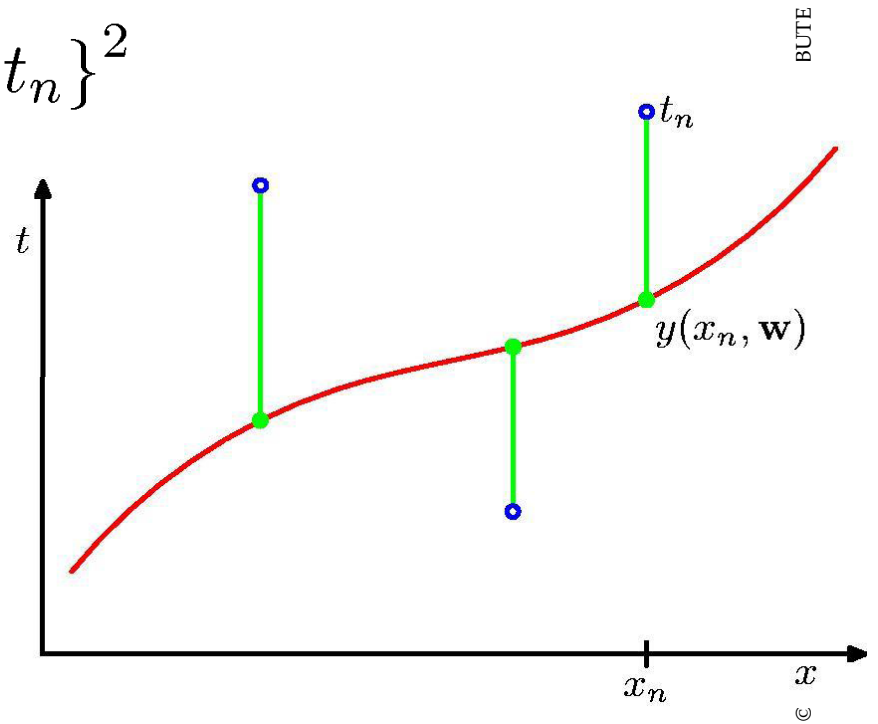
Coefficients **w** determined by fitting the polynomial to the training data.

Note: *y* is a linear function in the unknown parameters **w**, hence it is a ***linear model***.

Minimize an *error function:* the misfit between the function $y(x,\mathbf{w})$, for any given value of $\mathbf{w}$, and the training set data points.

Sum-of-Squares Error (SSE) Function

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



For a linear model, unique solution! → w*

Learned model: $y(x,\mathbf{w}^*)$

To summarize

(1) Given a dataset

$$\mathbf{x} \equiv (x_1, \ldots, x_N)^{\mathrm{T}}$$
$$\mathbf{t} \equiv (t_1, \ldots, t_N)^{\mathrm{T}}$$

(2) choose a (family of) models

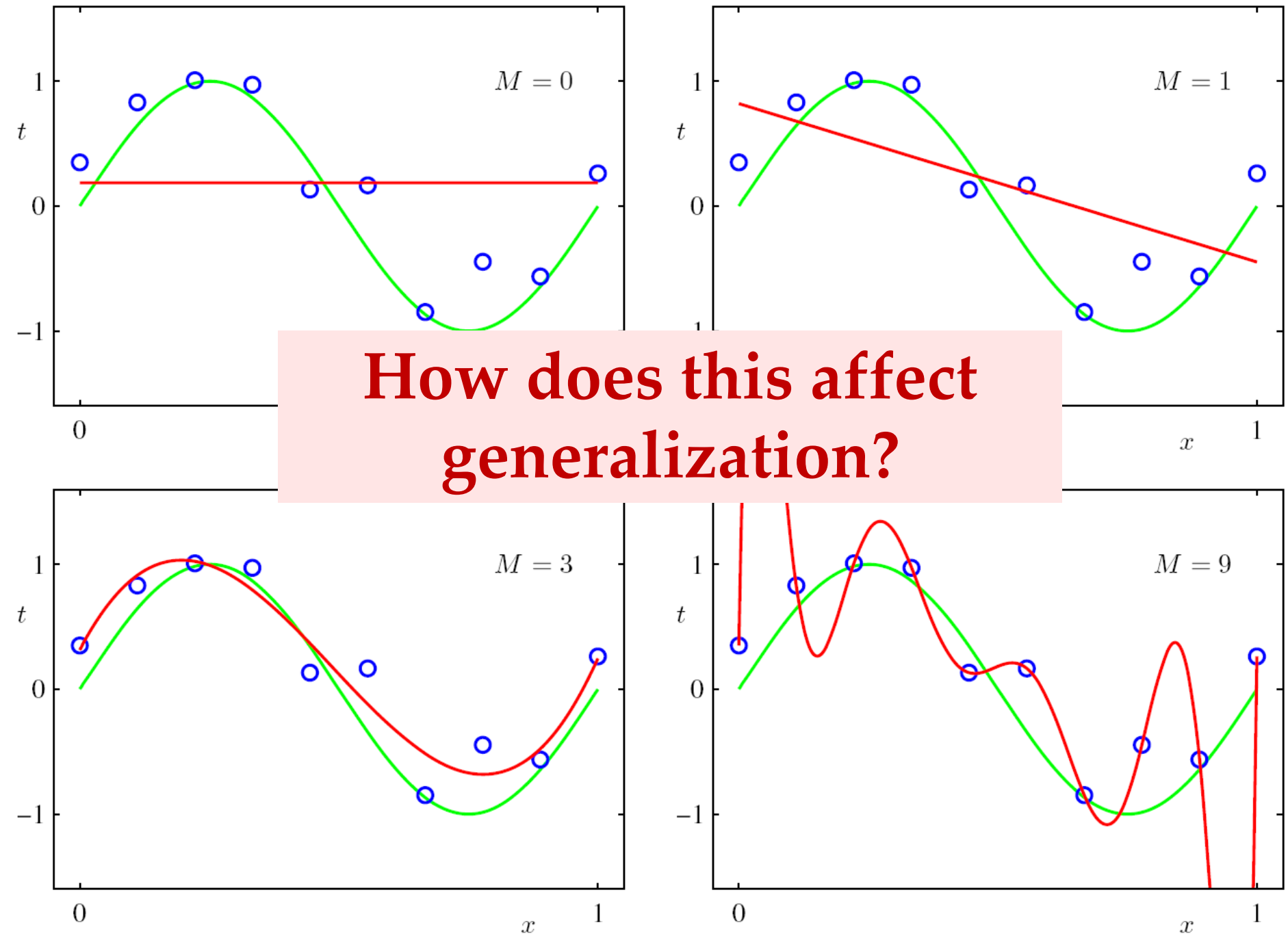
$$y(x, \mathbf{w}) = w_0 + w_1 x + w_2 x^2 + \ldots + w_M x^M = \sum_{j=0}^{M} w_j x^j$$

(3) learn the model by minimizing an error function (in this case SSE):

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

(4) we use the learned model $y(x,\mathbf{w}^*)$ to predict the value of y for new x (e.g. $x = 0.5738$)

How does the choice of the degree of the polynomial (M) affect the solution?

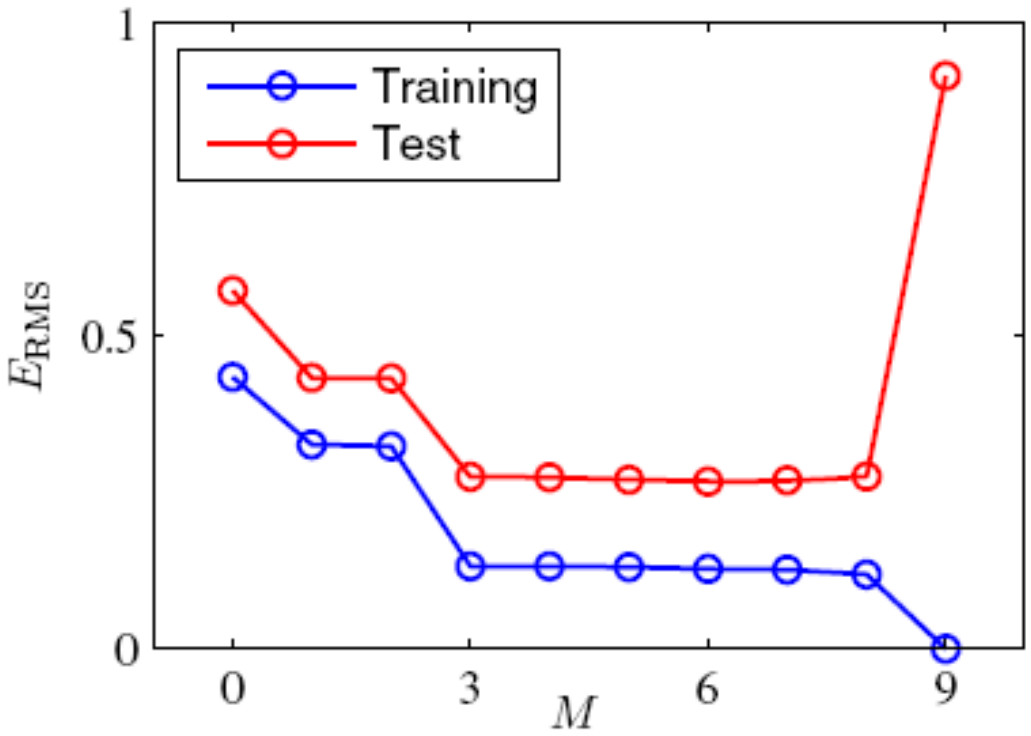


How does this affect generalization?

Take a "test set" and measure the error on the test set

Root-mean-square error (RMS) (but I could have also used the SSE)

$$E_{\mathrm{RMS}} = \sqrt{2E(\mathbf{w}^\star)/N}$$



☺ It's obvious !

☹ It's strange !

Let's look at the coefficients w* for different values of M

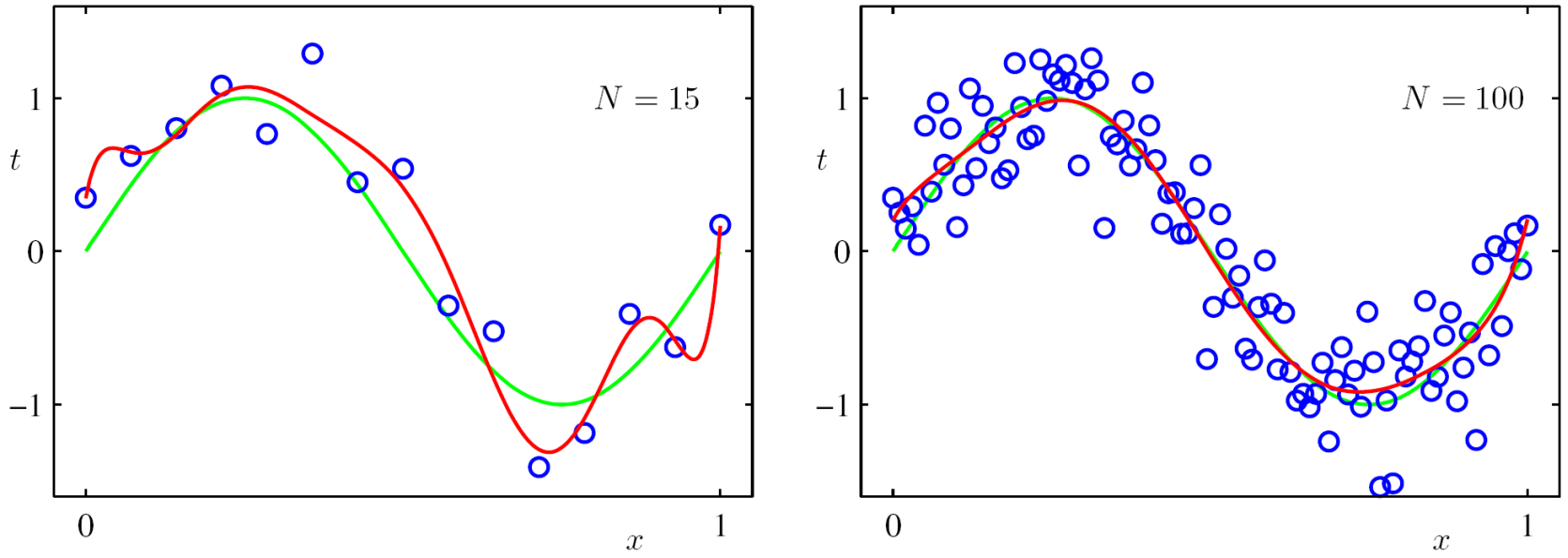
	$M=0$	$M=1$	$M=6$	$M=9$
$w_0^\star$	0.19	0.82	0.31	0.35
$w_1^\star$		-1.27	7.99	232.37
$w_2^\star$			-25.43	-5321.83
$w_3^\star$			17.37	48568.31
$w_4^\star$				-231639.30
$w_5^\star$				640042.26
$w_6^\star$				-1061800.52
$w_7^\star$				1042400.18
$w_8^\star$				-557682.99
$w_9^\star$				125201.43

as *M* increases, the coefficients typically get larger

More flexible polynomials are becoming increasingly tuned to the random noise on the target values: **OVERFITTING**

(1) let's increase the dataset

$M = 9$



Over-fitting becomes less severe as the size of the data set increases.

The larger the data set, the more complex the model that we can afford to fit to the data.

(2) Regularization

We add a penalty term to the error function in order to discourage the coefficients from reaching large values.

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

penalty/regularization term

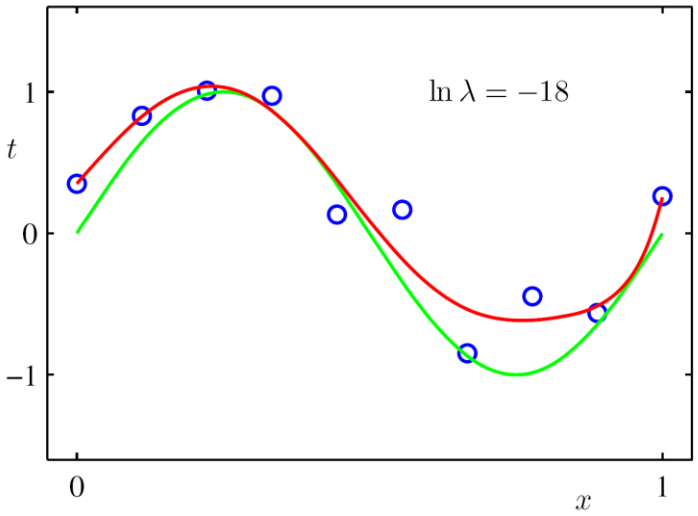
where:

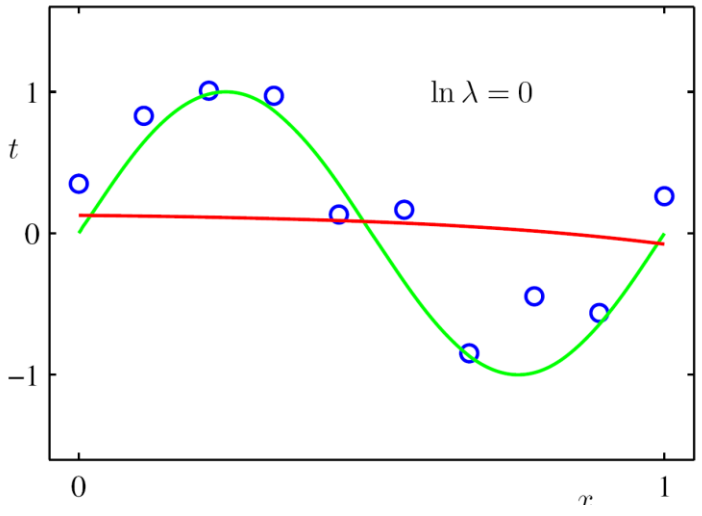
$$\|\mathbf{w}\|^2 \equiv \mathbf{w}^{\mathrm{T}}\mathbf{w} = w_0^2 + w_1^2 + \ldots + w_M^2$$

λ controls the importance of the regularization term

(shrinkage, ridge regression, weight decay)

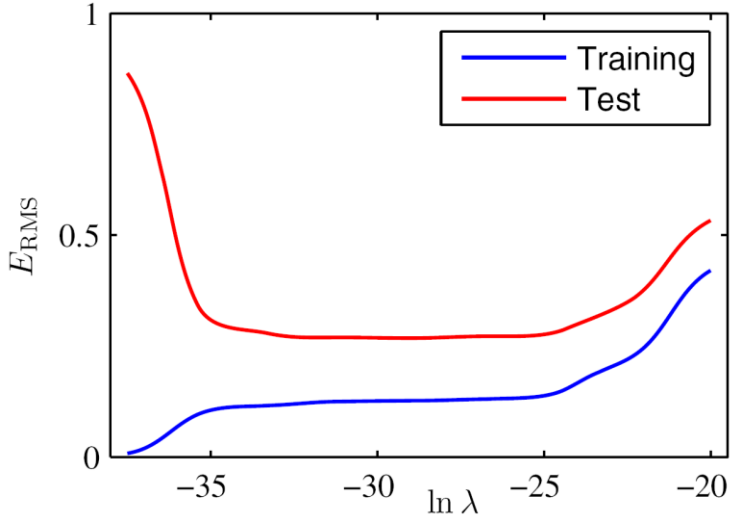
$M = 9$





	$\ln \lambda = -\infty$	$\ln \lambda = -18$	$\ln \lambda = 0$
$w_0^\star$	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

(no regularization)



Regularization had a similar effect to increasing the number of datapoints ! (reduced overfitting)

The curse of Dimensionality

In many practical applications we will have to deal with spaces of high dimensionality

(many input variables/ many features per datapoint)

Variables can be numerical or categorical

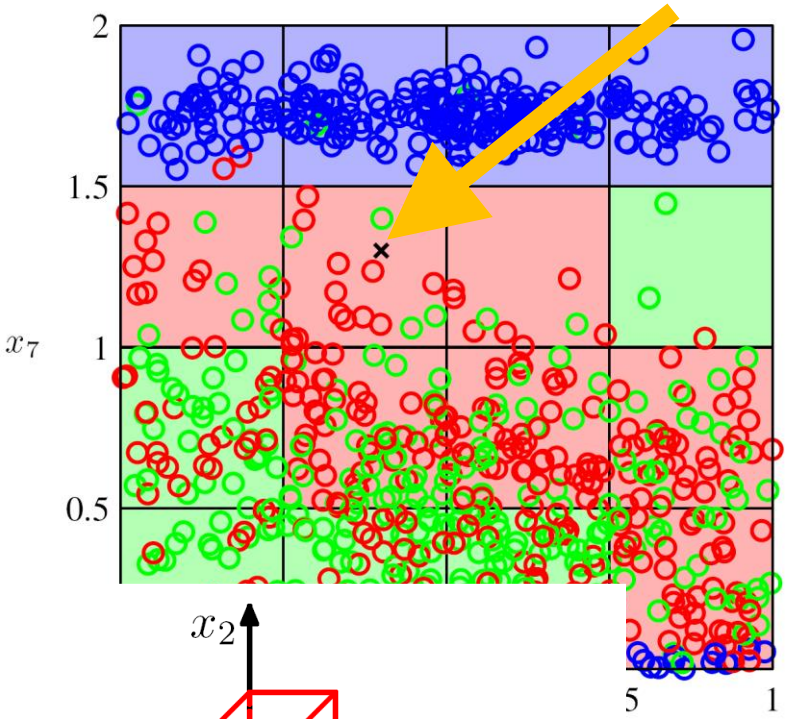
We write datapoints as:

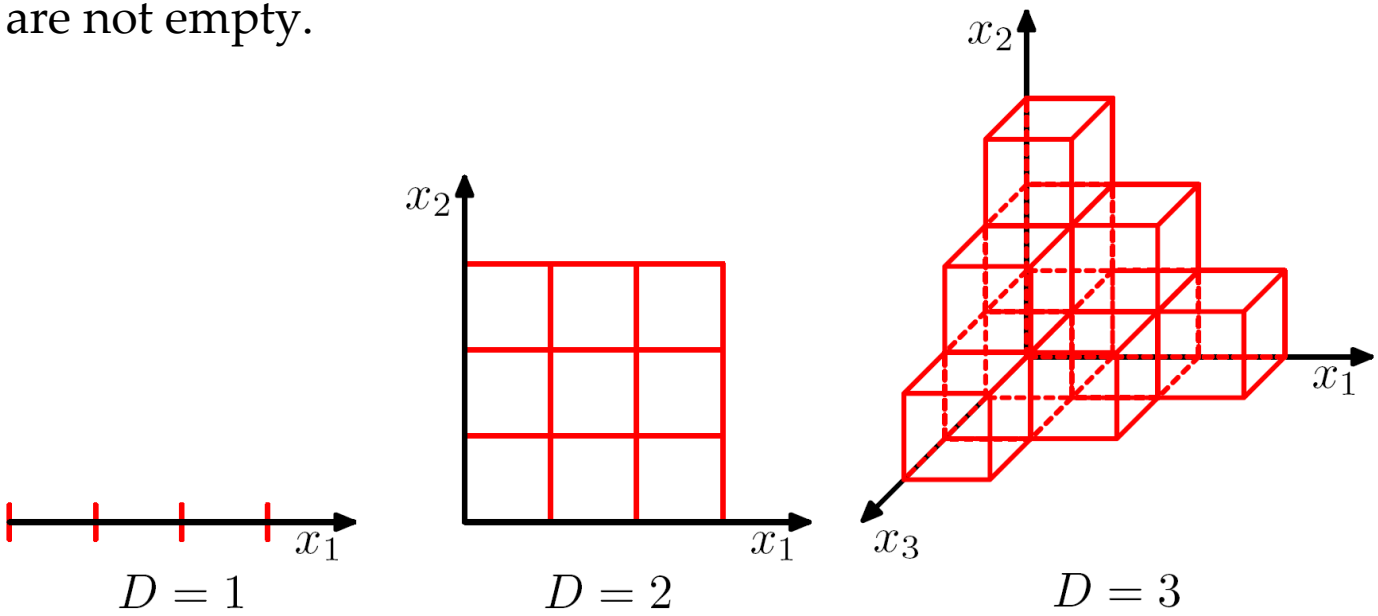
$$\boldsymbol{x} = (x_1, x_2, x_3, \ldots x_n)$$

Example: points that belongs to 3 classes

Idea: determine the identity of the cross using nearby points from the training set.

- The number of cells grows exponentially with the dimensionality of the space.
- We need an exponentially large quantity of training data to ensure cells are not empty.





…back to polynomial curve fitting

We saw the example with one input variable:

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

If we have *D* input variables, then a general polynomial with M = 3 would take the form

$$y(\mathbf{x}, \mathbf{w}) = w_0 + \sum_{i=1}^{D} w_i x_i + \sum_{i=1}^{D}\sum_{j=1}^{D} w_{ij} x_i x_j + \sum_{i=1}^{D}\sum_{j=1}^{D}\sum_{k=1}^{D} w_{ijk} x_i x_j x_k$$

As *D* increases the number of model parameters grows proportionally to D^3…

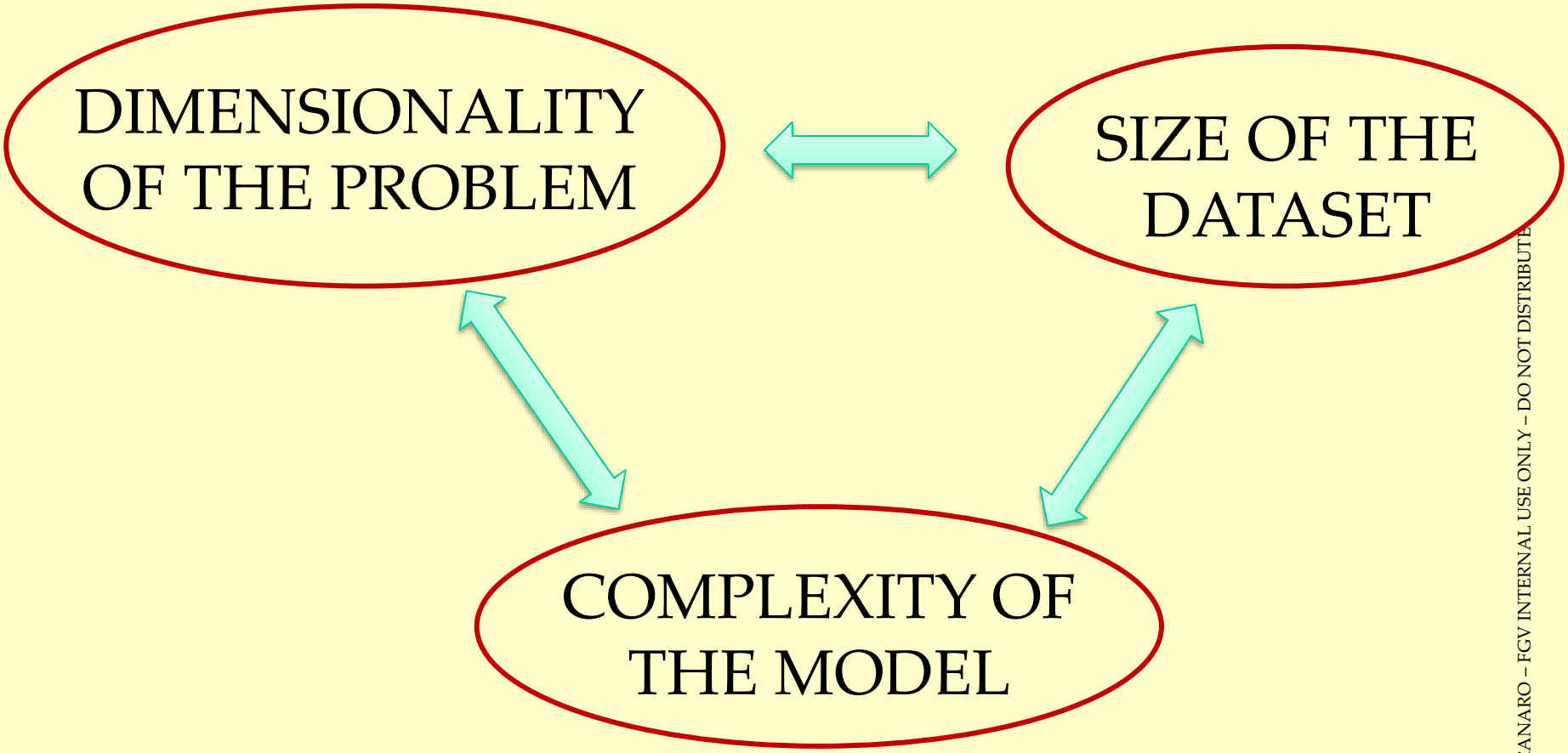
Is this hopeless then? How can *anything* work ?

Because real data will often:

1. be confined to a region of the space having **lower effective dimensionality**

2. **exhibit some smoothness properties** (at least locally), so we can exploit local interpolation-like techniques

KEY MESSAGE



Model Selection

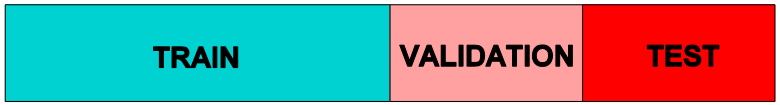
In any model you will have 2 types of parameters:

1. Parameters that the ML algorithm will tune (e.g. **w** in the polynomial curve fitting)
2. Parameters that need to be set by the user (e.g. λ controlling the complexity of the model, or even the type of model)

Goal: **GENERALIZATION !**

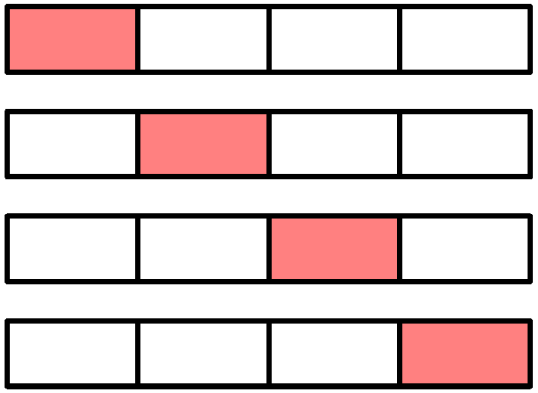
Setting aside a test set is not enough… ☹

- If there is plenty of data:



- Otherwise, CROSSVALIDATION:

1. Partition data into *S* groups
2. Use *(S-1)* groups for training and 1 group for testing
3. Repeat step 2 *S* times
4. Average the *S* performance scores



S = 4

When S = N (size of the dataset) then it is called leave-one-out (LOO)

What can go wrong in practical applications?

- Insufficient quantity of training data
- Non representative training data
- Poor quality data (high level of noise, missing data)
- Irrelevant features