Linear and K-nn models

Alessio Micheli

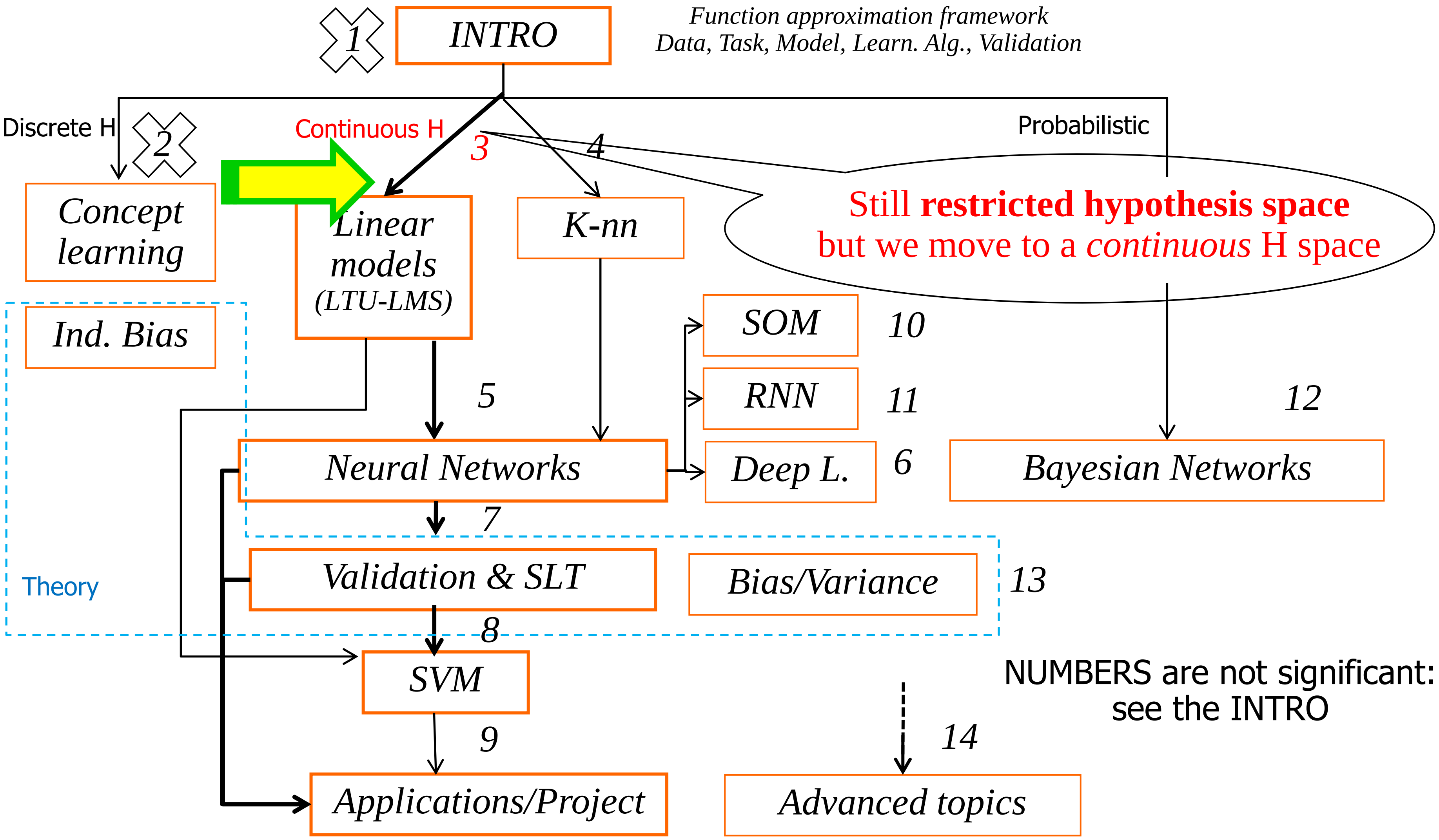
micheli@di.unipi.it

Dipartimento di Informatica
Università di Pisa - Italy

**Computational Intelligence &
Machine Learning Group**

Sept, 2022

ML Course structure
Where we go

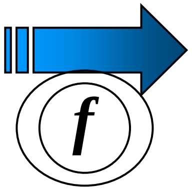


NUMBERS are not significant:
see the INTRO

Tasks: Supervised Learning

- Given: Training examples as $<input,output>=(x,d)$ (***labeled* examples**) for an unknown function f (known only at the given points of example)
  - Target value: desiderate value d or t or y … is given by the teacher according to $f(\mathbf{x})$.
- Find: A *good* approximation to f (a hypothesis h that can used for prediction on unseen data x', i.e. that is able to generalize)

x
Input space



Categories o real values (*IR*)

- Target d (or t or y): a categorical or numerical *label*
  - ***Classification***: discrete value outputs:
    $f(\mathbf{x}) \in \{1,2,\ldots,K\}$ *classes* (*discrete-valued function*)
  - ***Regression***: real continuous output values (approximate a real-valued target function)

Both as a *task of function approximation*

A premise on DATA notation

Pattern	x_1	x_2	x_i	x_n
Pat 1	$x_{1,1}$	$x_{1,2}$		$x_{1,n}$
…				
Pat *p*	$x_{p,1}$	$x_{p,2}$	$\boxed{x_{p,i}}$	$x_{p,n}$
…				

$\mathbf{X}$ is a matrix $\boldsymbol{l} \times \boldsymbol{n}$

$\boldsymbol{l}$ rows, $\boldsymbol{n}$ columns

$p=1..l, \quad i=1..n$

We often need to omit some indices when the context is clear, e.g.:

- Each row, generic $\mathbf{x}$ (vector - bold), a raw in the table: (input) example, pattern, instance, sample ,…, input vector, …
- x_i or x_j (scalar): component *i* or *j* (given a pattern $\mathbf{x}$, i.e. omitting p)
- $\mathbf{x}_p$ or $\mathbf{x}_i$ (vector – bold) *p-th* or *i-th* raw in the table = pattern p or *i*
- $\boxed{x_{p,i}}$ (scalar) also as $(\mathbf{x}_p)_i$: component *i* of the pattern p
  or also, often, $x_{p,j}$ for the component j, etc.
- For the target y we will typically use just y_p with $p=1..l$ (the same for *d* or *t*)

Linear models

- Regression
- Classification

Linear models

The linear model has been the mainstay of statistics.

- "*Despite the great inroads made by modern nonparametric regression techniques, linear models remain important, and so we need to understand them well*". (Hastie)

Plenty of studies and in many books (mathematics, statistics, numerical analysis, applicative fields, ML, …)

- We start with the simplest form, linear in the input variables
- A baseline for learning (first: is it a linear problem?)

Regression

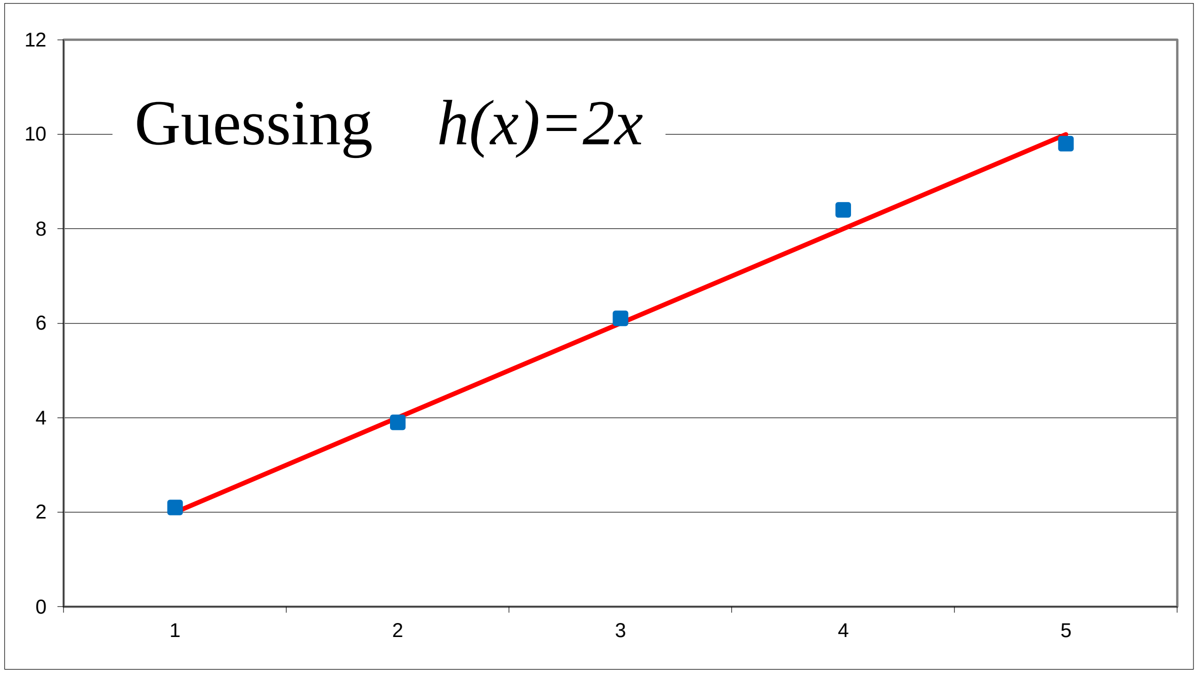
- **Just to see how formulate the learning problem as a LMS on the R_{emp}**
  - Then (next lecture) we will consider also the control of complexity
- **We formulate a first derivation in a simplified setting (univariate case)**

Repetita: regression: example

- Process of estimating of a real-value function on the basis of finite set of noisy samples
  - known pairs(x , $f(x)$+*random noise*)

Task (exercise): find f for the data in the following table:

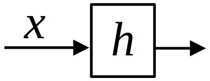
x	*target*
1	2.1
2	3.9
3	6.1
4	8.4
5	9.8
…	…



Now, we want to solve it (how to find w) in a «systematic» way

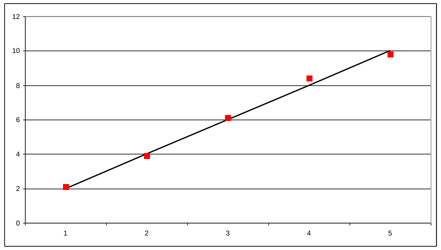
Univariate Linear Regression

- Univariate case, simple linear regression :
- We start with 1 input variable x, 1 output variable y
- We assume a model $h_{\mathbf{w}}(x)$ expressed as $\boxed{out=w_1x+w_0}$



- *where* ***w*** *are real-valued coefficients/free parameters (weights)*

- ***Fitting*** the data by a "straight line"



- Infinite hp space (continuous w values) but we have nice solution from classical math (going back to Gauss/Legendre ~1795!)
  - Surprisingly we can "learn" by this basic tool
  - Although simple it includes many relevant concept of modern ML and it is a basis of evolved methods in the field

Build it: Learning via LMS (I)

- Learn → find ***w*** such that minimize **error**/empirical **loss** (best data fitting – on the training set with *l* examples)
- **Given** a set of *l* training examples $(\mathbf{x}_p, y_p)$ $p=1..l$
- **Find**: $h_w(x)$ in the form w_1x+w_0 (hence the values of ***w***) that minimizes the expected loss on the training data.

- For the loss we use the square of errors:
- Least (Mean) Square: Find ***w*** to *minimize* the residual sum of squares $[argmin_{\mathbf{w}}\ Error(\mathbf{w})\ in\ L_2]$:

p runs over patterns/examples

$$Loss(h_{\mathbf{w}}) = E(\mathbf{w}) = \sum_{p=1}^{l}(y_p - h_{\mathbf{w}}(x_p))^2 = \sum_{p=1}^{l}(y_p - (w_1x_p + w_0))^2$$

where x_p is p-th input/pattern/example, y_p the output for p, w free par., l num. of examples

Note: to have the mean divide by *l*

On the notation: Indeed for the univariate case, with 1 variable: $x_p = x_{p,1} = (\mathbf{x}_p)_1$

Build it: Learning via LMS (II)

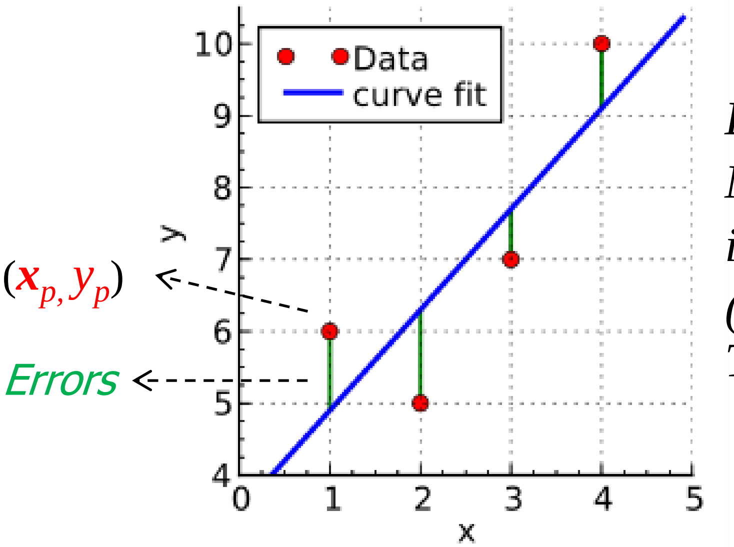
Dip. Informatica
University of Pisa

Why LMS to find the best *h*?

- Least (Mean) Square: Find $\mathbf{w}$ to *minimize* the residual sum of squares [$argmin_{\mathbf{w}}\ Error(\mathbf{w})$ *in* L_2]:

$h_{\mathbf{w}}(x)$

$y = w_1 x + w_0 + noise$



Different blue lines will have different green bars.

Minimizing the green bars (residuals /errors)

is a way to find the best approximation/fitting of the data (i.e. our $h_{\mathbf{w}}(x)$ *or blue line).*

The squares of errors $E(\mathbf{w})$ *quantify such green bars:*

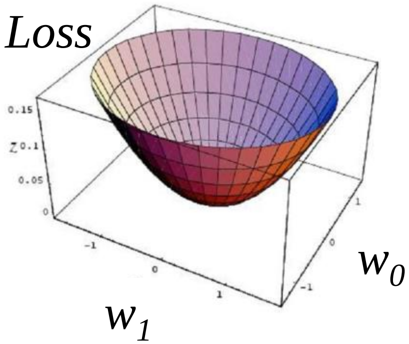
$$E(\mathbf{w}) = \sum_{p=1}^{l} \left(y_p - h_{\mathbf{w}}(x_p)\right)^2$$

- The method of **least squares** is a standard approach to the approximate solution of over-determined systems, i.e., sets of equations in which there are more equations than unknowns.

How to solve?

- Remember: local minimum as stationary point: the gradient is zero

$$\frac{\partial E(\mathbf{w})}{\partial w_i} = 0, \qquad i = 1,\ldots,\dim\_input + 1 = 1,\ldots,n+1$$



- For the simple Lin. Regr. (2 free parameters)

Search the w such that $$\frac{\partial E(\mathbf{w})}{\partial w_0} = 0 \qquad \frac{\partial E(\mathbf{w})}{\partial w_1} = 0$$

Convex loss function → *we have the following solution (no local minima)*
(just to know that it exists!)

$$\boxed{w_1 = \frac{\sum x_p y_p - \frac{1}{l}\sum x_p \sum y_p}{\sum x_p^2 - \frac{1}{l}(\sum x_p)^2} = \frac{\mathrm{Cov}[x,y]}{\mathrm{Var}[x]}}, \qquad \boxed{w_0 = \bar{y} - w_1 \bar{x}}$$

$$p: 1 \to l$$

$$\frac{1}{l}\sum_{p->l} y_p \qquad \frac{1}{l}\sum_{p->l} x_p$$

Exercise: *compute w_0 and w_1 according to the next slide results for the gradient (extended to l patterns)*

Compute the gradient for 1 (each) pattern *p*

Redo this by yourself as an *Exercise*

Basic rules:

$$\frac{\partial}{\partial w} k = 0, \frac{\partial}{\partial w} w = 1, \frac{\partial}{\partial w} w^2 = 2w$$

$$\frac{\partial (f(w))^2}{\partial w} = 2 f(w) \frac{\partial (f(w))}{\partial w}$$

Der. sum = sum of der.

We will call (y-$h(x)$) "delta"

$$\frac{\partial E(\mathbf{w})}{\partial w_i} = \frac{\partial (y - h_{\mathbf{w}}(x))^2}{\partial w_i} =$$

$$= 2(y - h_{\mathbf{w}}(x)) \frac{\partial (y - h_{\mathbf{w}}(x))}{\partial w_i} = 2(y - h_{\mathbf{w}}(x)) \frac{\partial (y - (w_1 x + w_0))}{\partial w_i}$$

$$\boxed{\frac{\partial E(\mathbf{w})}{\partial w_0} = -2(y - h_{\mathbf{w}}(x))} \qquad \boxed{\frac{\partial E(\mathbf{w})}{\partial w_1} = -2(y - h_{\mathbf{w}}(x)) \cdot x}$$

Then we will sum up for l patterns (x_p, y_p) …

And we will extend to multidimensional $\mathbf{x}$ and $\mathbf{w}$ *(n=1 here)* …

Linear model: notation for multidimensional inputs

- Assuming column vector for $\boldsymbol{x}$ and $\boldsymbol{w}$ *(in bold)*
- Number of data l, dimension of input vector n, y_p *(targets)* $p=1..l$

d_p or t_p

$$\boldsymbol{w}^T\boldsymbol{x} + w_0 = w_0 + w_1x_1 + w_2x_2 + \cdots + w_nx_n = w_0 + \sum_{i=1}^{n} w_ix_i \qquad \text{(eq. 1)}$$

- Note that sometimes (in NN) the transpose notation T in $\boldsymbol{w}^T$ is omitted

- w_0 is the *intercept, threshold, bias, offset.....*

Often it is convenient to include the constant $x_0 = 1$ so that we can write eq.1 as :

$$\boldsymbol{w}^T\boldsymbol{x} = \boxed{\boldsymbol{x}^T\boldsymbol{w}}$$

Inner product

$$\boldsymbol{x}^T = [1, x_1, x_2, .., x_n]$$

$$\boldsymbol{w}^T = [w_0, w_1, w_2, .., w_n]$$

So, the "linear "model can be written as a function that for each $\boldsymbol{x}_p$ compute:

$$h(\boldsymbol{x}_p) = \boldsymbol{x}_p{}^T\boldsymbol{w} = \sum_{i=0}^{n} x_{p,i}\, w_i$$

Note: *w* continuous (free) parameters "weights"

Learning algorithm: just wait!

- For the learning algorithm in the multidimensional case for linear regression: please wait.
- We will provide it along with the learning algorithm of the linear classifier in the next few slides.

Summing up:

- Given the data set and the linear model, we can state the learning problem as LMS problem
- Once we find the best $\boldsymbol{w}$ parameters values, we have our $h_w(x)$ for regression purposes

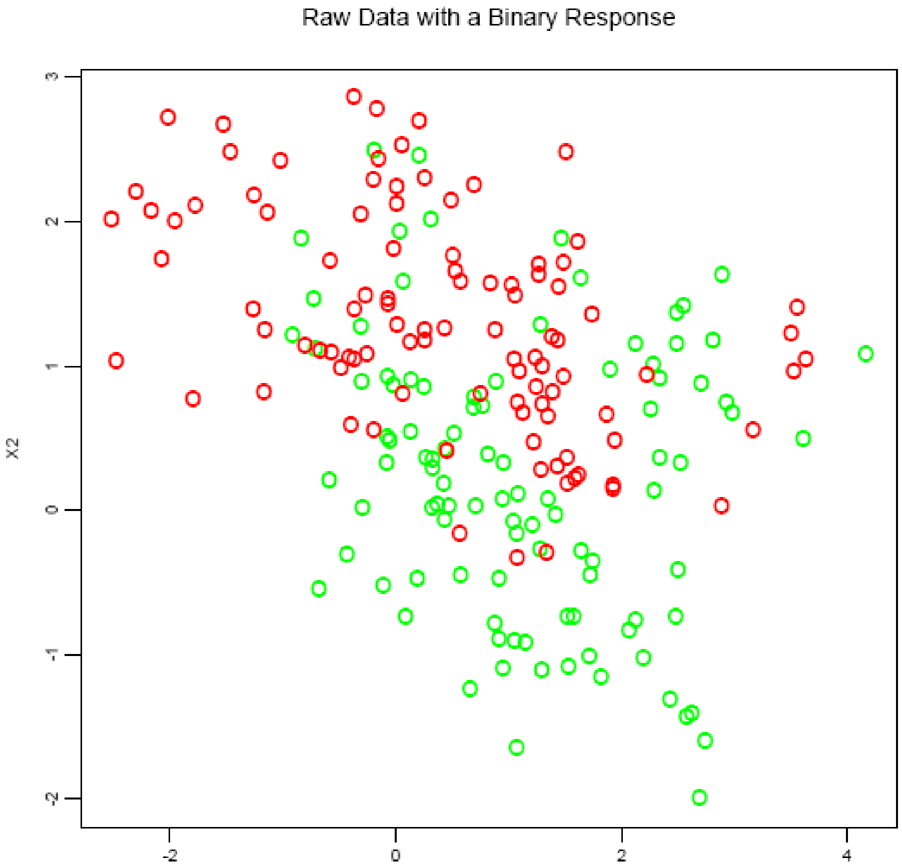
- For students that need a soft intro to just LMS Regression:
  - https://svivek.com/teaching/machine-learning/lectures/slides/linear-models/lms-regression.pdf

Classification

What we are looking at

1. The classification by hyperplanes
2. See how the model works after training ("use it" slides)
3. How to state/formulate a (regression)/classification learning problem for a linear model by LMS
4. How to derive the learning algorithm
5. Proposing two learning algorithms to build a linear classifier

Problem: example



200 points generated in IR^2 from an unknown distribution; 100 in each of two classes.

Can we build a rule to predict the color of future points?

Data may be generated by gaussian distribution (for each class) with different means

or by a mixture of different low variance gaussian distributions.

We reuse the linear model

- The same models (used for regression) can be used for **classification**: categorical targets (y or d), **e.g. 0/1 or -1/+1.**

○ In this case we use a hyperplane ($\boldsymbol{wx}$) assuming negative or positive values

○ We exploit such models to decide if a point $\boldsymbol{x}$ belong to positive or negative zone of the hyperplane (to classify it)

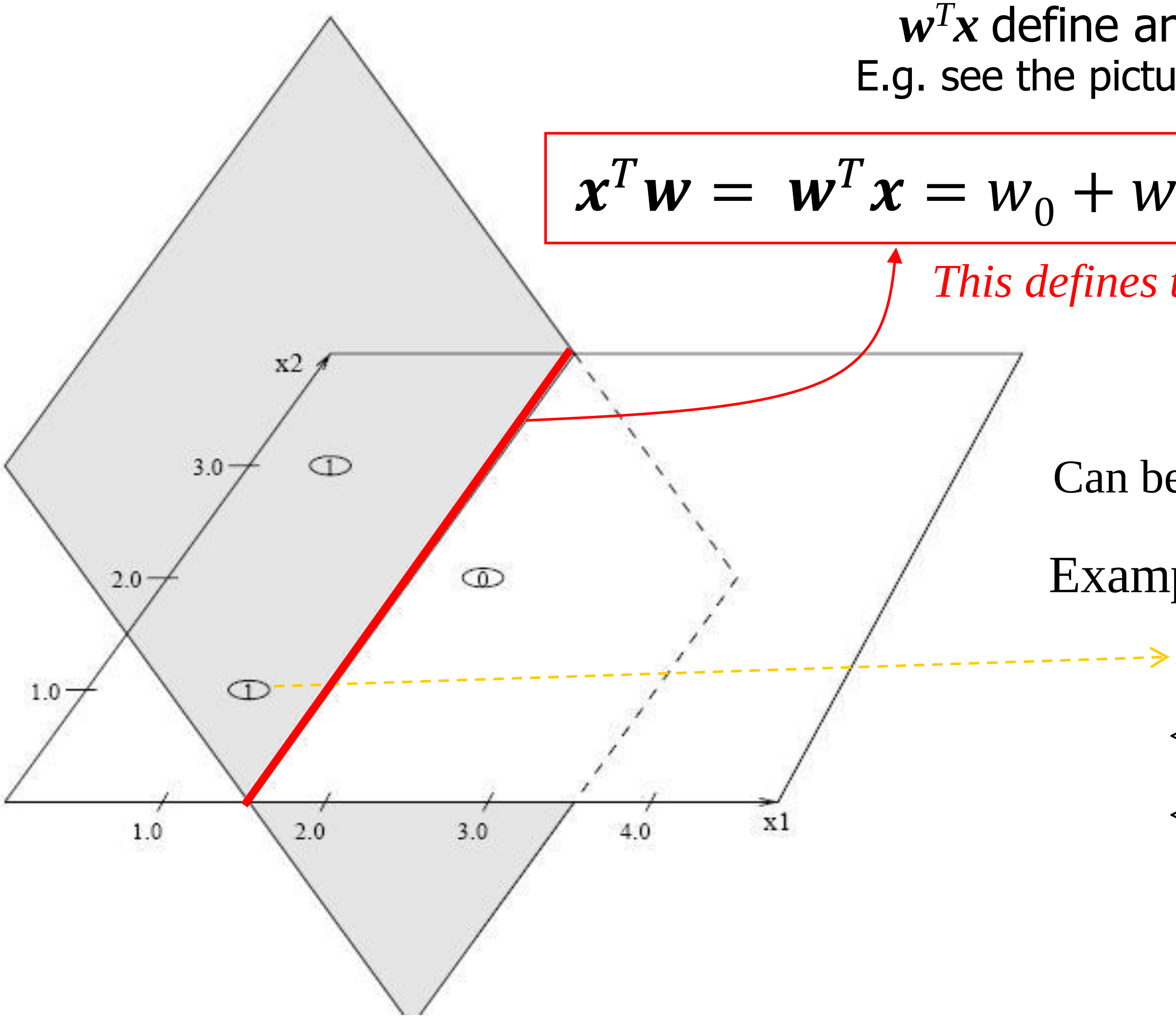
○ So we want to set $\boldsymbol{w}$ (by learning) s.t. we get good classification accuracy

Geometrical view: hyperplane

$\boldsymbol{w}^T\boldsymbol{x}$ define an hyperplane.
E.g. see the picture for 2 variables

$$\boxed{\boldsymbol{x}^T\boldsymbol{w} = \boldsymbol{w}^T\boldsymbol{x} = w_0 + w_1x_1 + w_2x_2 = 0}$$

This defines the decision boundary



Can be used to classify:

Examples $<(x_1,x_2), y>$:

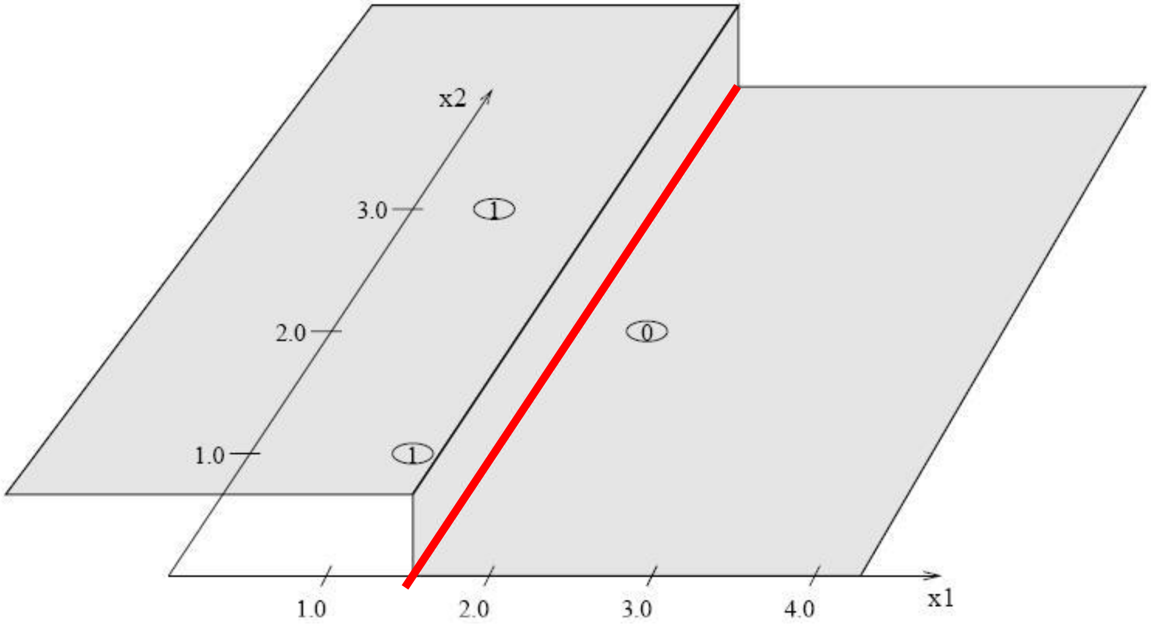
<(1.0, 1.0), 1>

<(0.5, 3.0), 1>

<(2.0, 2.0), 0>

Geometrical view: classifier

Introducing a threshold function



Examples:

<(1.0, 1.0), 1>

<(0.5, 3.0), 1>

<(2.0, 2.0), 0>

[0,1] output range

$$h(\mathbf{x}) = \begin{cases} 1 & \_\,if \quad \mathbf{w}^T\mathbf{x} + w_0 \geq 0 \\ 0 & \_\_\_\_\_\_\_\,otherwise \end{cases}$$

or

[-1,+1] output range

$$h(\mathbf{x}) = sign(\mathbf{w}^T\mathbf{x} + w_0)$$

Using $\mathbf{x}_p$ and including w_0 in $\mathbf{w}$

$$h(\mathbf{x}_p) = sign(\mathbf{x}_p{}^T\mathbf{w}) = sign\left(\sum_{i=0}^{n} x_{p,i} w_i\right)$$

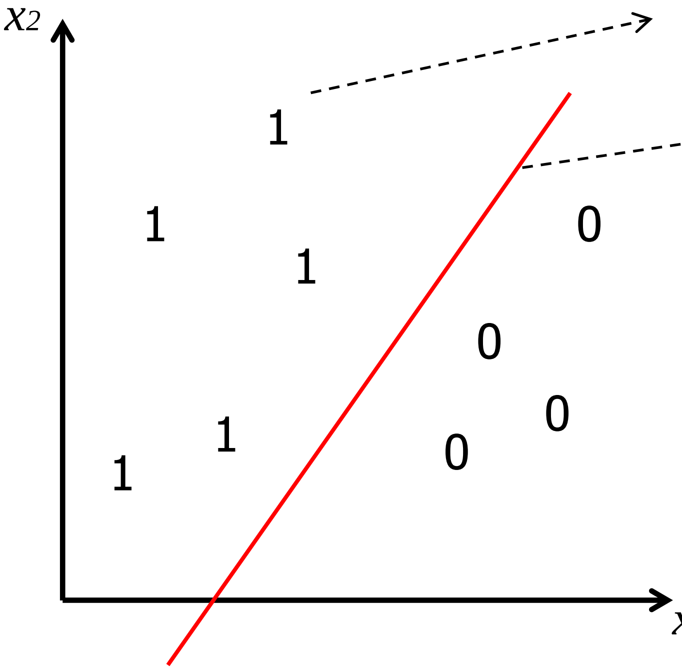
*In this slide **w** is omitted from $h_{\mathbf{w}}$ (we use just h)*

Classification by linear decision boundary [repetita]

The classification may be viewed as the allocation of the input space in decision regions (e.g. **0/1**)

Example: linear separator on

2-dim instance space $\boldsymbol{x}=(x_1,x_2)$ *in* $I\!R^2$, $f(\boldsymbol{x})=0/1$ (*or* $-1/+1$)



Point belonging to class 1

Separating (hyper)plane : ***x*** s.t.

$$\boldsymbol{w}^T\boldsymbol{x} + w_0 = w_1x_1 + w_2x_2 + w_0 = 0$$

$$\boxed{h(\boldsymbol{x}) = \begin{cases} 1 & \_if \quad \boldsymbol{w}^T\boldsymbol{x} + w_0 \geq 0 \\ 0 & \_\_\_\_\_\_\_otherwise \end{cases}}$$

[0,1] output range

or

$$\boxed{h(\boldsymbol{x}) = sign(\boldsymbol{w}^T\boldsymbol{x} + w_0)}$$

[-1,+1] output range

Linear threshold unit (LTU)

Indicator functions

How many? (H): set of dichotomies induced by hyperplanes

Threshold (bias w_0)

Note that, given the bias w_0, in the LTU

saying $h(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + w_0 \geq 0$

is equivalent to say $h(\mathbf{x}) = \mathbf{w}^T\mathbf{x} \geq -w_0$

with $-w_0$ as the «threshold» value

- The two forms identify the same positive zone of the classifier
- The second one emphasizes the role of the bias as a threshold value to "activate" the +1 output of the classifier.

Use it: Example (AIMA)
Classify a new data (x_1,x_2)

Dip. Informatica
University of Pisa

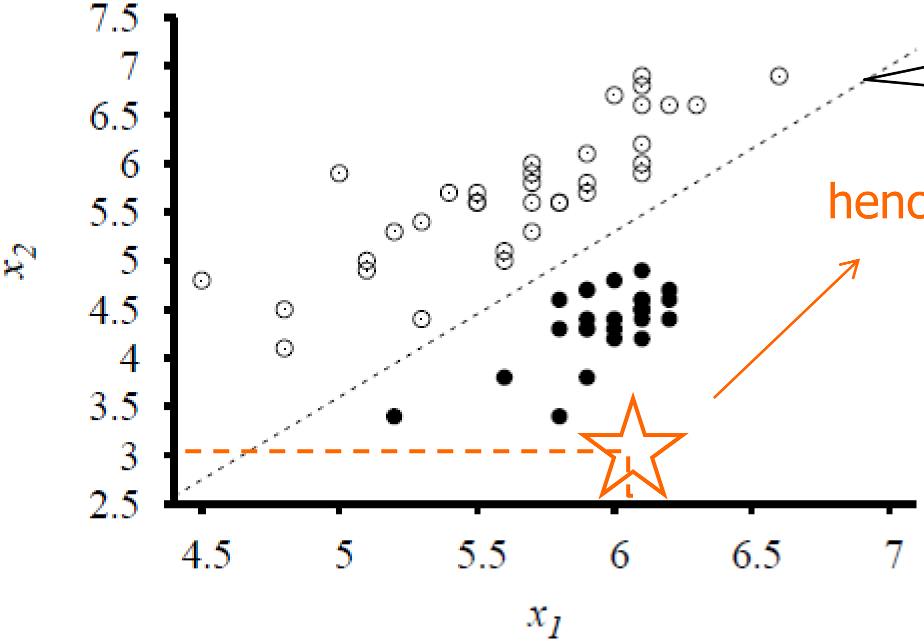
Find h s.t. given (x_1,x_2) return 0/-1 for *Earthquakes* and 1 for *Nuclear Explosion*

Some alg. finds this decision boundary:

$$w_0 + w_1 x_1 + w_2 x_2 = 0$$

$$\Updownarrow \quad \Updownarrow \quad \Updownarrow$$

$$-4.9 + 1.7 x_1 - 1 x_2 = 0$$



hence

***h(6,3)**=sign(w_0 +w_1*6+w_2*3)=*

=sign(- 4.9 + 1.7*6 -1*3)

=sign(2.3) =+1 ➔ nuclear expl.

Seismic data.

x_1 body wave magnitude, x_2 surface wave magnitude

Earthquakes (white) Nuclear Explosion (black)

1982-1990 Asia

Getting new examples is expensive;-)

Use it: Example (Spam)

- Find *h*(*mail*) +1 for *spam*, -1 *not-spam*
  - Features Phi(*mail*) = words [0/1] or phrases ("free money") [0/1] or length [integer]
  - e.g $\phi_k(\mathbf{x}) = contain(word_k)$ [*bag of words* representation]
- w → weight contribution of the input features to prediction
  - e.g. positive weight for "free money", negative for ".edu" or "unipi"
- $\mathbf{x}^T \mathbf{w}$ is the weight combination
- $h_{\mathbf{w}}(x)$ provides the threshold to decide spam/not spam

$$h_{\mathbf{w}}(\mathbf{x}) = sign\left(\sum_k w_k \phi_k(\mathbf{x}) \right)$$

>0 → + 1 = Spam !

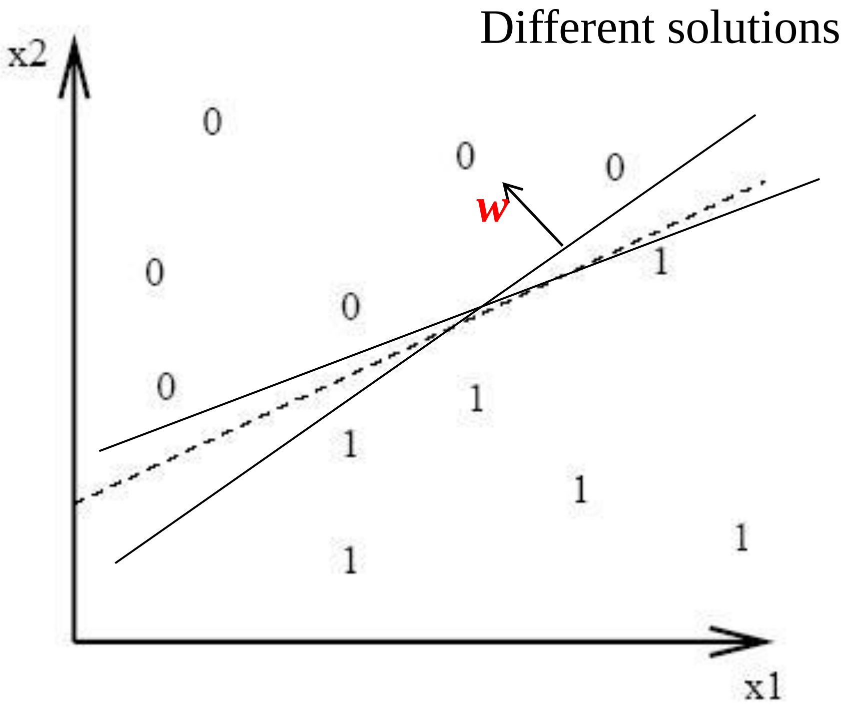
Some useful properties

(we will use them in future lectures)

Dip. Informatica
University of Pisa

$$\boldsymbol{w}^T\boldsymbol{x} + w_0 = w_1x_1 + w_2x_2 + w_0 = 0$$

Exercise: Draw $x_2 = - x_1\, w_1/w_2 - w_0/w_2$ with different $\boldsymbol{w}$ values



A linearly separable problem

- If w_0=0 the line goes through the *origin* of the coordinate system.
- If $n > 2$ → *hyperplane (decision boundary)*
- *Scaling freedom*: the same decision boundary multiplying $\boldsymbol{w}$ by K
- $\boldsymbol{w}$ is a vector *orthogonal* to the hyperplane:

Given $\mathbf{x}_{a,}\ \mathbf{x}_b$ ***(belonging to the sep. hyperplane):***
$\mathbf{w}^T\mathbf{x}_a + w_0 = 0;\ \mathbf{w}^T\mathbf{x}_b + w_0 = 0$ *(take the diff.)*→
$\mathbf{w}^T(\mathbf{x}_a - \mathbf{x}_b) = 0$ → ***orthogonal vectors*** (dot prod. 0)

- If it exist, in general, the solution is *not unique*: there are many possible hyperplanes separating these points: also many ML alg.!!!

Learning Algorithms

- We are going to introduce 2 learning algorithms
for the regression and for the classification task using a linear model, both based on LMS:

  1. A direct approach based on **normal equation** solution

  2. An iterative approach based on **gradient descent**

- We start redefining the learning problem and the loss for them (for *l* data and multidimensional inputs)

The learning problem (classification tasks)

- **Given** a set of l training examples $(\mathbf{x}_p, y_p)$ and a loss function (measure) *L*
  $y_p = \{0,1\}$ or $y_p = \{-1,+1\}$

- **Find**: The weight vector $\mathbf{w}$ that minimizes the expected loss on the training data

$$R_{emp} = \frac{1}{l}\sum_{p=1}^{l} L(h(\mathbf{x}_p), y_p)$$

- For classification: Using a piecewise constant (over $sign(\mathbf{w}^T\mathbf{x})$) for the loss can make this a difficult problem.

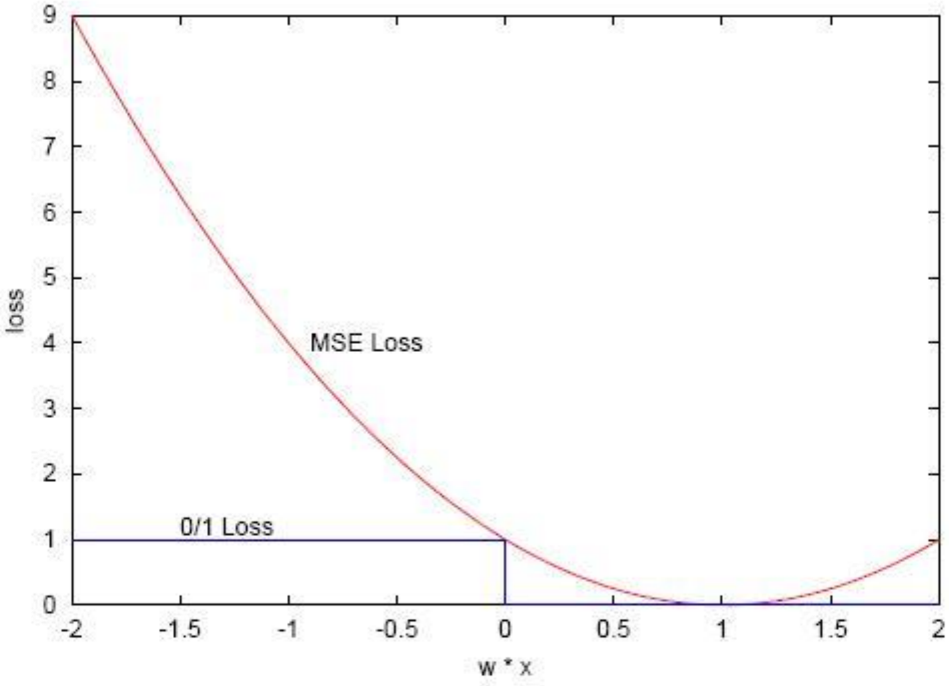
- Assume we still use the *least squares (as for the regression case)*

Approximating expected loss by a smooth function (#)

Dip. Informatica
University of Pisa

- Initially, we can make the optimization problem easier by replacing the original objective function *L (0/1 loss)* by **a smooth, *differentiable function***. For example, consider the popular *mean squared error (MSE loss)* *:

Both losses satisfy the minimization of error (*),
Let us start with LMS avoid to introduce combinatorial problems



(*) Example:
*h(**x**)=1 if* $\mathbf{w}^T\mathbf{x}>0$,
and for y (target)=1
→ *minimum err. is for* $\mathbf{w}^T\mathbf{x}>0$

Hence no classif. error minimizing either 0/1 loss or MSE loss

Learning (a classifier) by Least Squares

- Find optimal values for w (for fitting of training (TR) data) by *least squares*:
- **Given** a set of l training examples (x_p, y_p), **find** w to *minimize* the residual sum of squares:

$$E(\mathbf{w}) = \sum_{p=1}^{l} (y_p - \underline{\boldsymbol{x}_p{}^T \boldsymbol{w}})^2 = \quad ||\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w}||^2$$

Where $\mathbf{x}_p$ is p-th input vector, y_p the output for p, $\mathbf{w}$ free par., l num. of examples, n input dim.

  - Min error: if $y_p=1$ then $\boldsymbol{x}_i{}^T\boldsymbol{w} \rightarrow 1$; if $y_p=0$ then $\boldsymbol{x}_p{}^T\boldsymbol{w} \rightarrow 0$
  - ***Note***: *in E(**w**) we do **not** use h(**x**), as for regression, to hold a continuous **differentiable** loss (because h(**x**)=sign(**w**^{T}**x**) for classification) !!!*

- This is a quadratic function → minimum always exists (but may be not unique) [see course of CM]
- X is a matrix l x n with a row for each input vector x_p
- Note: The same approach is used for a *regression problem*

Learning Algorithms

- We will introduce 2 learning algorithms
for the regression and for the classification tasks using a linear model, both based on LMS

1. A direct approach based on **normal equation** solution

2. An iterative approach based on **gradient descent**

Normal equation & direct approach solution

- Differentiate $E(\boldsymbol{w})$ with respect to $\boldsymbol{w}$:
  Blackboard or Exercise (a next slide).

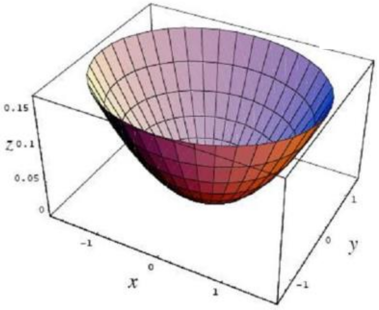
Result synthesis:

- In the derivation we find that

$$\frac{\partial E(\boldsymbol{w})}{\partial w_j} = -2\sum_{p=1}^{l}\left(y_p - \boldsymbol{x}_p{}^T\boldsymbol{w}\right) x_{p,j}$$

- We can get the **normal equation** (point with gradient of E w.r.t $\boldsymbol{w}$ =0):

$$(\boldsymbol{X}^T\boldsymbol{X})\boldsymbol{w} = \boldsymbol{X}^T\boldsymbol{y}$$



See also CM course

- If X^TX is not singular the unique solution is given by

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

\`+\` Moore-Penrose *pseudoinverse* (also if $\boldsymbol{X}$ is not invertible)

- Else the solution are infinite (satisfying the normal equation): we can choose the *min norm* ($\boldsymbol{w}$) solution

Direct approach by SVD

- The *Singular Value Decomposition (SVD)* can be used for computing the pseudoinverse of a matrix ($\boldsymbol{X}^{+}$)

$$\boldsymbol{X} = \boldsymbol{U}\underset{\substack{\downarrow \\ \text{diagonal}}}{\Sigma} V^{T} => \boldsymbol{X}^{+} = \boldsymbol{V}\boldsymbol{\Sigma}^{+}\boldsymbol{U}^{T}$$

by replacing every nonzero entry by its reciprocal

- Moreover we can apply directly SVD to compute $\boldsymbol{w} = \boldsymbol{X}^{+}\boldsymbol{y}$

obtaining the minimal norm (on $\boldsymbol{w}$) solution of least squares problem.

- Note: THIS IS the learning alg. for the direct approach solution on $\boldsymbol{w}$

- A practical tool. E.g. see “numerical recipes” in C and many numerical/statistical tools and scientific library (also in R, Octave, Matlab,…)
  - e.g. ARMADILLO: since 2011/12 C++ linear algebra library, NumPy etc.
- Many algorithms addressing the problems of efficiency and stability
- See also CM course

Solution (to find the normal eq.) Make as an Exercise !!!(#)

$$\frac{\partial E(\boldsymbol{w})}{\partial w_j} = \frac{\partial \sum_{p=1}^{l}(y_p - \boldsymbol{x}_p{}^T\boldsymbol{w})^2}{\partial w_j} = \sum_{p=1}^{l} 2(y_p - \boldsymbol{x}_p{}^T\boldsymbol{w})\ \frac{\partial(y_p - \boldsymbol{x}_p{}^T\boldsymbol{w})}{\partial w_j} =$$

$$= \sum_{p=1}^{l} 2(y_p - \boldsymbol{x}_p{}^T\boldsymbol{w})\ \left(0 - \cancel{\frac{\partial(\boldsymbol{x}_p)_1 w_1}{\partial w_j}} - \cancel{\frac{\partial(\boldsymbol{x}_p)_2 w_2}{\partial w_j}} - \ldots \frac{\partial(\boldsymbol{x}_p)_j w_j}{\partial w_j} - \cancel{\ldots.}\right) =$$

Only the component j is not 0

$$= \sum_{p=1}^{l} 2(y_p - \boldsymbol{x}_p{}^T\boldsymbol{w})\left(-\frac{\partial(\boldsymbol{x}_p)_j w_j}{\partial w_j}\right) = -2\sum_{p=1}^{l}(y_p - \boldsymbol{x}_p{}^T\boldsymbol{w})(\boldsymbol{x}_p)_j$$

Imposing this =0, we can easily obtain the *normal equation* (first by “sums”, then in matrix notations)

And we also obtained the gradient of *E*

rewritten as: $\frac{\partial E(\boldsymbol{w})}{\partial w_j} = -\sum_{p=1}^{l} 2\,\overset{\delta_p}{(y_p - \boldsymbol{x}_p{}^T\boldsymbol{w})}\, x_{p,j} = -2\sum_{p=1}^{l} \delta_p\, x_{p,j}$

(we will use this form in the future, with Neural Networks)

Other approaches to LS

Many, for instances by an **iterative/gradient descent technique** we can search for:

- More efficient solutions (previous is cubic with dim of matrix X)
- Regularization (to reduce complexity of the model)
- Better approximation with noisy data by stopping searching before the minimum

- Other approaches that can be applied also to NON-LINEAR models !!!

Hence, we are going to present the second (more important) learning algorithm

Learning Algorithms

- We will introduce 2 learning algorithms
for the regression and for the classification tasks using a linear model, both based on LMS

1. A direct approach based on **normal equation** solution
2. An iterative approach based on **gradient descent**

This approach will be the basis for fundamental approaches we will see later

Gradient descent

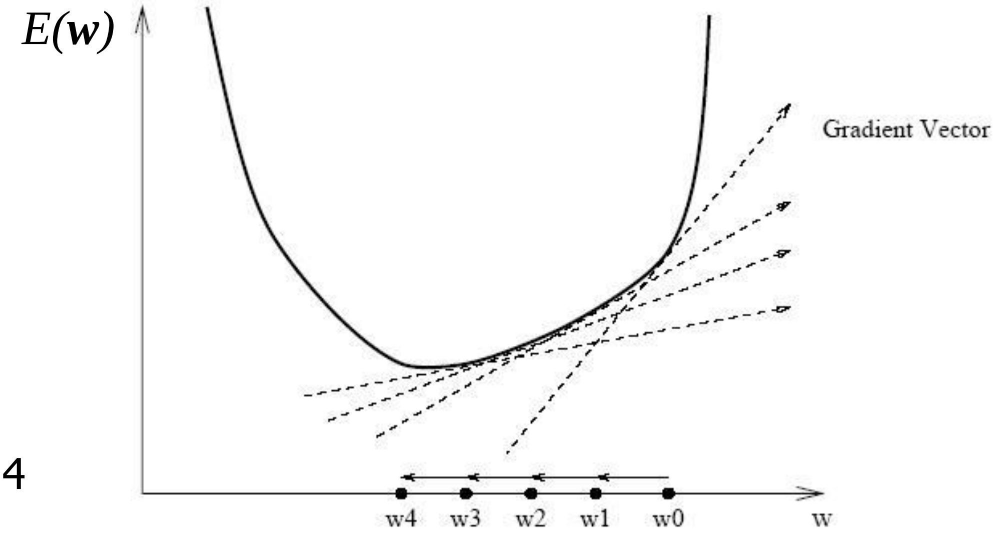
- Previous derivation suggest the line to construct an iterative algorithm based on :

*Where $\mathbf{x}_p$ is p-th input pattern, y_p the output for p, **w** free par., l num. of examples*

$$\frac{\partial E(\mathbf{w})}{\partial w_j} = -2\sum_{p=1}^{l}(y_p - \boldsymbol{x}_p{}^T\mathbf{w})(\boldsymbol{x}_p)_j$$

Component j of pattern p, also $x_{p,j}$

- Gradient = ascent direction: we can move toward the minimum with a gradient descent (chancing w with Δw= - gradient of $E(\mathbf{w})$)
  - *Local search*: it begins with an initial weight vector. We modify it iteratively to decrease up to minimize the error function (steepest descent).



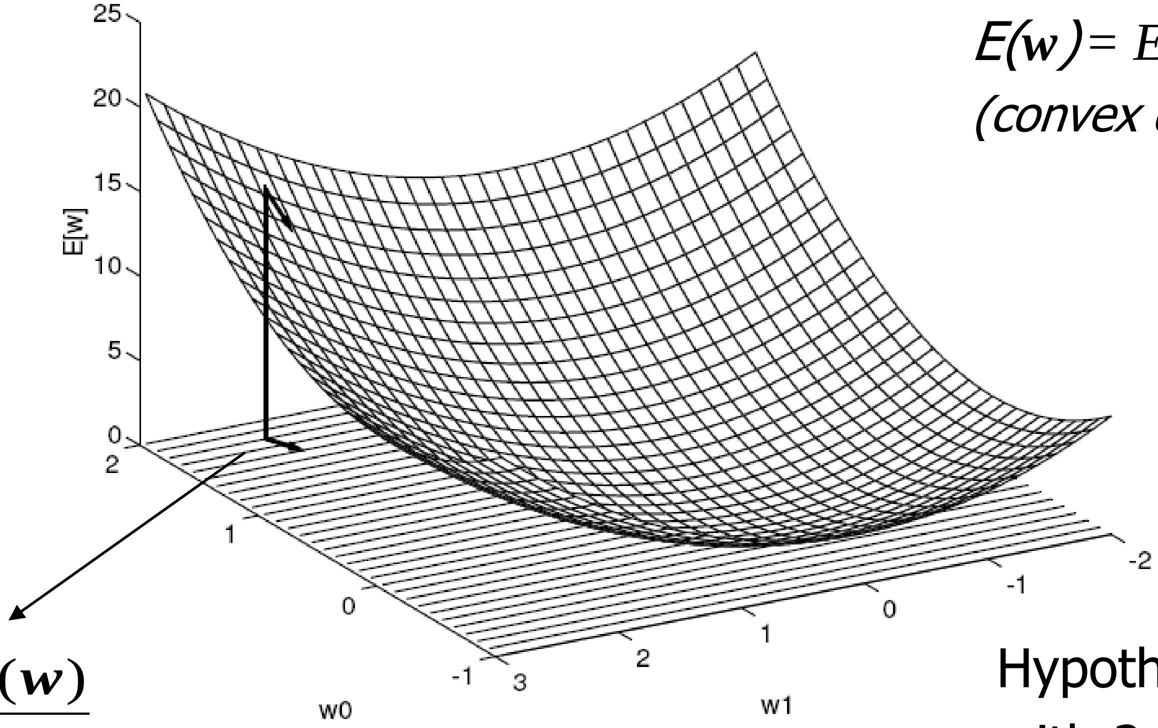
A single w at step 0,1,2,3,4

Error surface for linear model with 2 weights (*w*)

Parabolic for the

$E(\mathbf{w}) = E([w_0, w_1]^T)$

(convex quadratic function)



$$-\frac{\partial E(\mathbf{w})}{\partial \mathbf{w}}$$

Our "compass" to find the minimum

Hypothesis space with 2 parameters

w_0, w_1

The gradient vector

$$\Delta \boldsymbol{w} = -\frac{\partial E(\boldsymbol{w})}{\partial \boldsymbol{w}} = \begin{bmatrix} -\frac{\partial E(\boldsymbol{w})}{\partial w_1} \\ -\frac{\partial E(\boldsymbol{w})}{\partial w_2} \\ -\frac{\partial E(\boldsymbol{w})}{\partial w_j} \\ \dots \\ -\frac{\partial E(\boldsymbol{w})}{\partial w_n} \end{bmatrix} = \begin{bmatrix} \Delta w_1 \\ \Delta w_2 \\ \Delta w_j \\ \dots \\ \Delta w_n \end{bmatrix}$$

We can work in a multi-dim space without the need to visualize it
P.S. w_0 is omitted above

Using the Delta Rule

- Hence, as iterative approach we will move using a learning rule based on a «delta» (changing) of $\boldsymbol{w}$ proportional to the (opposite) of the local gradient
- The «movements» will be made iteratively according to

$$\boxed{\boldsymbol{w}_{\text{new}} = \boldsymbol{w} + eta * \Delta \boldsymbol{w}}$$ (or componentwise, i.e. for each w_j)

- that is the "***learning rule***"
- and *eta* (η) is the "step size" (learning rate) parameter (ruling the speed of our gradient descending)

Gradient descent algorithm

A simple algorithm:

1) Start with weight vector $\boldsymbol{w}_{\text{initial}}$ (small), fix *eta* $(0<eta<1)$.

2) Compute $\Delta \boldsymbol{w}$= -“gradient of $E(\boldsymbol{w})$” = - $\dfrac{\partial E(\boldsymbol{w})}{\partial \boldsymbol{w}}$ (or for each w_j)

3) Compute $\boxed{\boldsymbol{w}_{\text{new}} = \boldsymbol{w} + eta * \Delta \boldsymbol{w}}$ (or for each w_j)

Repeat (2) until convergence or $E(\boldsymbol{w})$ is “sufficiently small”

- $\Delta w/l$: *least mean squares* (dividing by l, that will be the standard case)
- Batch versions ($\Delta \boldsymbol{w}$ after each “epoch” of l training patterns)

eta (η): step size = *learning rate*: speed/stability trade-off: can be (gradually) decreased to zero (guarantee convergence, avoiding oscillation around the min.): many variants will be introduced later

Batch/On-line

- For **batch version** the gradient is the sum over all the *l* patterns:

$$\frac{\partial E(\boldsymbol{w})}{\partial w_j} = -2\sum_{p=1}^{l}(y_p - \boldsymbol{x}_p{}^T\boldsymbol{w})\, x_{p,j}$$

  - provide a more "precise" evaluation of the gradient over a set of *l* data

  And we upgrade the weights <u>after</u> this sum

- For the **on-line/stochastic version** we upgrade the weights with the error that is computed <u>for each pattern</u>
  - hence, the 2nd pattern output is based on weights already updated from the 1st, and so ahead
  - It makes progress with each examples it looks at: it can be the <u>faster</u>, but need smaller *eta:*

$$\frac{\partial E_p(\boldsymbol{w})}{\partial w_j} = -2(y_p - \boldsymbol{x}_p{}^T\boldsymbol{w})x_{p,j} = -\Delta_p w_j$$

- We will see intermediate cases later (as **mini-batch**)

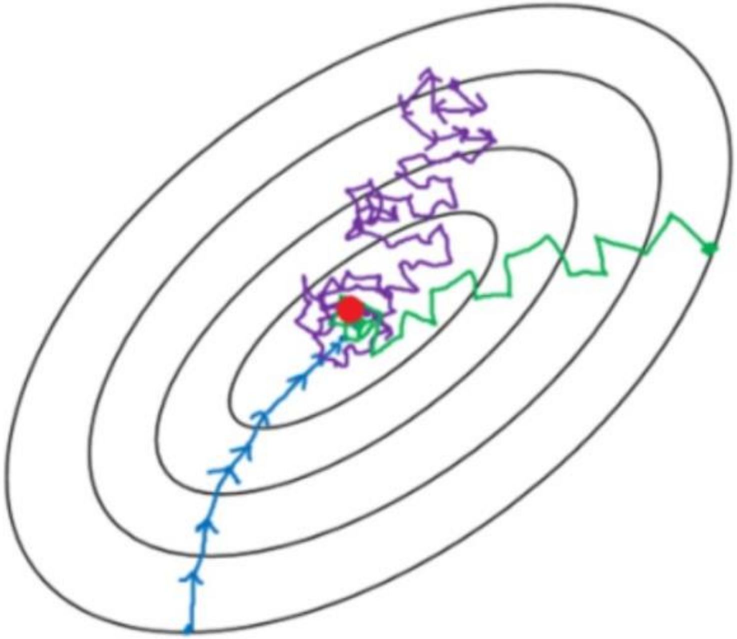
Examples

Batch algorithm

We update ***w*** after (repeating) an "epoch" of l training data → *(blu)*

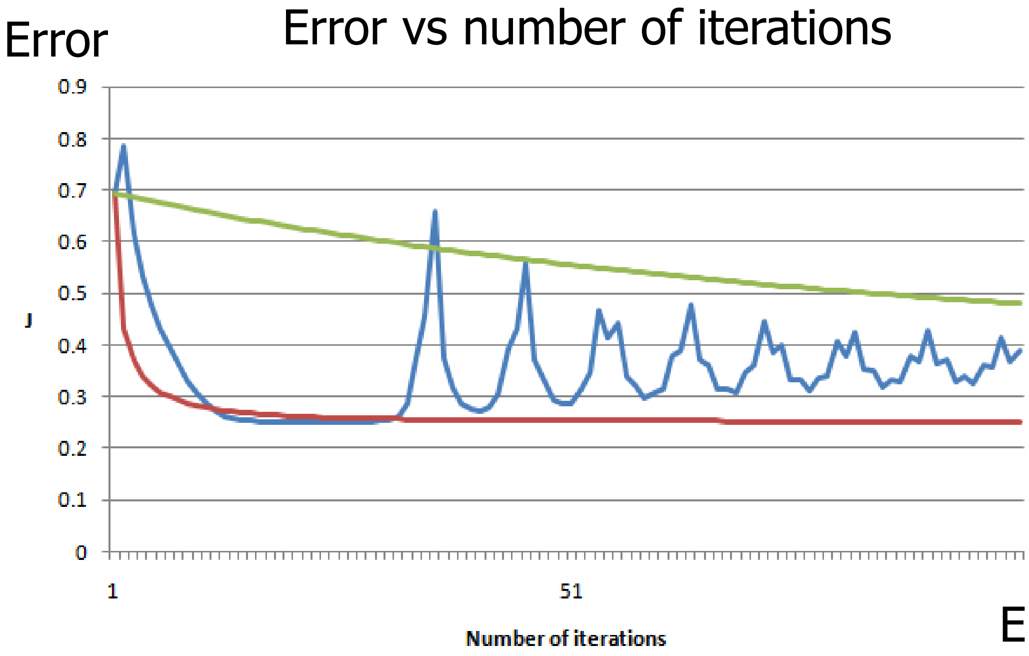
On-line algorithm (stochastic gradient descent - *SGD*)

We update ***w*** after each pattern *p* ($\Delta_p \mathbf{w}$ *for each pattern*→ (purple and green)



Paths over the error surface by Batch or On-line version

Learning curve examples



Epochs

These are **learning curves**:
They show how the error decreases through gradient descent iterations

P.S. No relation with color in the previous slide!

Exercise: *1 is slow, 1 is unstable, 1 is good: which one?*

Gradient descent as Error correction delta rule

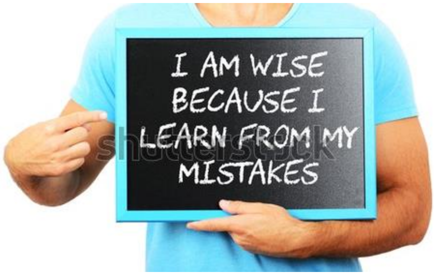
$$\Delta w_j = 2\sum_{p=1}^{l} (y_p - \boldsymbol{x}_p{}^T \boldsymbol{w})\, x_{p,j}$$

$$\boldsymbol{w}_{\text{new}} = \boldsymbol{w} + eta * \Delta \boldsymbol{w}$$

*Where $x_{p,j}$ is the component j of the input pattern p, y_p the output for p, **w** free par., l num. of examples.*
The constant 2 can be omitted

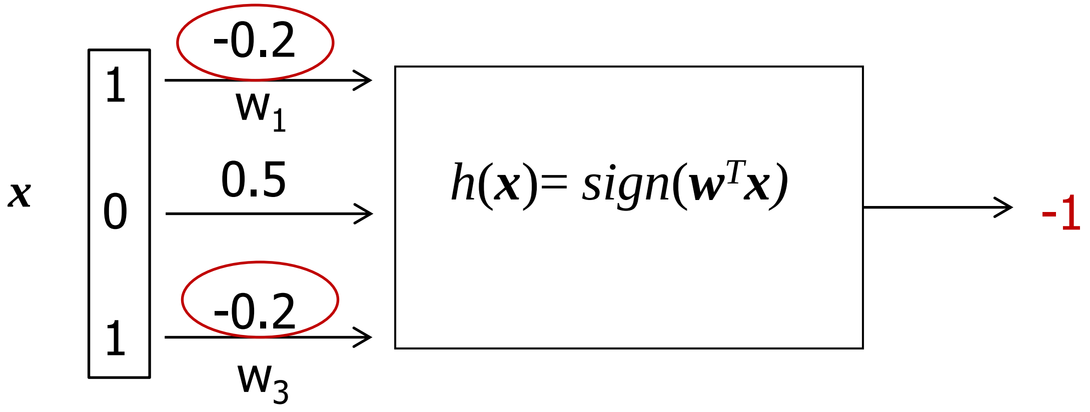
- This is an *"error correction" rule* (Widrow-Hoff or **delta rule**) that change each w_j proportionally to the error (target y - output):
  - E.g. (target y – output) = err=0 → no correction
  - (input$_j$>0) if err + (output is too low), positive delta → increase w_j → increment the output → less err
  - (input$_j$>0) if err - (output is too high), negative delta → decrease w_j → reduce output → less err
  - … [*exercise*: all the cases]
- We improve by learning from previous errors

  "seeking and blundering we learn (Goethe)"



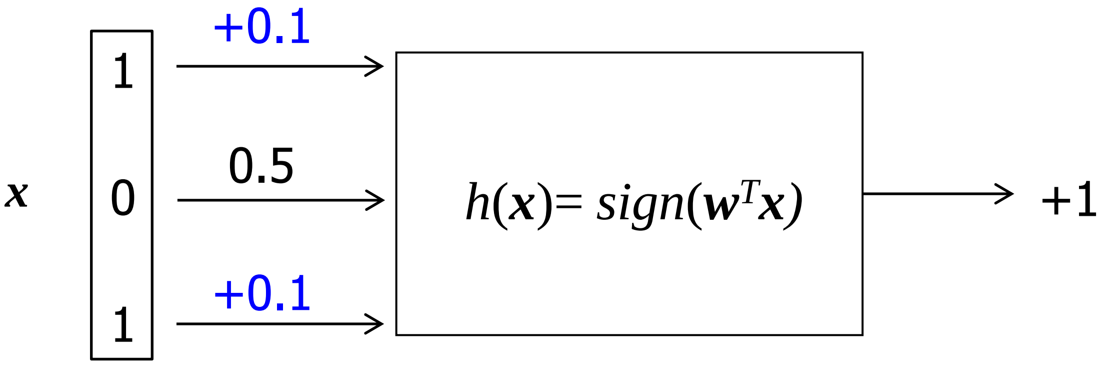
Delta-W as Error Correction Learning rule (II)

Inputs Weights $(w_0=0)$



If misclassified (because target is +1) → (1-(-???)) → High positive delta for w_1 and w_3 → increase them proportionally (with eta) to the *delta*, hence a positive value in this case [*error correction rule*] *!!!*

e.g. (see figure):



Now is correct !!!

__Exercise__: compute the values for delta (hint: above) and for also eta

Gradient descent: final discussion

Gradient descent approach it is simple and effective *local search* approach to LMS solution and

- It allows us to search through an *infinite (continuous) hypothesis* space!
- It can be easily always applied for *continues* H and *differentiable loss*
- NOT ONLY to linear models !!!! (we will see for Neural Networks and deep learning models)
-
- *Efficient*? Many improvement are possible, e.g. Newton & quasi-newton methods; Conjugate Gradient, …! → CM course*

Summarizing

- Model trained (on TR set) with LS (LMS) on ***wx***
  - by the simple gradient descent algorithm used for linear regression

- Model used for classification applying a threshold function, obtaining $h(\mathbf{x}) = sign(\mathbf{w}^T\mathbf{x})$

- The error can be computed as *classification error* or number of misclassified patterns (not only by the Mean Square Error)

$$\boxed{L(h(\mathbf{x}_p), d_p) = \begin{cases} 0 & \_if \quad h(\mathbf{x}_p) = d_p \\ 1 & \_\_\_\_\_\_\_ otherwise \end{cases}}$$

$$mean\_err = \frac{1}{l}\boxed{\underbrace{\sum_{p=1}^{l} L(h(\boldsymbol{x}_p), d_p)}_{num\_err}}$$

- **ACCURACY** = mean of correctly classified = $(l\text{-}num\_err)/l$

Coming back to the problem: Linear model solution

Dip. Informatica
University of Pisa

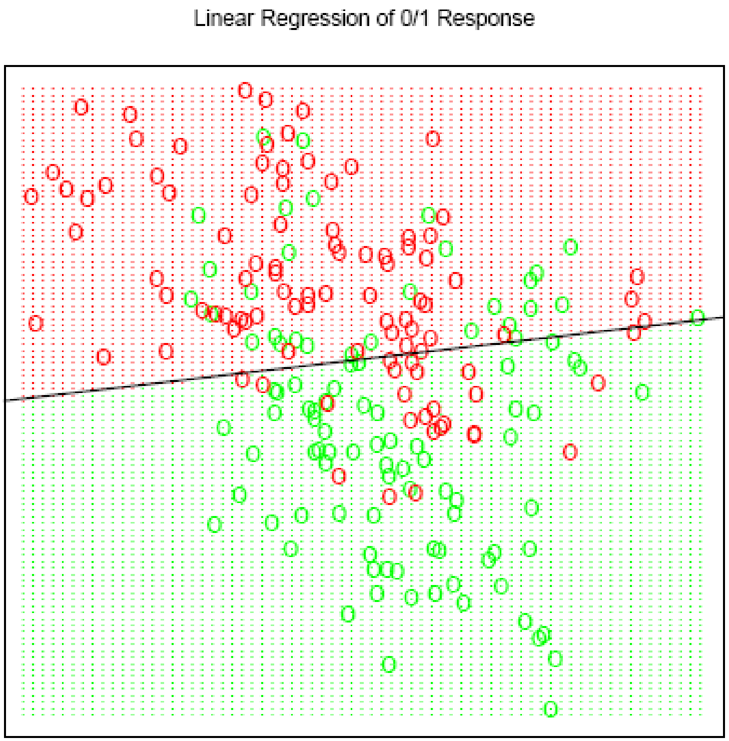


Figure 2.1 (© HTF 2001):

A classification example in two dimensions. The classes are coded as a binary variable
GREEN = 0, RED = 1
and then fit by linear regression. The line is the decision boundary defined by
$\boldsymbol{x}^T\boldsymbol{w} = 0.5$.
The red shaded region denotes that part of input space classified as RED, *while the green region is classified as* GREEN.

$$h(\boldsymbol{x}) = \begin{cases} 1 & \_if \quad \boldsymbol{x}^T\boldsymbol{w} > 0.5 \\ 0 & \_\_\_\_\_\_otherwise \end{cases}$$

Linear threshold unit

The *decision boundary* is {***x*** | ***x****ᵀw* = 0.5 } is linear (and seems to make many errors on the training data). Is it true?

Good or bad approximation? (#)

- Possible scenarios (we know the true target function!)
  - Scenario 1: The data in each class are generated from a Gaussian distribution with uncorrelated components, same variances, and different means.
  - Scenario 2: The data in each class are generated from a mixture of 10 gaussians in each class.

- For Scenario 1, the linear regression rule (by LS) is almost **optimal** (is the best one can do). The region of overlap is inevitable (due to errors in the input data).
- For Scenario 2, it is far too rigid: next models for it!

* Least squares corresponds to the maximum likelihood criterion if the experimental errors have a normal distribution

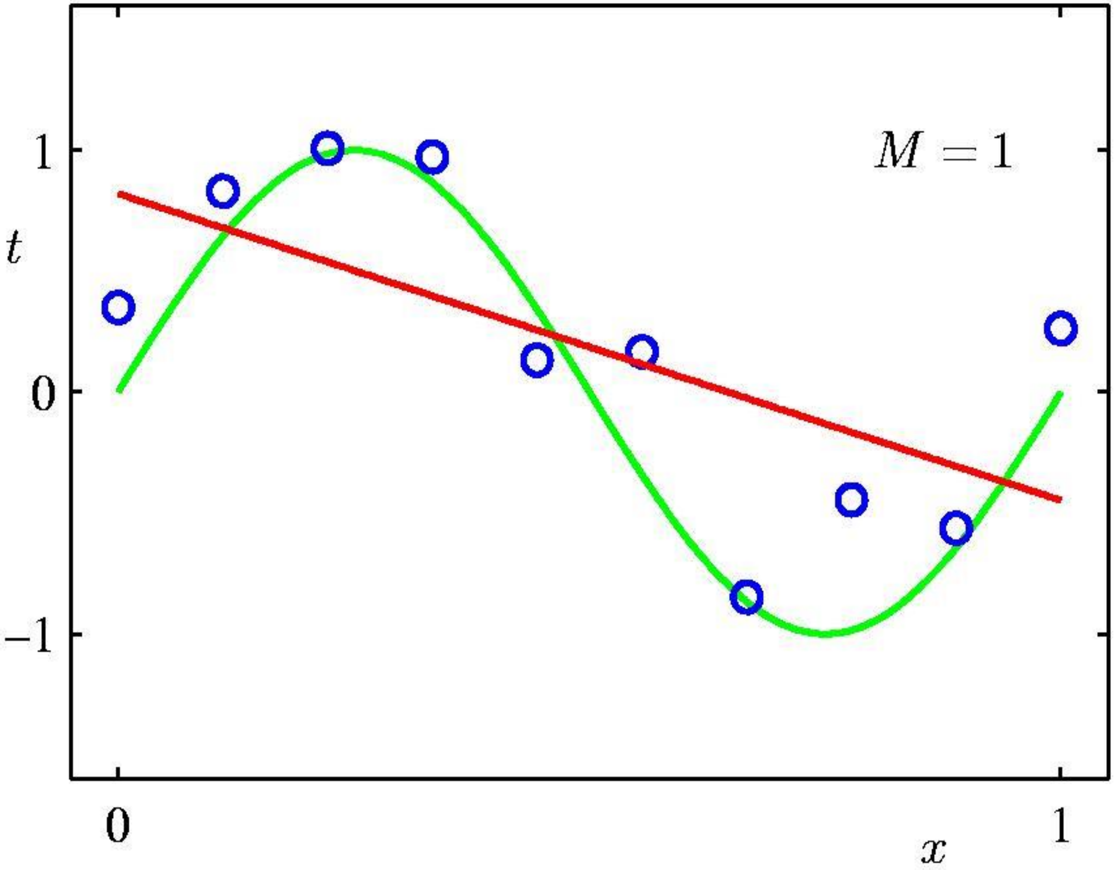
Linear model (in ML): Inductive Bias (alla Mitchell)

- Language bias: the H is a set of linear functions (may be very restrictive and rigid)

- Search bias: ordered search guided by the Least Squares minimization goal
  - For instance, we could prefer a different method to obtain a restriction on the values of parameters, achieving a different solutions with other properties (in particular to consider the generalization issue), …

It shows that even for a "simple" model there are many possibilities. We need a principled approach! (see theory of ML)…

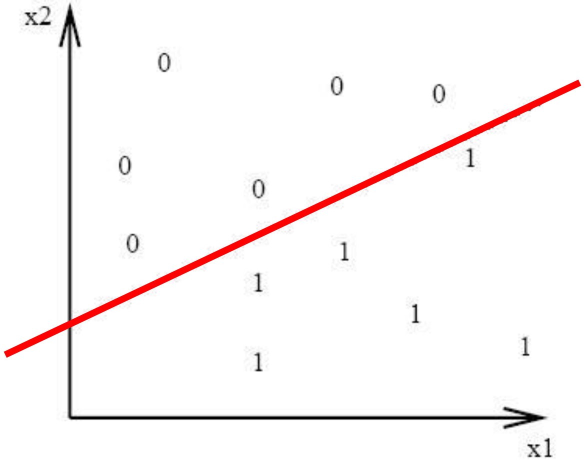
Limitations: regression tasks for non linear problems



Too poor solution

Limitations (classification) (language bias)

- In geometry, two set of points in a two-dimensional plot are ***linearly separable*** when the two sets of points can be completely separated by a single line
- In general, two groups are *linearly separable* in *n*-dimensional space if they can be separated by an (*n* – 1)-dimensional hyperplane.



- The linear decision boundary can provide exact solutions only for linearly separable sets of points

Example: Conjunctions

Dip. Informatica
University of Pisa

- We can represent conjunctions by the linear models, e.g.:
- **Conjunctions (**see the example in the introduction lectures**):**

4 var.: $x_1 \wedge x_2 \wedge x_4 \leftarrow\rightarrow y$

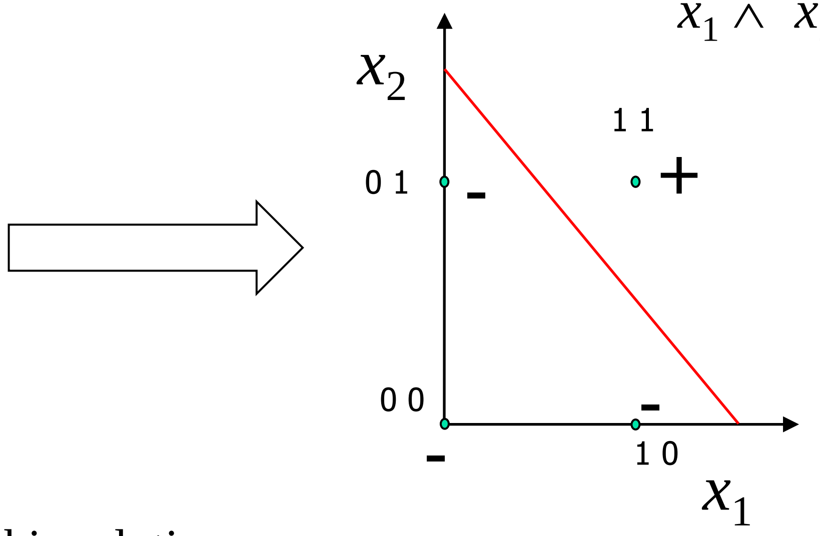
- $1\ \ x_1 + 1\ \ x_2 + 0\ \ x_3 + 1\ \ x_4 \geq 2.5$

$$h(\boldsymbol{x}) = \begin{cases} 1 & \_if \quad \boldsymbol{w}^T\boldsymbol{x} + w_0 \geq 0 \\ 0 & \_\_\_\_\_\_otherwise \end{cases}$$

In the plot:

2 var.: $x_1 \wedge x_2 \leftarrow\rightarrow y$

- $1\ \ x_1 + 1\ \ x_2 \geq 1.5$



$x_1 \wedge x_2$	AND
00	0
01	0
10	0
11	1

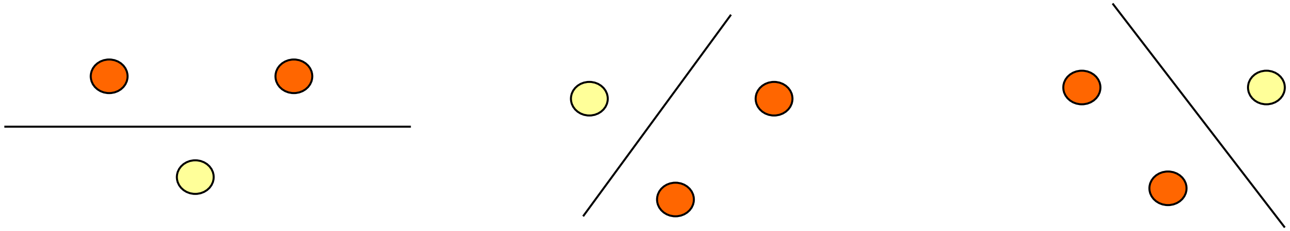
w can be learned to find this solution

Limitations: classification tasks

- Given **3 points**, can we always find a separation plane for every assignment of *f(**x**)*?

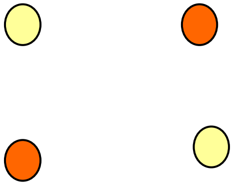
  No, 3 aligned points with 0 in the middle and others 1; yes if they are not aligned (existence!).

Here NOT complete:
There are 2^3 cases



- Given **4 points**, can we always find a separation plane for every assignment of *f(**x**)* ?
  No (XOR)

  we can find a labeling such that the linear classifier fails to be perfect



XOR	
00	0
01	1
10	1
11	0

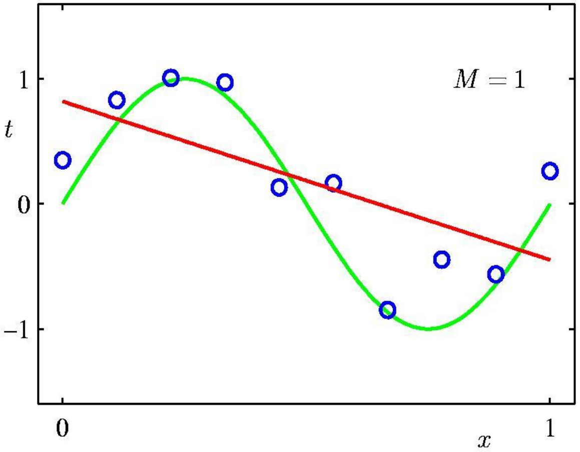
***Exercise**: try to redo it by yourself!*

How to extend the linear model

Dip. Informatica
University of Pisa

- Note that in $h_{\mathbf{w}}(x)=w_1x+w_0$ or $h_{\boldsymbol{W}}(\boldsymbol{x}) = \boldsymbol{w}^{\mathrm{T}} \cdot \boldsymbol{x}$
- As Statistical Parametric models:

  "linear" does not refer to this (red) straight line, but rather to the way in which the regression coefficients ***w*** occur in the regression equation



- Hence, we can use also transformed inputs, such are $x, x^2, x^3, x^4, \ldots$. with *non-linear* relationship inputs and output, holding the learning machinery (Least Square solution) developed so far…

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

polynomial regression

A generalization (LBE)

(shown for regression*)

- Basis transformation: **linear *basis expansion* (LBE):**

$$h_{\boldsymbol{w}}(\boldsymbol{x}) = \sum_{k=0}^{K} w_k \phi_k(\boldsymbol{x})$$

- Augment the input vector with additional variables which are transformations of x according to a function phi (ϕ_k: $R^n \rightarrow R$)
- E.g
  - Polynomial representation of $\mathbf{x}$: $\phi(\mathbf{x}) = x_j^2$ or $\phi(\mathbf{x}) = x_j x_i$, or ....
  - Non-linear transformation of single inputs: $\phi(\mathbf{x}) = log(x_j)$, $\phi(\mathbf{x}) = root(x_j)$, ...
  - Non-linear transformation of multiple input: $\phi(\mathbf{x}) = ||\mathbf{x}||$
  - ***Splines, …, ….***
- Number parameters $K > n$ (before it was n)
- The model is *linear in the parameters (also in phi, not in* $\mathbf{x}$*): we can use the* ***same learning alg.*** *as before!*
- *Note: it can be applied for regression (here) or classification (Exercise: HOW?)*

Basis Expansion: examples

- Basis transformation: **linear *basis expansion* :**

$$h_{\mathbf{w}}(\boldsymbol{x}) = \sum_{k=0}^{K} w_k \phi_k(\boldsymbol{x})$$

EXAMPLES:

- *[1-dim **x**]* $\phi_j(x) = x^j.$

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

1-dim polynomial regression (K=M)

Already seen in the introduction

- Or *"any other", e.g.* $\phi(\mathbf{x})=\phi([x_1,x_2,x_3])$

$$h(\mathbf{x}) = w_1x_1 + w_2x_2 + w_3 log(x_2) + w_4 log(x_3) + w_5(x_2x_3) + w_0$$

Basis Expansion criticism

- Which phi (ϕ) ? Toward the so called “**dictionary**” approaches

- PROS: Can model more complicated relationships (than linear): it is more expressive.
- CONS: With *large* basis of functions, we easily risk *overfitting*, hence we require methods for *controlling the complexity* (*) of the model
  - *Curse of dimensionality* (the volume of the problem space increases so fast that the available data become sparse, the data became no more sufficient to support the model complexity) → we will see later
  - Phi are *fixed before* observing training data
    - versus adaptive /non-linear in parameters e.g. NN!

- *We will see the alternative NN and SVM solutions!*

(*) Whereas **complexity** is **not** for the computational cost but a measure of the flexibility of the model to fit the data (see the VC-dim)

Improvements: How to control model complexity? Important !

- Many approaches… e.g. coefficient shrinkage:
- **Ridge regression**: (**Tikhonov regularization**): smoothed model
  → possible to add constraints to the sum of value of $|w_j|$ penalizing models with high values of $|w|$, i.e. favoring "sparse" models using less terms due to weights $w_j = 0$ (or close to 0) (it means a less complex model)

“Error” data term [(M)SE] Regularization/penalty term

$$Loss(\mathbf{w}) = \sum_{p=1}^{l}(y_p - \boldsymbol{x}_p{}^T\mathbf{w})^2 + \boxed{\lambda||\mathbf{w}||^2} \longrightarrow \sum w_j{}^2$$

Lambda (λ): regularization (hyper)parameter
(a small positive value chosen by the “model selection” phase)

The sum is over the number of w

- Note that for the ***objective function*** we use here the name *Loss* (used for the model training cost function) to distinguish from the *Error E* (useful to evaluate the model error and used for the *data term* inside this Loss).
  Hence, differently from previous assumptions the two terms are not more equivalent in our use (in the course).

Tikhonov Regularization: Solving it

$$Loss(\boldsymbol{w}) = \sum_{p=1}^{l} (y_p - \boldsymbol{x}_p{}^T \boldsymbol{w})^2 + \lambda ||\boldsymbol{w}||^2$$

– For the direct approach:

$$\mathbf{w} = \boxed{(\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}} \mathbf{X}^T\mathbf{y} \dashrightarrow$$

this matrix is always invertible

– For the gradient approach:

- As an **Exercise** compute the gradient of the two terms (error and penalty terms) of the *Loss*($\mathbf{w}$) w.r.t. weights w_i separately, using eta only for term *E*
- Formulate the new update rule:
- You will obtain: $\boldsymbol{w}_{\text{new}} = \boldsymbol{w} + eta^*\Delta\boldsymbol{w} - 2\lambda\boldsymbol{w}$
- That is a **weight decay** technique (basically add $2\lambda w$ to the gradient)
  - E.g. with 0 gradient, it decreases the value of each w with a fraction of the old w

Andrej Nikolaevič Tikhonov

Russian mathematician
1906 –1993

Tikhonov regularization: trade-off

$$Loss(\boldsymbol{w}) = \sum_{p=1}^{l}(y_p - \boldsymbol{x}_p{}^T\boldsymbol{w})^2 + \lambda||\boldsymbol{w}||^2$$

- Note the balancing (trade-off) between the two terms:
  - Small ***lambda*** (λ) value → minimizing the loss the focus is on obtaining a small error data term (first term, minimize just the training error) with a too complex model (high norm of the weights), the risk is of ***overfitting***,
  - High ***lambda*** (λ) → minimize the loss the focus is on the second term, hence the data error (first term) could grow too much, i.e. moving to ***underfitting***
  - The trade-off is ruled by the value of ***lambda*** (λ)
  - We will see soon some exampels

- The main advantage is that we have a concrete realization of the control of model complexity, easy to be implemented and of general applicability.

Tikhonov and SLT

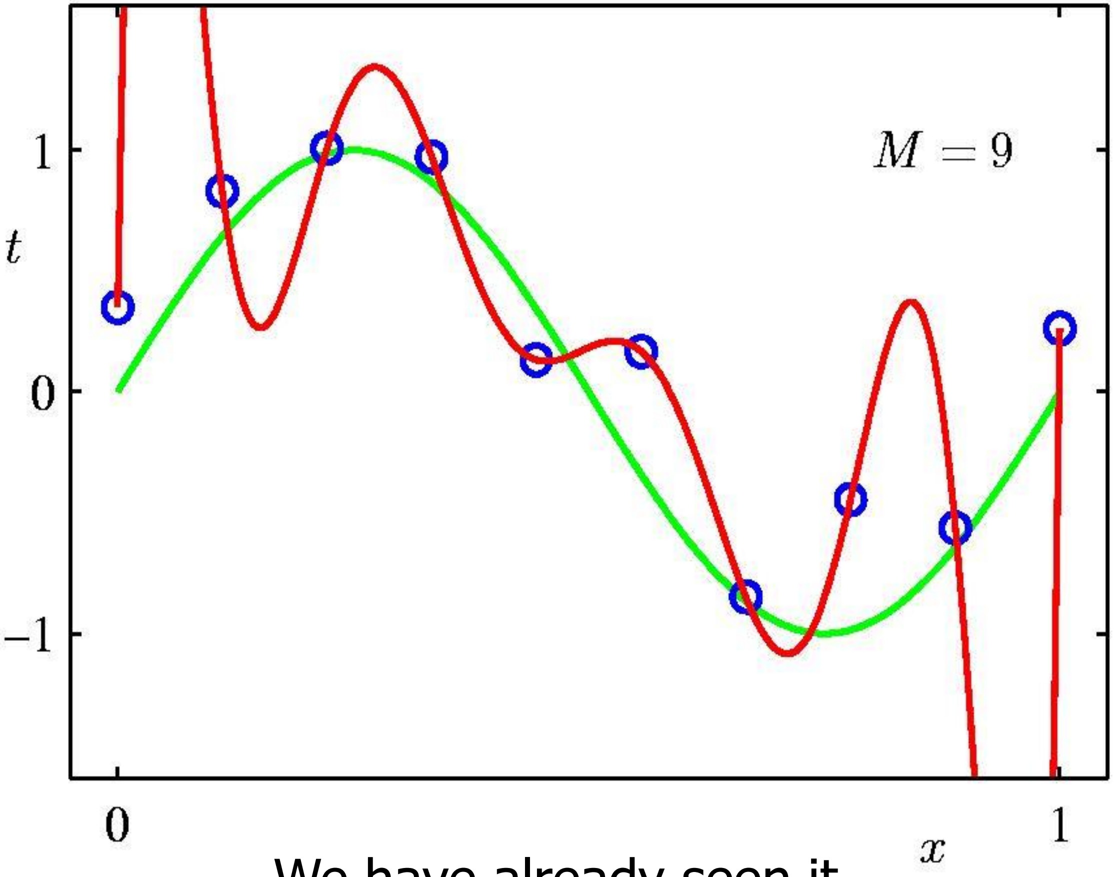
- The penalty term penalizes high value of the weights and tends to drive all the weights to smaller values
  - E.g. Some weights values can go even to zero
- It implements a control of the model complexity
- This leads to a model with less (or proper) *VC-Dim* (with a trade-off obtained through **just a (1) parameter** that you can control: the λ)

- **Exercise**: connect Tikhonov regularization with the slide on SLT (see the introduction) to see
  - why this can help to have a better bound on *R*, and
  - how lambda values can rule the underfitting/overfitting cases

- **Exercise:** derive the new learning rule with weight decay using the Tikhonov loss: computing again the partial derivatives of *Loss* with respect to w, separating data term (• eta) and penalty term (• λ)

Slide from Intro slides:
9th Order Polynomial

As to have Lambda 0



We have already seen it

E(***w***)= 0 on training data and **overfitting**

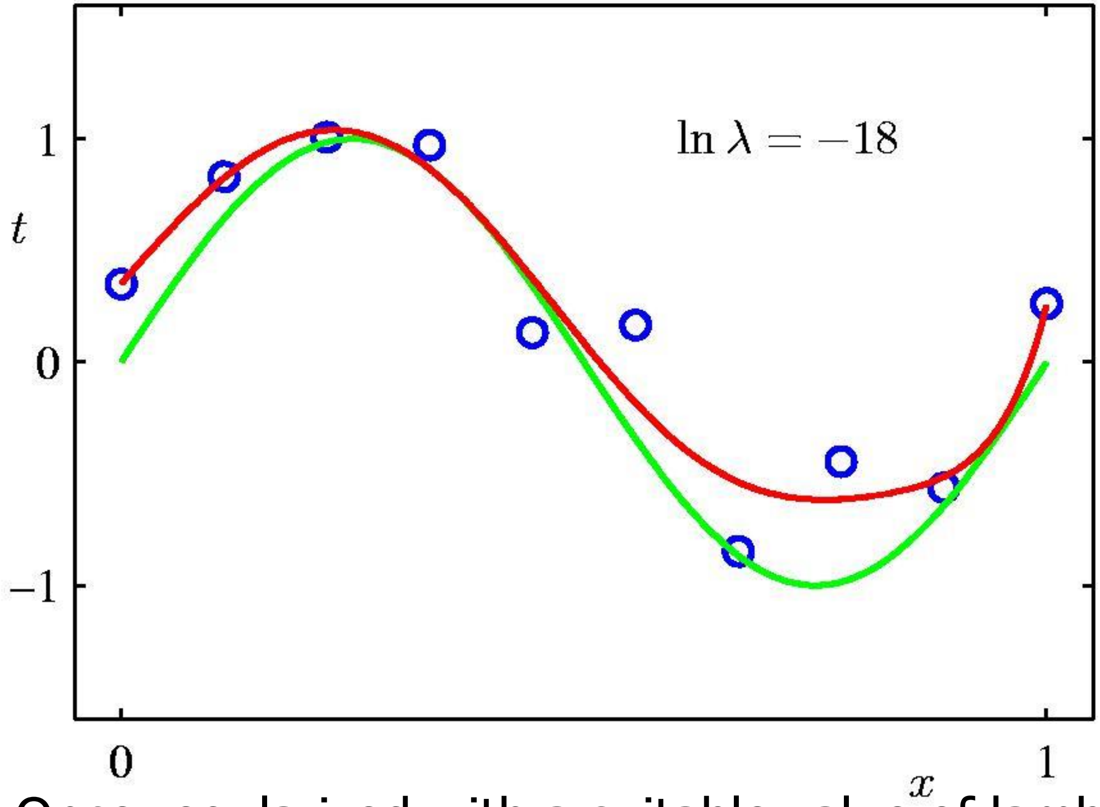
Regularization effect

$\ln \lambda = -18$

- 9^{th} Order Polynomial

Lambda small positive

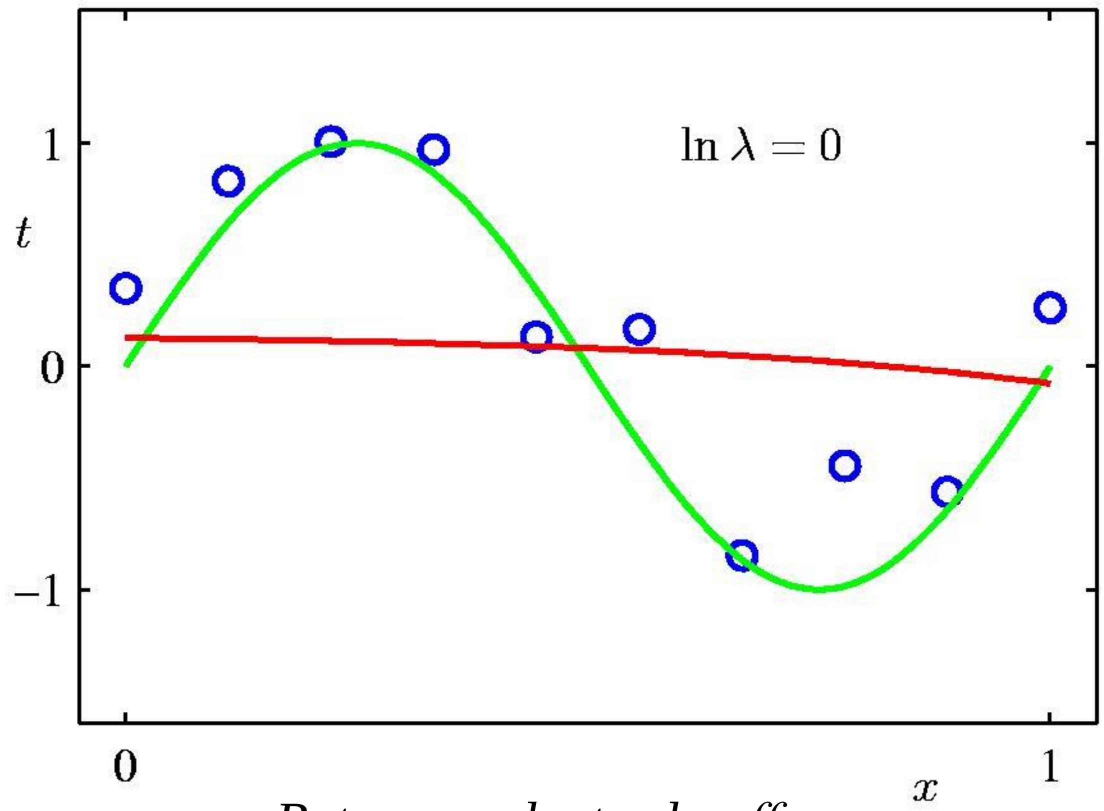
λ=0.0000000152



Once regularized with a suitable value of lambda
it works well! We avoid the overfitting

Too High lambda ($\ln \lambda = 0$)

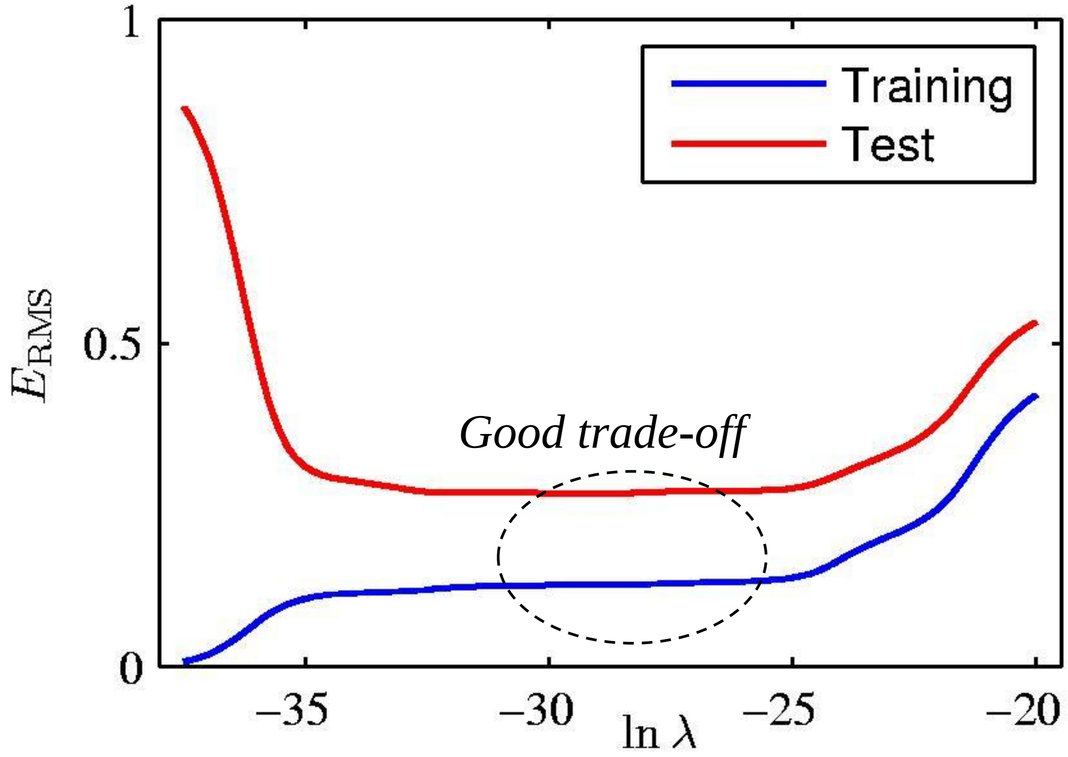
Very high lambda, close to 1



But we need a trade-off …

(because a too high lambda leads as to underfitting)

Regularization: E_{RMS} vs $\ln \lambda$



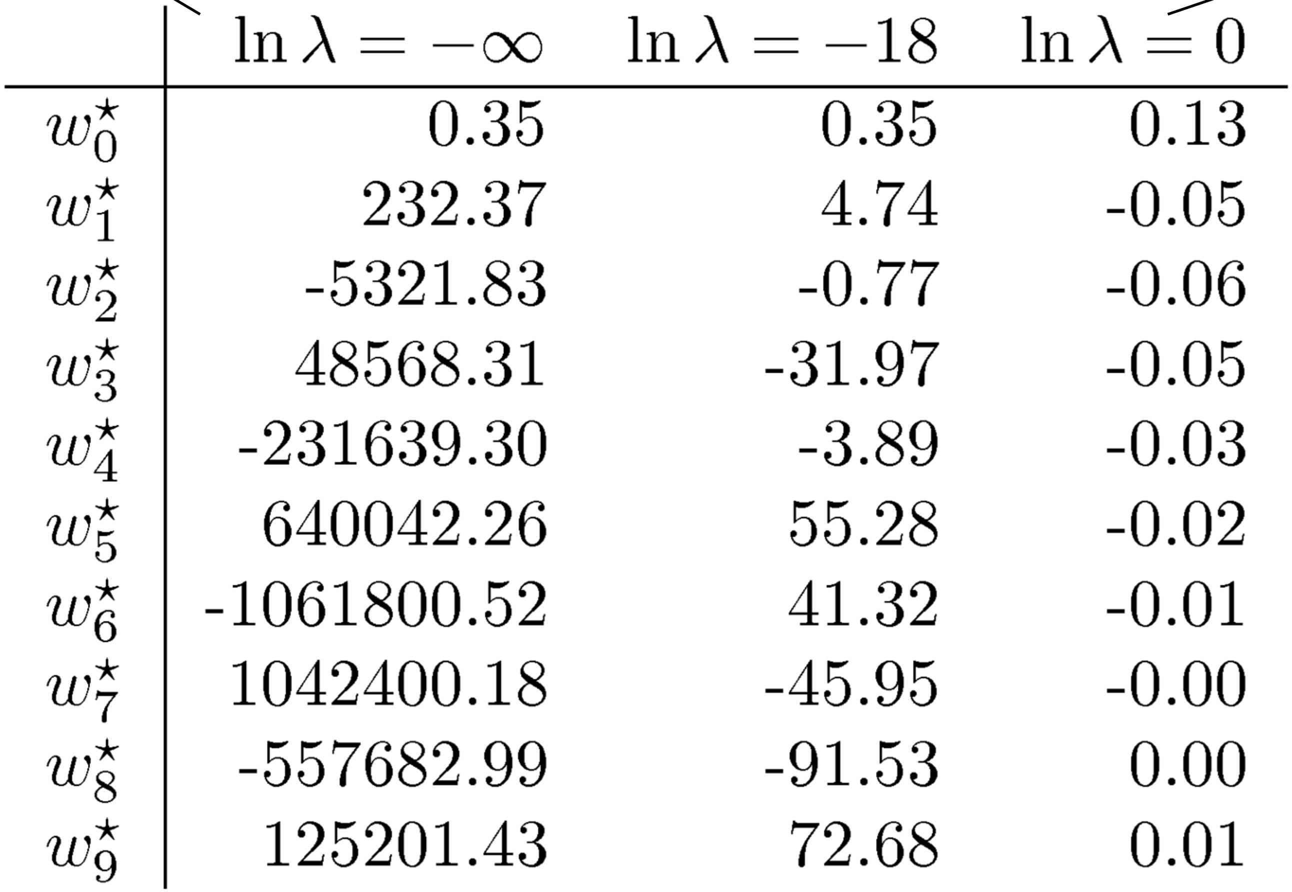
Low lambda → overfitting

High lambda → underfitting

Polynomial Coefficients

0 lambda

Very high lambda



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

Other Regularization Tech. for Linear Models

- Ridge regression ($\|\ \|_2$)
- Lasso ($\|\ \|_1$)
- Elastic nets (use both $\|\ \|_1$ and $\|\ \|_2$)

- We will introduce them later in the course (or in CM course), anyway:
- The *L2 norm* penalizes the square value of the weight and tends to drive all the weights to smaller values.
- On the other hand, the *L1 norm* penalizes the absolute value of the weights and tends to drive some weights to exactly zero (while allowing some weights to be large) → toward feature selection!
  - Unfortunately $\|\ \|_1$ (absolute value) introduce a non differentiable loss → needs other approaches

Improvements (cnt)

- Inputs with added noise (*data augmentation*)

- Derived inputs

  a small number of new variables is used in place of the x inputs, which are a linear combination of x.
  - Principal Component Regression
  - Partial Least Squares

Multi-class task (#)

Two very simple approaches for multi-class:

- Class 1-of-K rep: {red, green, blue} → (0,0,1), (0,1,0), (1,0,0). → solve 3 linear models
- **OVA:** "one-vs-all": a discriminant function for each class

built on top of real-valued binary classifiers

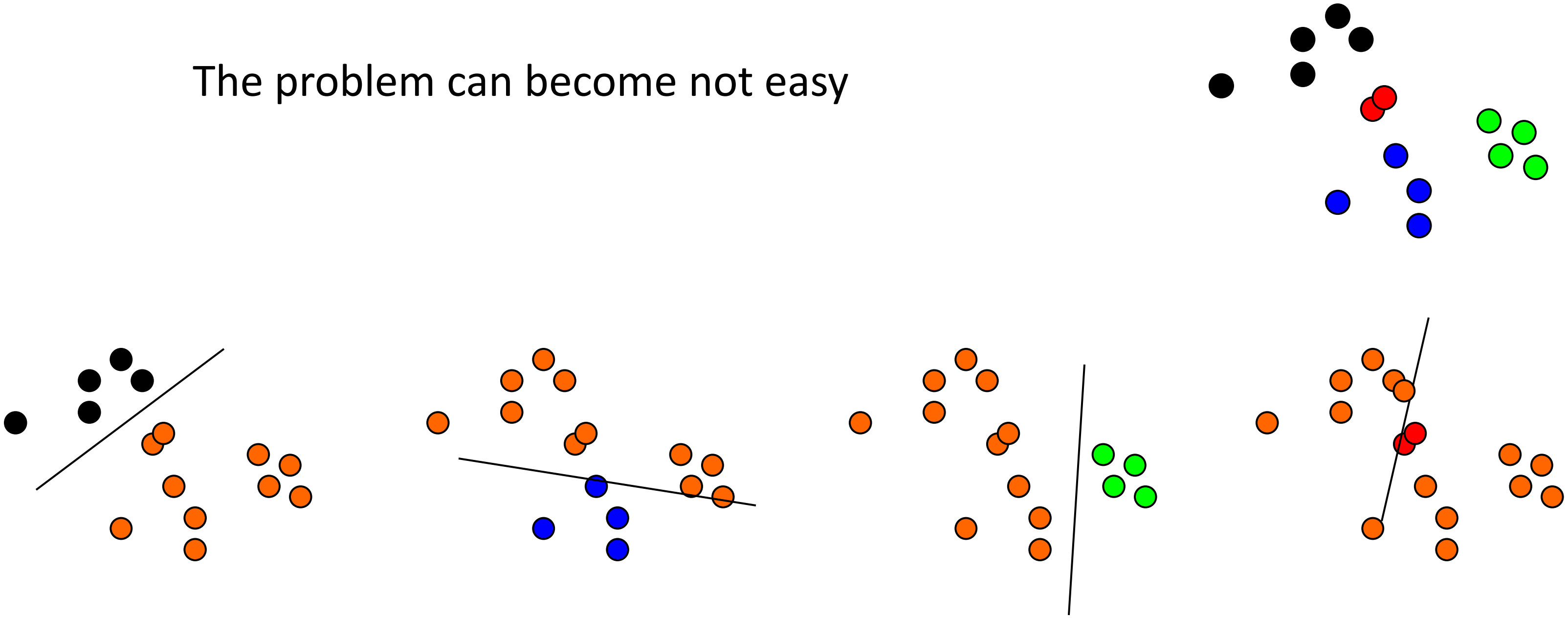
  - train K different binary classifiers, each one trained to distinguish the examples in a single class from the examples in all remaining classes.
  - to classify a new example, the K classifiers are run, and the classifier which outputs the largest (most positive) value is chosen.

- AVA: "all-vs-all" = "one-versus-one":
  - each classifier separates a pair of classes. Apply it to all the pairs: K(K-1)

    [How many training?*]
  - to classify a new example, all the classifiers are run, and the winner is the one with the max sum of outputs versus all the other classes **OR** the class with most votes
  - training data set for each classifier is much smaller

Criticism (##)

The problem can become not easy



- Masking: classes can be masked by others (for high K)
- A wide array of more sophisticated approaches for multiclass classification exists …
- Some models can deal directly with multi-output

Other learner models for classification (#)

- Linear Discriminant Analysis (also multi-class)

- Logistic regression
  - $P(y|\mathbf{x})$ starting from modeling the class density as a know density

Extensions: ML Models (pro future) (#)

In ML course:

- Perceptron (Rosenblatt 1958, biological inspiration):
  - "minimize (only) misclassifications" algorithm
  - basis for Neural Networks (set of units with layers)
  - Adaptive *basis expansions (phi learned by training)*
    - Feature representation learning in each layer (deep learning concept)
  - Gradient descent approach for learning
- SVM (Vapnik 1996):
  - *regularization* via the concept of maximum margin: Maximize the gap (margin) between the two classes on the training data.
  - enlarge the feature space via *basis expansions* (e.g. polynomials).
- NN and SVM models realize (also) a flexible ***non-linear*** approximation for classification and regression problems

Lessons Learned

- We concretely showed that is possible to *learn* by tuning free parameters ***w***, searching over a continuous space guided by a loss f.
  - How to state/formulate a regression/classification learning problem for a linear model by LMS
  - How to derive the learning algorithm (basic univariate, direct and **gradient** based forms)
  - Two learning algorithms
  - How the model works after training
  - Linear models have limitations (strong language bias)
- How to extend to non-linear modeling (by **LBE**)
- How to implement a **regularization approach** changing the loss f.
- The role of regularization for the **control of the model complexity**

Also useful for your self assessment (try to check if you can answer)

ML Course structure
Discussion on Linear Model

In the *file rouge* of ML, 2 main concepts up to now:

- Basis expansion → (implement) more flexibility
- Regularization → (implement) control of complexity

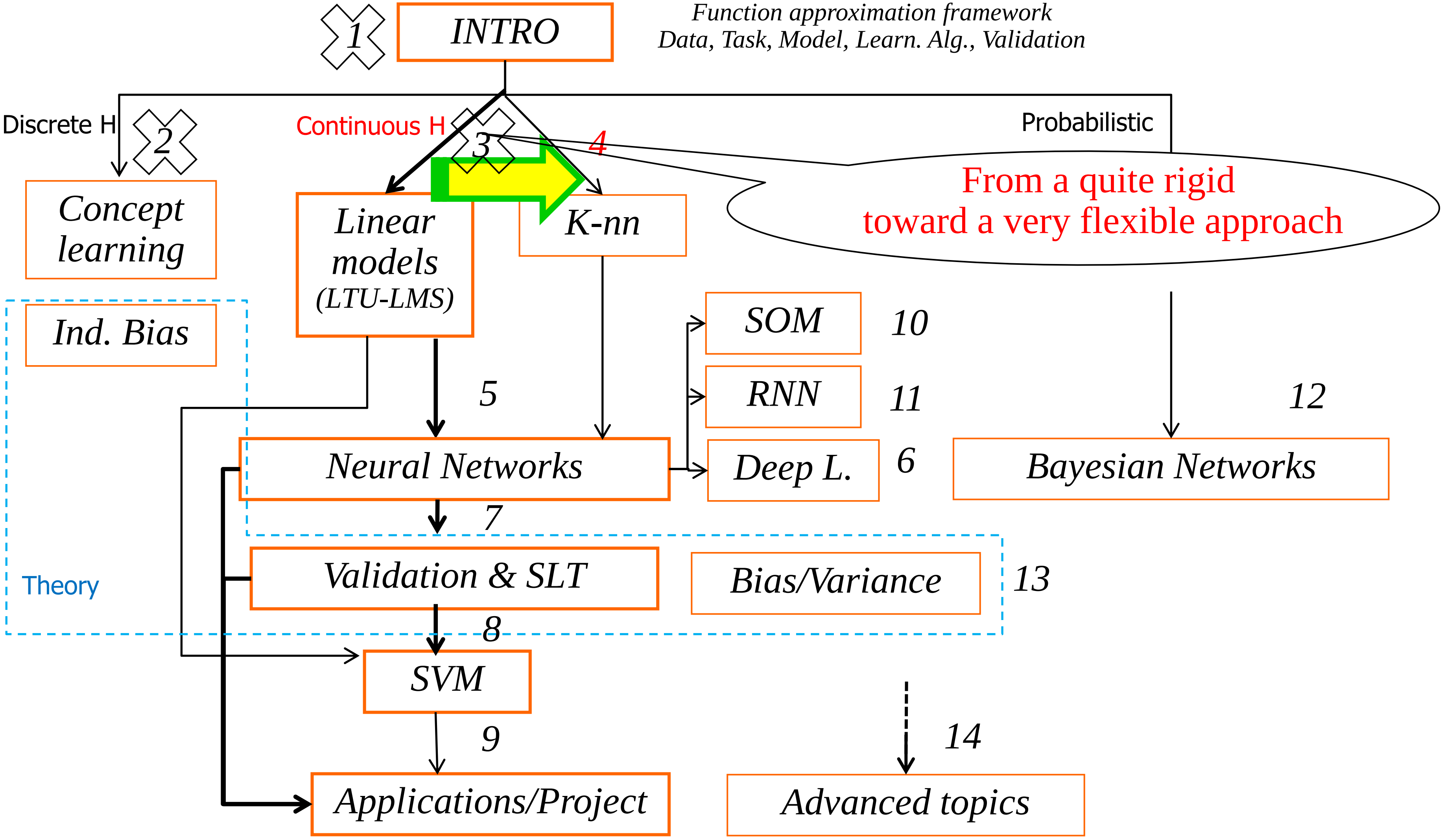
- The cases of the *linear models* provided an *instance* routed in the classical mathematical approaches but the we will find again these concepts, by different models and implemented in different/similar forms, in the modern ML (e.g. for NN & SVM in our course)!

- Now we move on the other extreme, toward a very flexible approach

K-nn

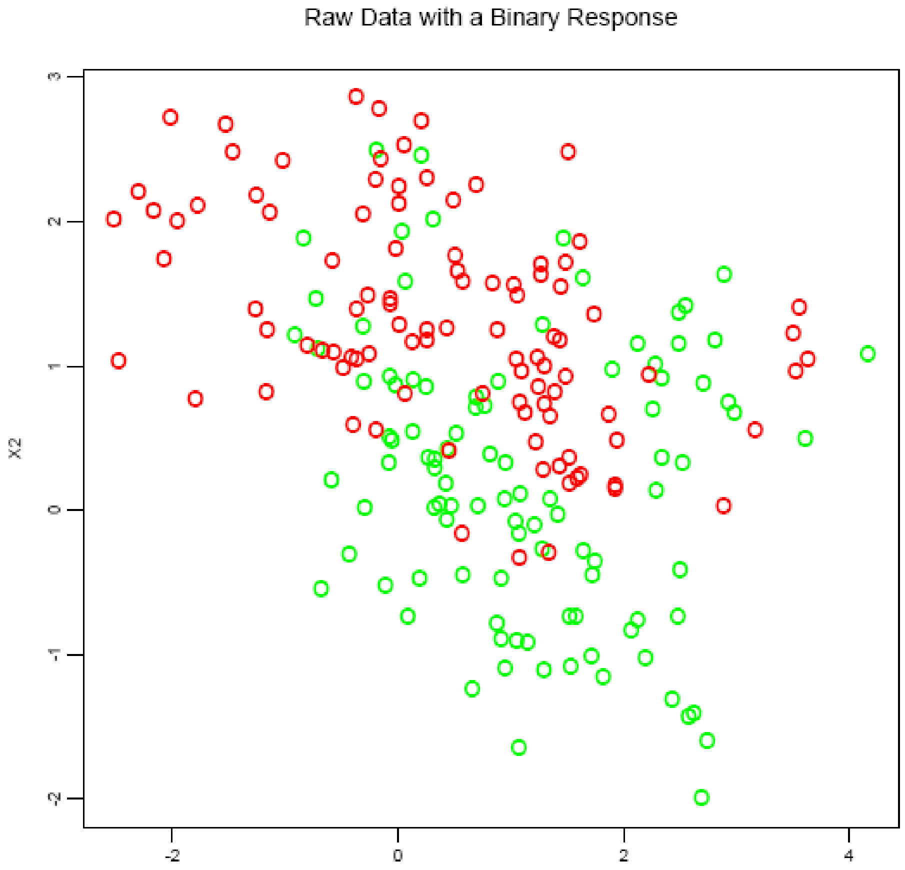
Still simple but flexible (and local)

ML Course structure
Where we go



Dip. Informatica
University of Pisa

Repetita: Problem: example



200 points generated in IR^2 from an unknown distribution; 100 in each of two classes.

Can we build a rule to predict the color of future points?

Data may be generated by gaussian distribution (for each class) with different means

or by a mixture of different low variance gaussian distributions.

Repetita: Find by LS a line to separate

Dip. Informatica
University of Pisa

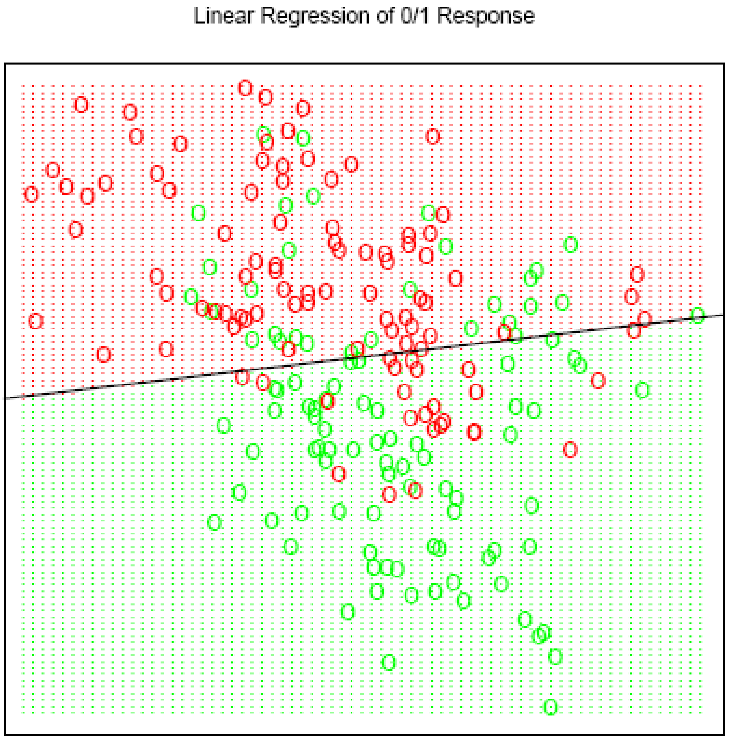


Figure 2.1 (© HTF 2001):

A classification example in two dimensions.
The classes are coded as a binary variable
GREEN = 0, RED = 1
and then fit by linear regression.
The line is the decision boundary defined by
$\boldsymbol{x}^T\boldsymbol{w} = 0.5$.
The red shaded region denotes that part of input space classified as RED*, while the green region is classified as* GREEN.

$$h(\boldsymbol{x}) = \begin{cases} 1 & \_if \quad \boldsymbol{x}^T\boldsymbol{w} > 0.5 \\ 0 & \_\_\_\_\_\_\_otherwise \end{cases}$$

Linear threshold unit

The decision boundary is {$\boldsymbol{x}$ | $\boldsymbol{x}^T w$ = 0.5 } is linear (and seems to make many errors on the training data). Is it true?

Learning... timing

- LEARNING ALG.

- Timing.
  - **Eager:** Analyze the training data and construct an explicit hypothesis.
  - **Lazy**: Store the training data and wait until a test data point is presented, then construct an ad hoc hypothesis to classify that one data point.

1-Nearest Neighbor

The algorithm:

- Simply store the training data $<\boldsymbol{x}_p, y_p>$ $p=1 \ldots l$
- Given an input $\boldsymbol{x}$ (with dimension n)
- Find the nearest training example $\boldsymbol{x}_{\boldsymbol{i}}$
    - Find $\boldsymbol{i}$ s.t. we have min $d(\boldsymbol{x},\boldsymbol{x}_i)$ → $i(\boldsymbol{x}) = \arg\min_p \; d(\boldsymbol{x}, \boldsymbol{x}_p)$
    - E.g. Euclidian distance :

$$d(\boldsymbol{x}, \boldsymbol{x}_p) = \sqrt{\sum_{t=1}^{n} \underset{\Big\downarrow \; \textit{Pattern } \mathbf{x_p}\textit{, component } t}{(x_t - x_{pt})^2}} = ||\boldsymbol{x} - \boldsymbol{x}_p||$$

- Then output $y_{\boldsymbol{i}}$

1-nn

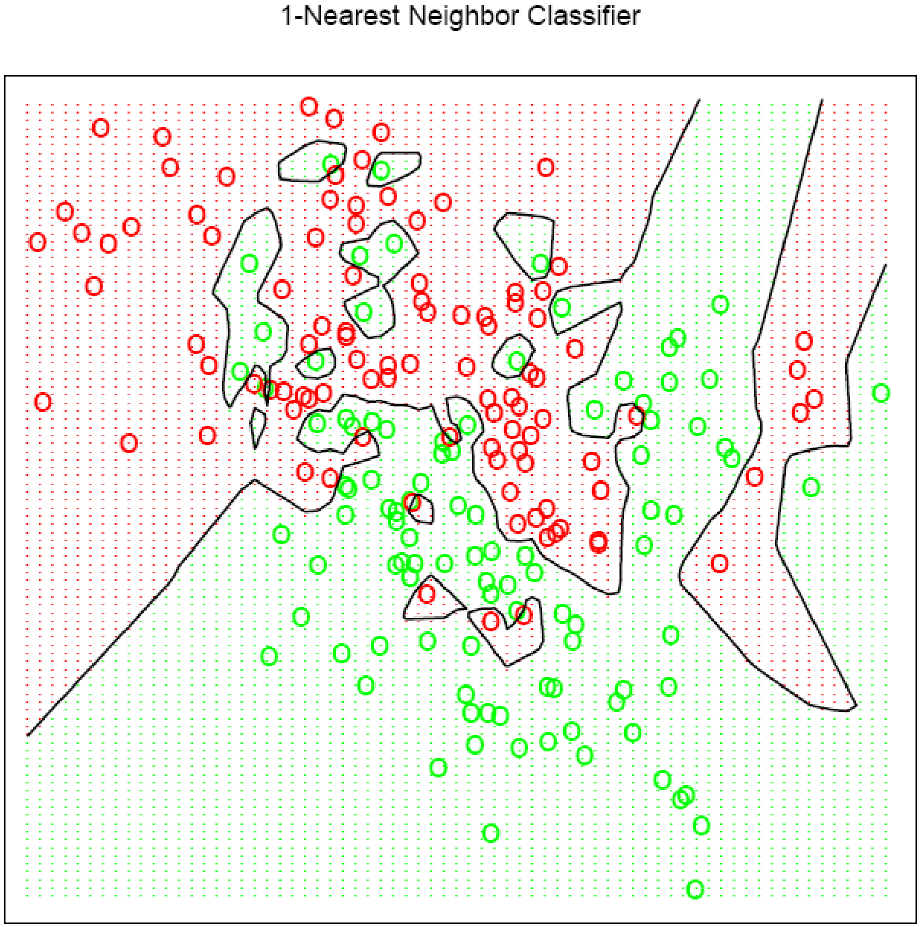
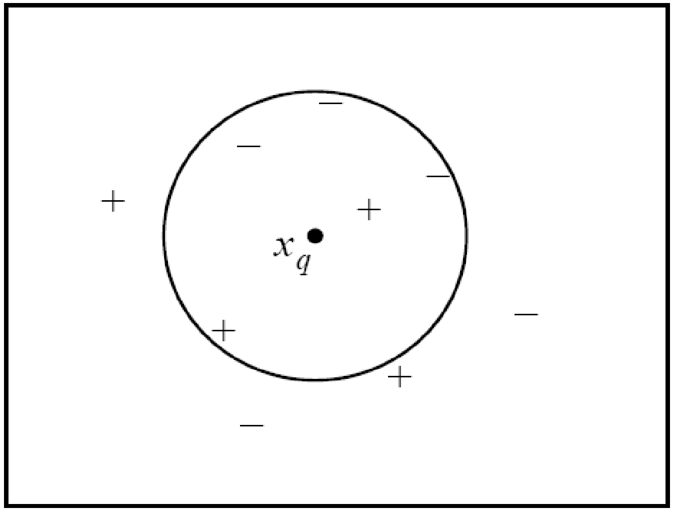


Figure 2.3:
The same classification example in two dimensions as in Figure 2.1.
GREEN = 0, RED = 1

- Very flexible !
- No misclassifications in TR data: 0 training error: what for test data ?
- Decision boundaries is *not* linear: it is quite *irregular*
- May be unnecessary noisy (e.g. for scenario 1)

1-nn vs 5-nn example

- 1-nn return + for x_q
- 5-nn return – for x_q



Smoothing over a set of neighbors for noisy data

K-Nearest Neighbors

- A natural way to classify a new point is to have a look at its neighbors,
- and take an average:

$$avg_k(\mathbf{x}) = 1/k \sum_{\mathbf{x}_i \in N_k(\mathbf{x})} y_i$$

- where $N_k(\mathbf{x})$ is a neighborhood of $\mathbf{x}$ that contains exactly k neighbors (closest patterns according to *distance d*):
  - k-nearest neighborhood: **K-nn.**

- If there is a clear dominance of one of the classes in the neighborhood of an observation ***x***, then it is likely that the observation itself would belong to that class, too. Thus the classification rule is the **majority voting** among the members of $N_k(\mathbf{x})$. As before,

$$\boxed{h(\mathbf{x}) = \begin{cases} 1 & \_if \quad avg_k(\mathbf{x}) > 0.5 \\ 0 & \_\_\_\_\_\_otherwise \end{cases}} \qquad for\ targets\ y\ \ in\ \{0{,}1\}$$

- For regression task: use directly the *avg*: mean over K-nn

15-nn

15-Nearest Neighbor Classifier

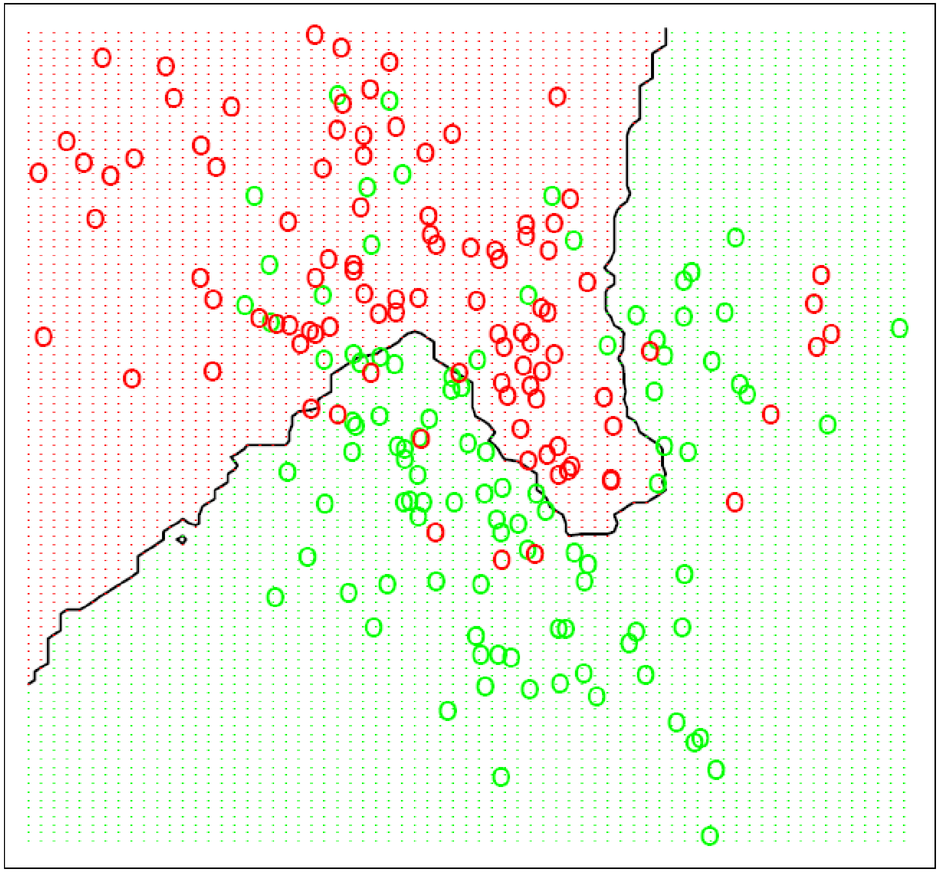
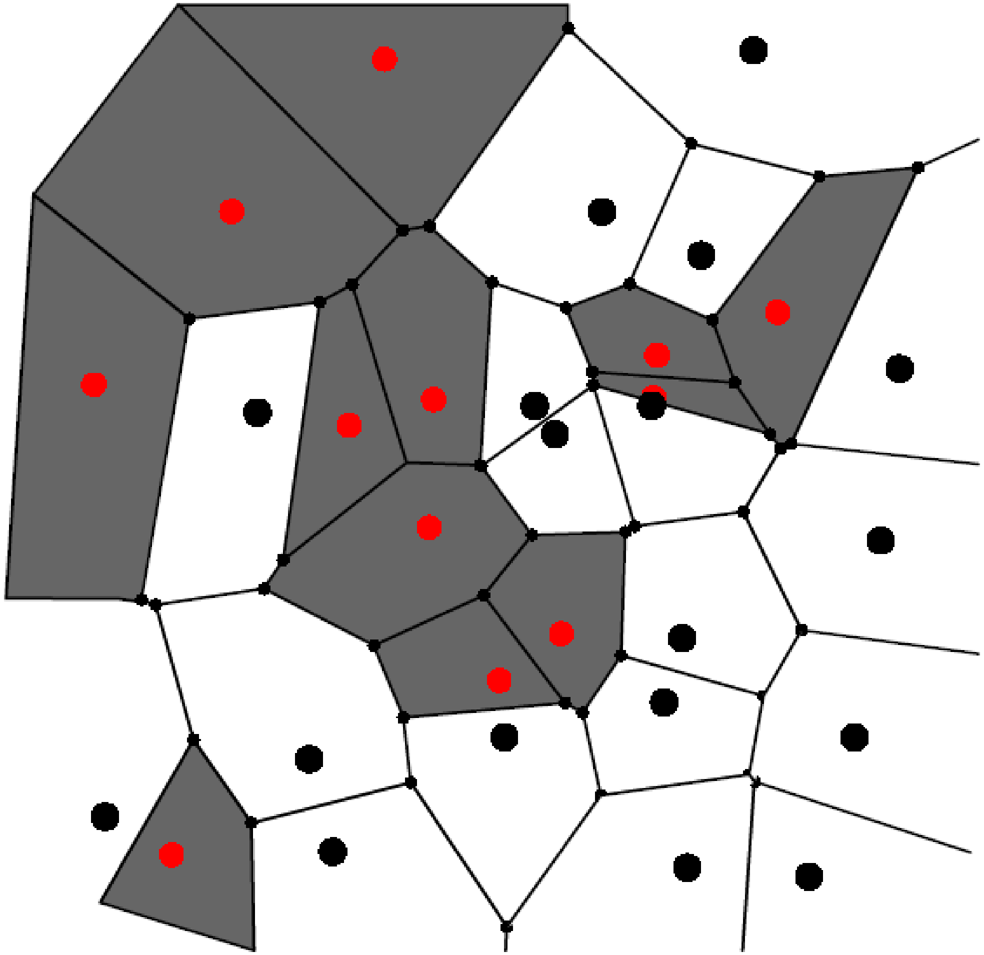


Figure 2.2:
The same classification example in two dimensions as in Figure 2.1.
GREEN = 0, RED = 1
The predicted class is hence chosen by majority vote amongst the 15-*nearest neighbors.*

- Still very flexible !
- Some misclassifications in TR data
- Decision boundaries is *not* linear: it is still quite, although less, *irregular*
- Decision boundary adapts to the local densities of the classes

Voronoi diagram



Each cell consisting of all points closer to ***x*** than to any other patterns.

The segments of the Voronoi diagram are all the points in the plane that are equidistant to two patterns.

Implicitly used by K-NN

K-nn for multi-class

- Return the class most common amongst its *k* nearest neighbors

$$h(\boldsymbol{x}) = \arg\max_{v} \sum_{\boldsymbol{x}_i \in N_k(\boldsymbol{x})} \boldsymbol{1}_{v,y_i}$$

$$\boldsymbol{1}_{v,y_i} = \begin{cases} 1 & \mathit{if} \quad v = y_i \\ 0 & \mathit{otherwise} \end{cases}$$

We count the classes in the neighbor (by the $\boldsymbol{1}_{v,y}$ for each v*)* taking the most frequent class ($\arg\max$)

K-nn variants: Weighted distance

- It can be useful to weight the contributions of the neighbors, so that the nearer neighbors contribute more to the average than the more distant ones.

$$h(\boldsymbol{x}) = \arg\max_{v} \sum_{\boldsymbol{x}_i \in N_k(\boldsymbol{x})} \boldsymbol{1}_{v,y_i} \bullet \frac{1}{d(\boldsymbol{x},\boldsymbol{x}_i)^2}$$

If d = 0 for a i , return y_i

$$\boldsymbol{1}_{v,y_i} = \begin{cases} 1 & if \quad v = y_i \\ 0 & otherwise \end{cases}$$

K-nn: An extreme

- Not a global hypothesis for all the instances → no model to be fit
  - We need to memorize the input examples
- Local estimations (by locally constant functions) vs global linear approximation/estimation of the target function (over the instance space)
- Lazy, memory based, instance-based, **distance-based methods**

K-nn versus Linear

Discussion on linear versus k-nn models

Two extremes of the ML panorama:

- Rigid (low variance) versus flexible (high variance):
  - In K-nn with small k few points can change the decision boundary
  - We may pay a price for this flexibility
- Eager versus lazy
- Parametric versus instance-based

- Linear regression uses 3 parameters to describe its fit in the example of Fig. 2.1, w_0,w_1,w_2 (or $n+1$ in general)
  Does K-nn use 1 (the value of k here)?
- More realistically, K-nn uses *l* /k "effective number of parameters" (Hastie-Tibshirani-Friedman 2001)*
  - where *l* is used the number of data (*N in the book)*

LS linear vs K-nn with various k values

Dip. Informatica
University of Pisa

Figure 2.4: *Misclassification curves for the simulation example used in Fig. 2.1, 2.2 and 2.3.*

Training set of size 200

Test set of size 10.000.

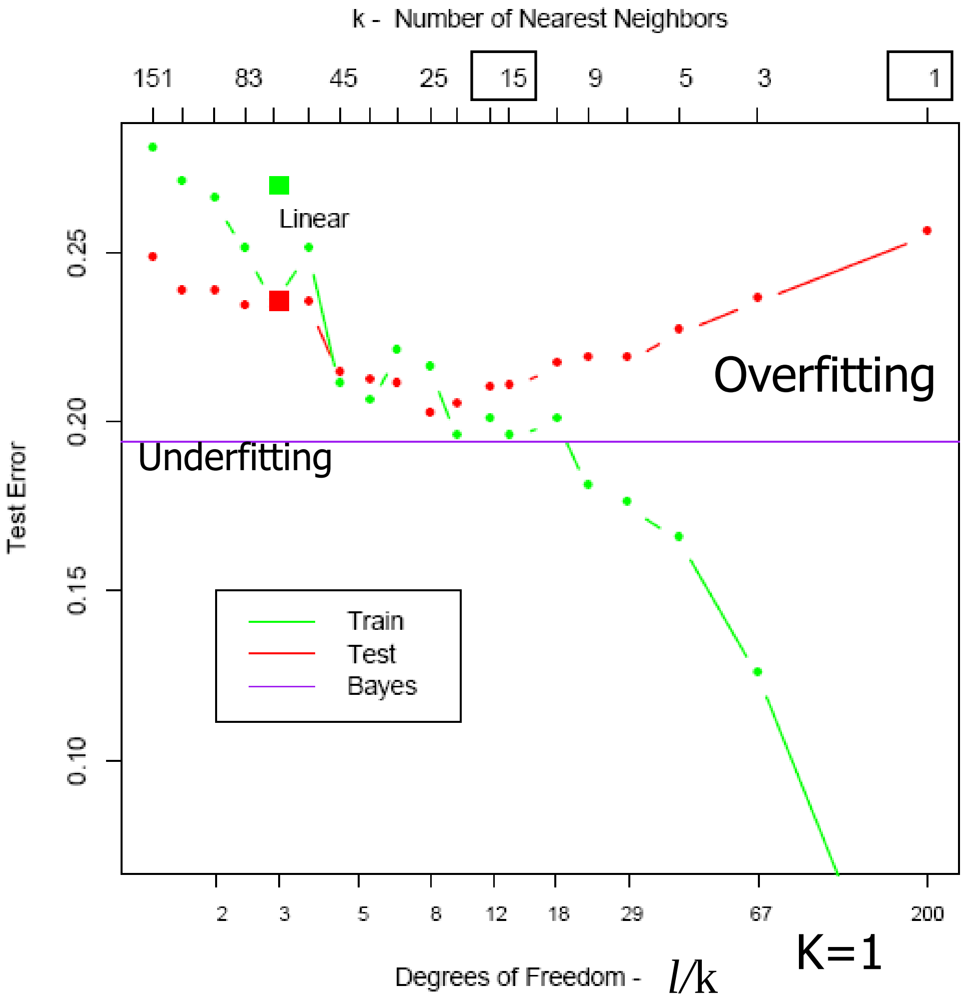
The red curves are test and

the green are training error for k-NN classification (changing K).

The results for linear regression are the bigger green and red dots at three degrees of freedom.

The purple line is the optimal Bayes Error Rate (See next slides)

Note how we move from underfitting to overfitting moving the values of k (i.e. the rate *l* /k): more flexibility allows to find the best result if we control "complexity" by K



Bayes error rate

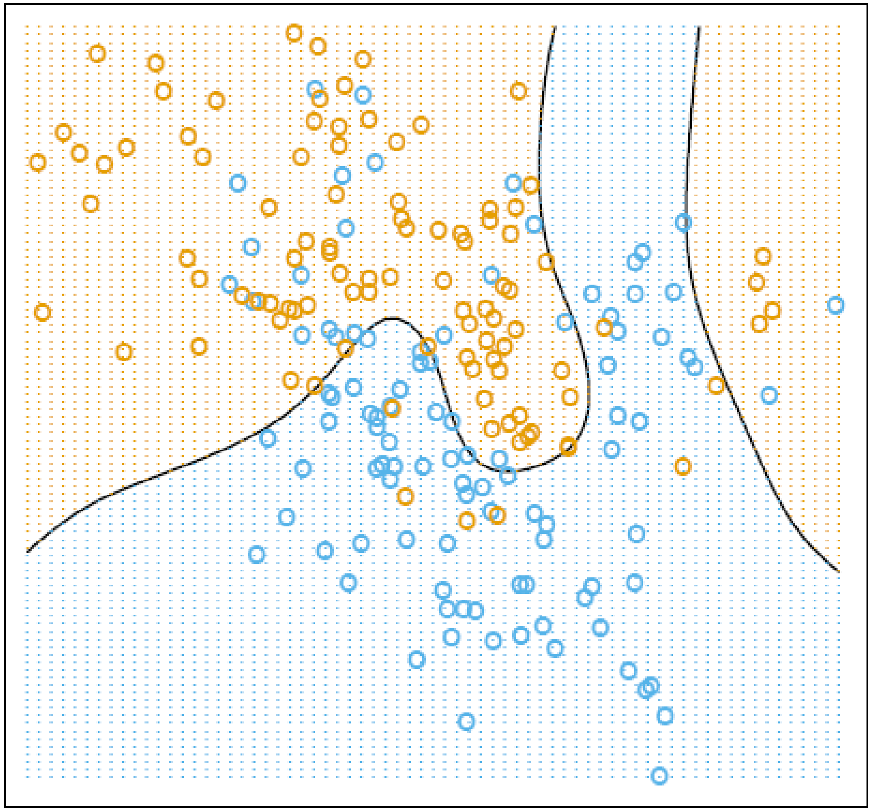
Bayes optimal solution (called Bayes classifier):

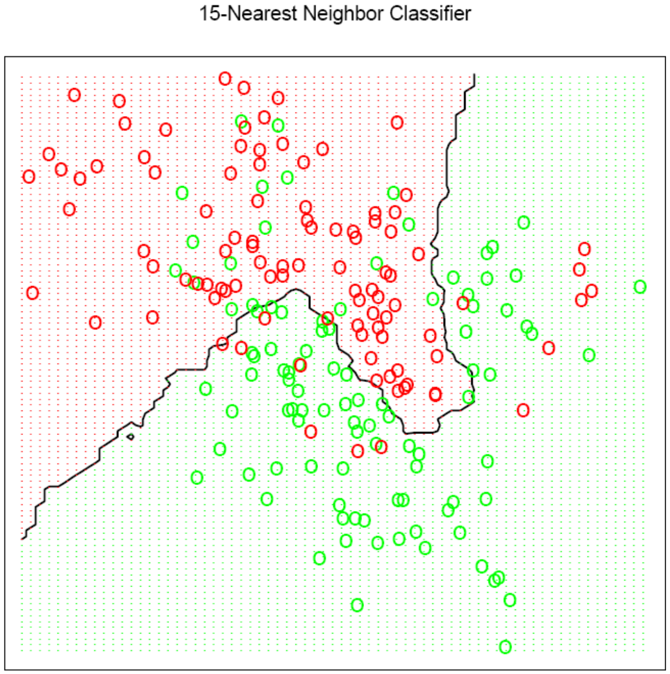
- If we know the density $P(\mathbf{x},y)$ we classify to the most probable class, using the conditional (discrete) distribution as:
  - Output the class v s.t. is max $P(v|\mathbf{x})$ (v *in* $\{C_1,C_2,C_3 \ldots.C_K\}$)

- The error rate of the (optimal) Bayes classifier is called the Bayes rate.
- I.e. the minimum achievable error rate given the distribution of the data (assuming that the generating density is known !!!)

- Note on K-nn: we see that the k-nn classifier directly approximates this solution (a majority vote in a nearest neighborhood amounts to exactly this), except that
  - conditional probability at a point is relaxed to conditional probability within a neighborhood of a point, and
  - probabilities are estimated by training-sample proportions.

Bayes Optimal Classifier: results





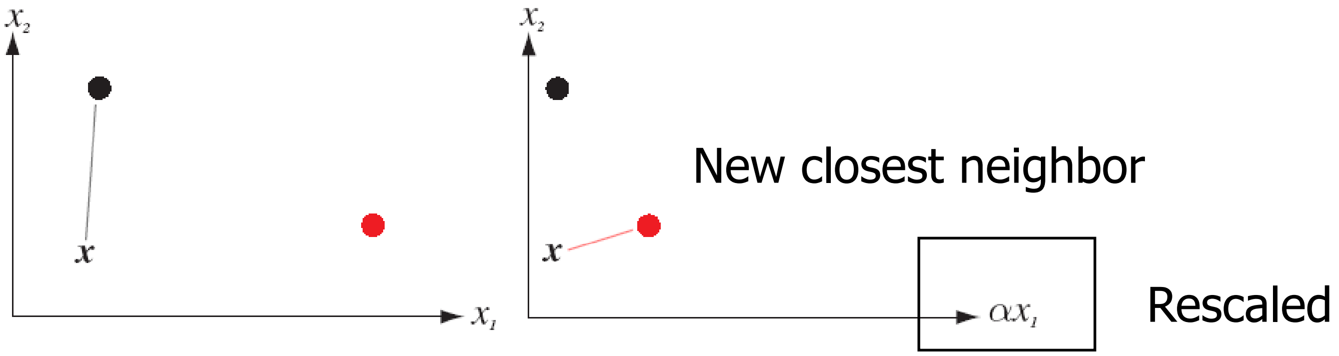
- The optimal Bayes decision boundary for the simulation example.
- Since the generating density is known for each class, this boundary can be calculated exactly
- 15-nn was close to this (indeed it shown a min. test err in the plot a couple of slides ago)

Inductive Bias of k-nn

- The assumed **distance** tells us which are the most similar examples
- The classification is assumed similar to the classification of the neighbors according to the assumed metric
  - E.g. we can use other metric than the Euclidian
  - Symbolic data require “ad-hoc” metrics (e.g. Hamming distance between two strings of equal length is the number of positions for which the corresponding symbols are different).
- Next slides:
  - criticism and limitations

1. Scale changes and 2. other metrics

- **Scale changes**: Domain knowledge dependent choice
- **If** variables should contribute equally:
  - Pay attention to disparity in the ranges of each variables
  - Rescale data to equalize inputs ranges (→ change the metric)!
    E.g. mean zero and variance 1 normalization
- Variable scaling can have a high impact (i.e. k-nn is fragile even with respect to basic preprocessing)



New closest neighbor

Rescaled

(some) Limits of K-nn: computational cost

- Note that K-nn makes the local approximation to the target function for each new example to be predicted:

➢ The computational cost is deferred to the **prediction phase**!

Moreover: high retrieval cost:

- Computationally intensive (in time) for each new input: computing the distances from the test sample *to all* stored vectors
  - The time is proportional to the number of stored patterns
  - "ad-hoc" proximity search algorithms to optimize
  - E.g. by indexing the patterns
- Cost in space (all the training data)

(some) Limits of K-nn

- K-nn models offer little interpretation.
  - Subjectivity of interpretation
  - Dependence on the metric

(some) Limits of K-nn: curse of dim.

K-nn provides a good approximation if we can find a significant set of data close to any ***x***, with dense sampling

When can it fail?

- When we have a lot of input variables (high *n, i.e. high dim.*), K-nn methods often fails because of the ***curse of dimensionality****:*

Many manifestations of this problem: we will examine a few

1. *It is hard to find nearby ("similar") points in high dimensions!*
2. *Low sampling density for high-dim data*
3. *Irrelevant features issue*

These are important in terms of ML,
as they inherently affect the ***generalization capability*** *!!!*

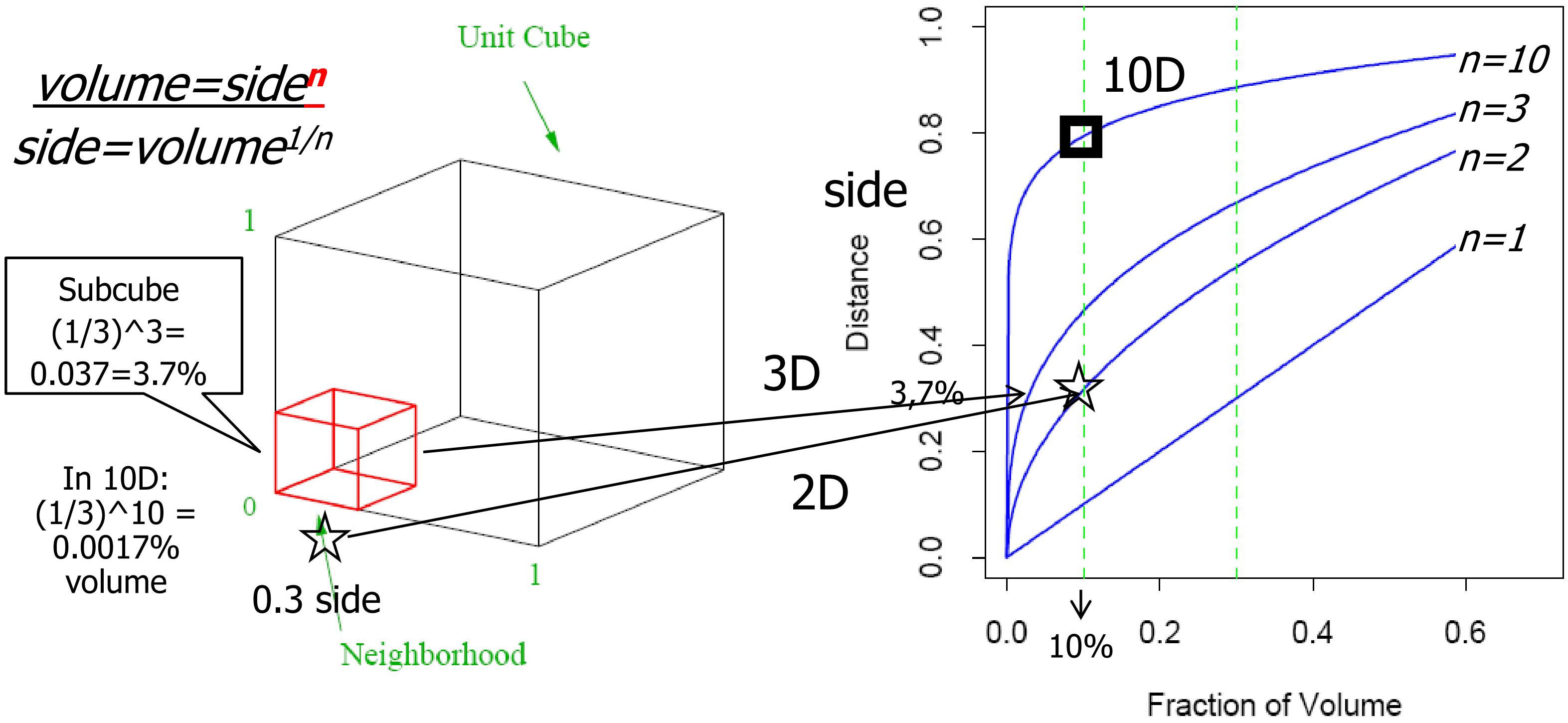
(some) Limits of K-nn: curse of dim.

1. It is hard to find nearby points in high dimensions!

- K-nearest neighbors can fail in high dimensions, because it becomes difficult to gather K observations "close" (in terms of of variables values, i.e. "similar") to a query point x_q:
  - near neighborhoods tend to be **spatially large**, and estimates are not longer local
  - reducing the spatial size of the neighborhood means reducing K, and the variance of the estimate increases (→ overfitting).

- Why? See the next slide…

(some) Limits of K-nn: curse of dim.



volume=side^n

side=volume$^{1/n}$

Subcube
(1/3)^3=
0.037=3.7%

In 10D:
(1/3)^10 =
0.0017%
volume

0.3 side

Figure 2.6: *The curse of dimensionality is well illustrated by a subcubical neighborhood for uniform data in a unit cube.*

The figure on the right shows the ***sidelength*** **(=$r^{1/n}$)** *of the subcube needed to capture a* ***fraction r*** *of the volume of the data, for different dimensions n.*

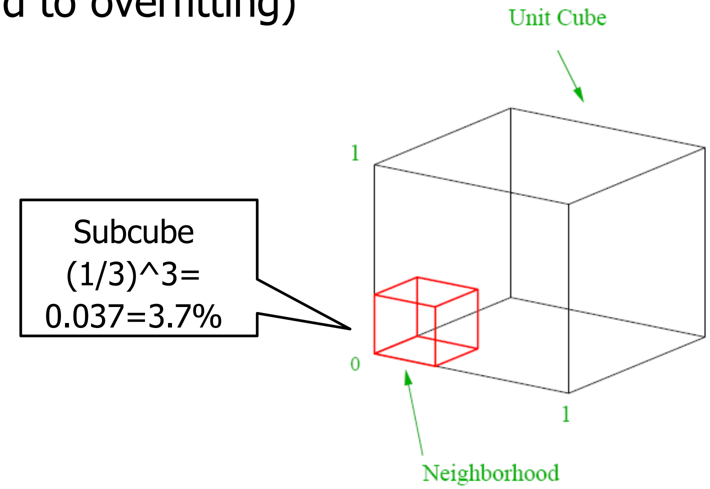
In ten dimensions we need to cover 80% of the range of each coordinate to capture □ *10% of the data. (since 0.1^1/10=~ 0.8), while in 2D 30% (0.316 sidelenght) was suff.* ☆

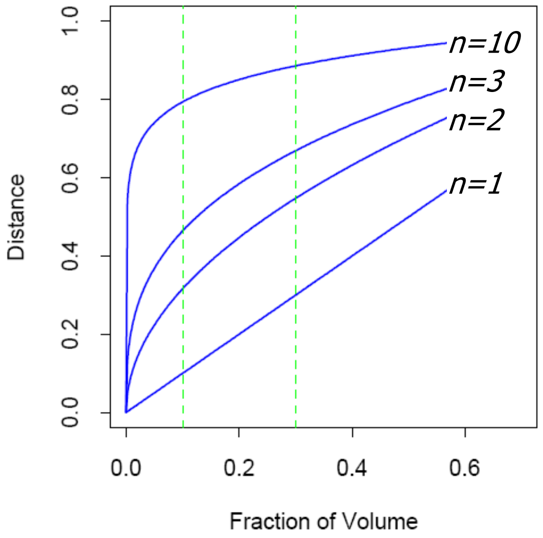
Further explanation

Loosing of generalization capability

1. Image that to have K data you need 10% of the volume. How much ***sidelength*** (features values range) do you need?
   - From 30% to 80% moving from 2D to 10D (**loosing similarity**!!!)
2. On the other side:
   - 1D: with 0.3 ***sidelength*** we take 30% of data volume
   - 2D: with 0.3 ***sidelength*** we take 10% of data volume (the red square)
   - 3D: with 0.3 ***sidelength*** we take 3.7% of data volume (the red cube)
   - 10D: with 0.3 ***sidelength*** we take 0.0017% of data volume.

   This sidelength can be not sufficient to have K data, unless we use **small K** (which can lead to overfitting)


Subcube (1/3)^3= 0.037=3.7%



(some) Limits of K-nn: curse of dim.

2. Low sampling density for high-dim data

- Sampling density is proportional to $l^{1/n}$ (l data, n=data dim).
  - if 100 points are sufficient to estimate a function in IR^1 (1-dim input),
  - 100^{10} are needed to achieve similar accuracy in IR^{10} (10-dim input)

(some) Limits of K-nn: curse of dim.

3. Irrelevant features: The Curse of Noisy

if the target depends on only few of many features in ***x*** (e.g. 2 out 20), we could retrieve a “similar pattern” with the similarity dominated by the large number of irrelevant features

It grows with the dimensionality

An improvement

Irrelevant features:

- We may weight features according to their relevance
  - Stretching the axes along some dimension
  - Weights can be searched by a (expensive) model selection approach or other approaches …

- Feature selection approaches: it eliminates some variables → reduce input dimension

Summary:
K-nn design choices

- The metric d to measure the closeness between patterns (e.g. Euclidian, Hamming, Manhattan distance…, weights on input features).
  Often this is the *key* for a successful application!

- K (number of neighbors: control underfitting/overfitting)

Often necessary to select:

- A subset of data (set of prototypes): e.g. by clustering
- A subset of features

Various approaches to deal with these issues.

Extension in ML

- Extensions to other local models in ML
  - Kernel smoothers
  - Local linear regression
  - Prototype methods
  - Case-based reasoning

K-nn in the course: some general lessons

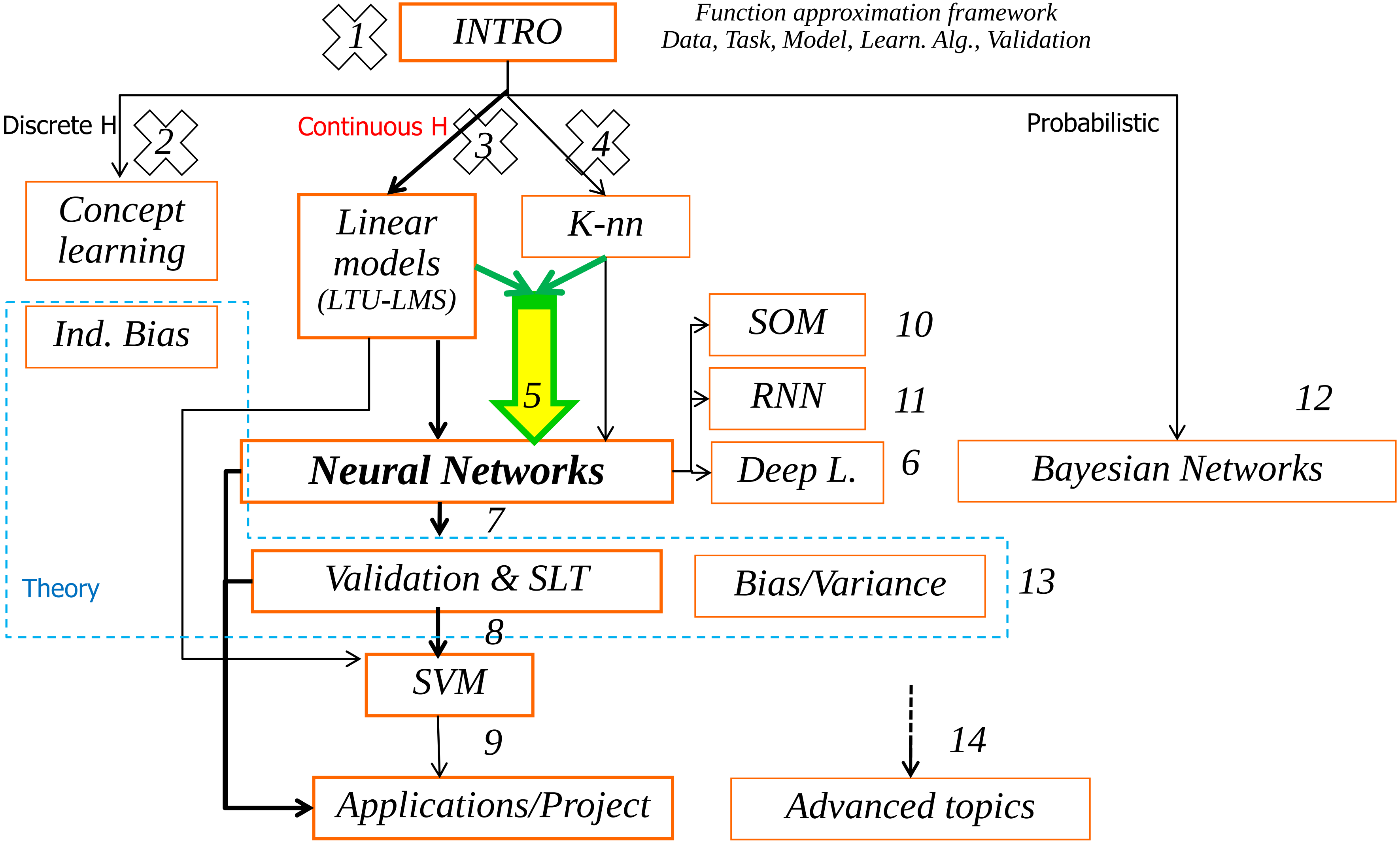
- Too low variance is poor (rigid linear models), too high variance is dangerous (K-nn)

- Other concepts presented todays that are interesting in general:
  - **Smoothing** techniques (in K-nn by increasing K)
  - **Curse of dimensionality** (the volume of the problem space increases so fast that the available data become sparse)
  - Again an **instance of the Statistical Learning Theory** bound plot: see the misclassification plot moving K
  - **Inductive Bias** for K-nn (metric based issue, which is underestimated in many books)

ML Course structure
And now?

- K-nn does not build a "learned model", not regularity/knowledge extraction/synthesis
  - It does not match the "learning" objective (that can forget the examples after the model is built)
- In the next lectures, we will look at model
  - Compact as the LTU model (all the knowledge in few parameters)
  - More flexible than the linear model (flexible as the K-nn)
    - with a suitable support to the control of the complexity

ML Course structure
Where we go



Bibliography (last 3 lectures)

- Hastie, Tibshirani, Friedman, The Elements of Statistical Learning, Springer Verlag, 2001-2017 (check the new Ed.): **chap 2** (note: exists a on-line pdf version)
- Mitchell:
  - Linear model and LMS alg.: chap 4.4
  - K-nn: **chap 8**
- Haykin (2nd edition):
  - Linear model and LMS alg.: chap 3 up to 3.7

    [details on Newton and Gauss-Newton methods are not strictly needed]
- Haykin (3rd edition):
  - Linear model and LMS alg.: chap 3

    [details on Newton and Gauss-Newton and other advanced approaches are not strictly needed]

Bibliography (last 3 lectures)

- Moroever…
- Further readings (not mandatory!):

  – Gently introduction to linear models:

    AIMA (Russel, Norvig, Artificial Intelligence: A Modern Approach), ed .3: **chap 18.6** (18.6.1,18.6.2,18.6.3)

  – To go in deep for **Linear least squares**

  You can start from www resource as (With nice examples):

  https://en.wikipedia.org/wiki/Linear_least_squares_%28mathematics%29

  – **LS in general**

  http://en.wikipedia.org/wiki/Least_squares

For information

Alessio Micheli

micheli@di.unipi.it

www.di.unipi.it/groups/ciml

Dipartimento di Informatica
Università di Pisa - Italy

**Computational Intelligence &
Machine Learning Group**