DISPEC

Scientific exploitation of space Data
for improved Ionospheric SPECification

DISPEC Training School

L3.1 Machine Learning: Basic Theory

Konstantinos Koutroumbas

21-25 September 2026
CIHEAM-MAICh, Chania, Crete, Greece

Email:
koutroum@noa.gr

Contents

Part I: Introductory concepts

- **Model definition**
- Definition of ML via simplified examples
- Significant issues: *noise/uncertainty* existence, *number of data points*, *data finiteness*
- **How** to **approximate** a **true model** and relevant issues

Part II: Basic problems in ML

- Regression
- Supervised classification
- Unsupervised classification (clustering)

Part III: Some ML models in more detail

- k-nearest neighbor (*k-NN*) classifier
- *Decision trees* and *Random Forests*
- Feedforward Neural Networks (*FNNs*)
- Recurrent Neural Networks (*RNNs*) – Long-Short Term Memory (*LSTM*) networks
- Some basic concepts on *time series*

Contents

Part I: Introductory concepts

- **Model definition**
- Definition of ML via simplified examples
- Significant issues: *noise/uncertainty* existence, *number of data points*, *data finiteness*
- **How** to **approximate** a **true model** and relevant issues

Part II: Basic problems in ML

- Regression
- Supervised classification
- Unsupervised classification (clustering)

Part III: Some ML models in more detail

- k-nearest neighbor (*k-NN*) classifier
- *Decision trees* and *Random Forests*
- Feedforward Neural Networks (*FNNs*)
- Recurrent Neural Networks (*RNNs*) – Long-Short Term Memory (*LSTM*) networks
- Some basic concepts on *time series*

Model definition

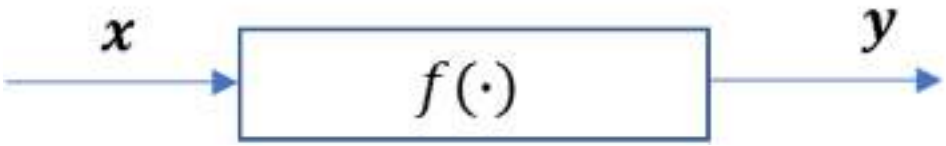
- Several **physical phenomena** evolve in our world
- They are produced by some **mechanisms**
- The appropriate **triggering** of such mechanisms produce phenomena

Lighting mechanism



Quantifying things

Triggered/input variables (e.g., cloud density) → *Mechanism /model* → *Resulting/output variables (e.g., light intensity)*

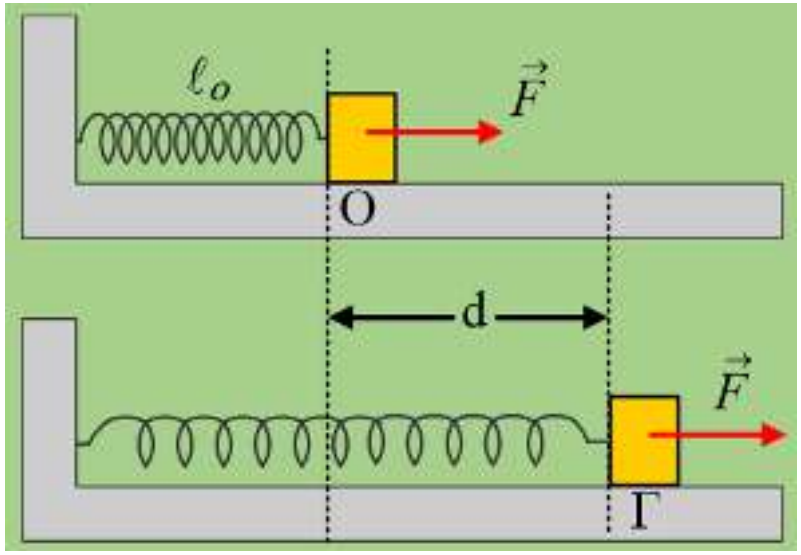


$$y = f(x)$$

Model: A **mathematical description** of a **mechanism** (associated with a **phenomenon**)

A simplified model paradigm (spring expansion)

A Phenomenon: When a **force** is **applied** on the free end of a **spring** (whose other end is anchored) the spring **expands**.



Its Quantification:

- What triggers the phenomenon? The force ⇨ **Quantification** by: *force magnitude* (F)
- What is the effect ? The spring expansion ⇨ **Quantification** by: end displacement (d)

Input variable: F
Output variable: d

- What is the **model** that describes the expansion **mechanism** (phenomenon)?

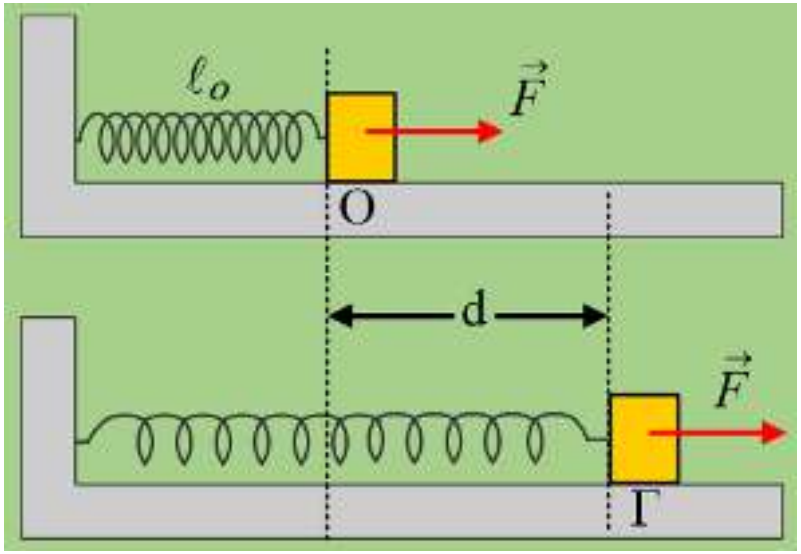
The inverse Hooke law: $d = \frac{1}{k}F$ (k = spring constant parameter)

A simplified model paradigm (spring expansion) (cont.)

So? What is the gain of modelling a phenomenon?

For a given spring, with **known** k, we can **compute** the displacement d for a given force magnitude F, *without the need to perform the relative experiment*.

E.g., for $k = 10\ (\frac{N}{cm})$ and $F = 10N$, it is $d = \frac{1}{k}F = \frac{1}{10}10 = 1cm$



A notational convention:

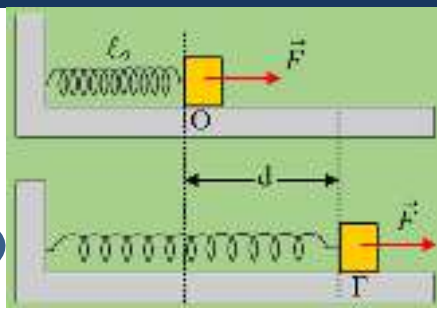
- The **input variables** are denoted by x (in the paradigm, $F \rightarrow x$).
- The **output variables** are denoted by y (in the paradigm, $d \rightarrow y$).

- $\boldsymbol{x}$ and $\boldsymbol{y}$ denote (column) vectors of input and output variables.

A simplified model paradigm (spring expansion) (cont.)

But, how can we proceed when k is unknown?

Clearly, we need to **compute** k before. But how?



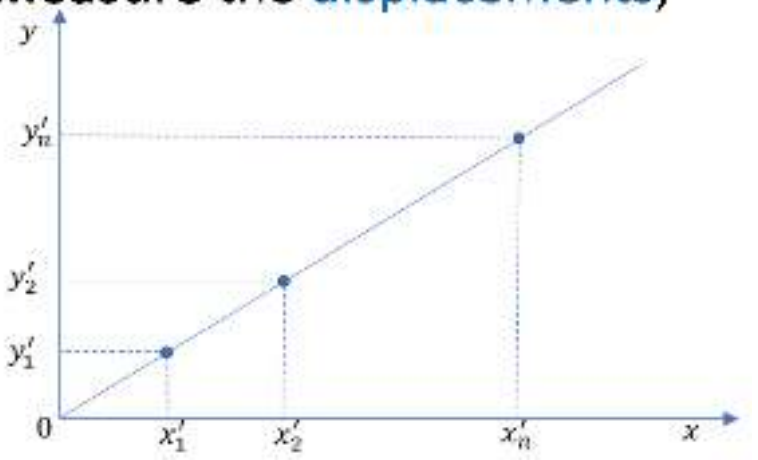
Based on data.

The **model** form is **known**. Its **parameters** are **unknown**.

Which data?

Data that are **produced** through **experimentation:**

apply forces of different magnitudes, $x'_1, x'_2, \dots, x'_n$, and **measure** the displacements, $y'_1, y'_2, \dots, y'_n$.



Data set: $S' = \{(y'_i, x'_i), i = 1, \dots, n\}$

Computation of k:

Due to inverse Hooke's law $(d = \frac{1}{k}F)$ $y = \frac{1}{k}x$, it is

$$\frac{1}{k} = \frac{y'_j}{x'_j} \Leftrightarrow k = \frac{x'_j}{y'_j}$$

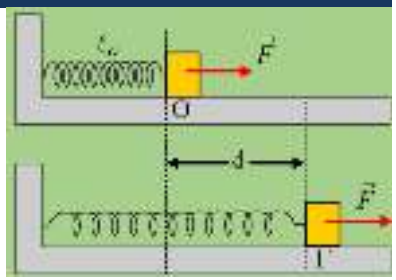
Since k is now **known,** we can **compute** the displacement y for a given force magnitude x, *without the need to perform the relative experiment*.

A simplified model paradigm (spring expansion) (cont.)

The computation of k before is valid under one assumption:

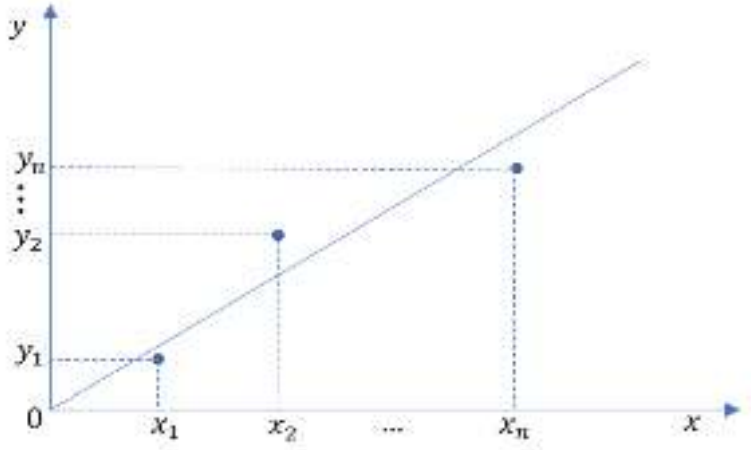
The ***measurements*** are ***"perfect".***

But always in practice **errors/uncertainties** are entered on the **measurements** (disturbed environmental conditions, measuring device accuracy,…)



Modelling of errors: $y_i = y_i' + e_i, i = 1, \dots, n$ (e_i is of **random** nature)

Data set: $S = \{(y_i, x_i), i = 1, \dots, n\}$



How can we **compute** k now based on data?

We **cannot**!!

We can only **estimate** k based on S. How?

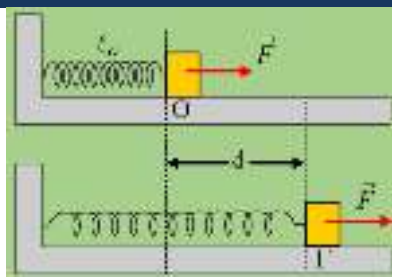
Due to the Hooke's inverse law $y = \frac{1}{k}x$ we will

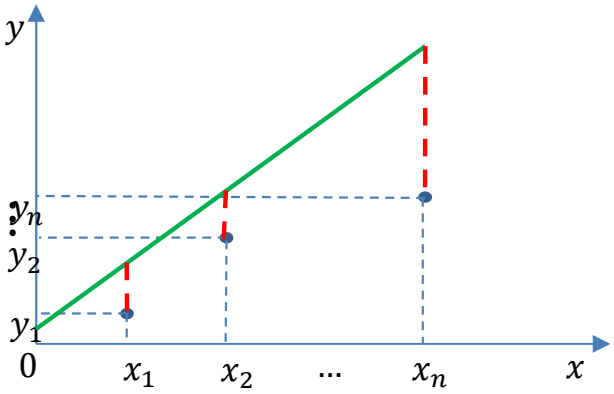
- **determine** the **line** $y = \lambda \cdot x$ that "**best fits**" the data points
- **estimate** k as $\hat{k} = \frac{1}{\lambda}$

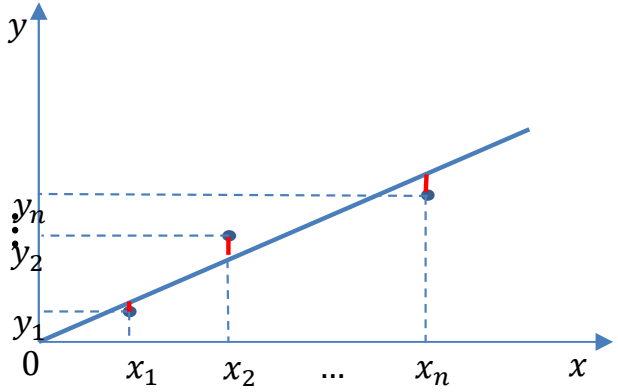
A simplified model paradigm (spring expansion) (cont.)

But what "best fit" means?

To give a meaning, a **criterion function** need to be **adopted**.







The line $y = \lambda \cdot x$ that **best fits** the data is the one that (ideally) **minimizes** all the **absolute differences** $|y_i - \lambda \cdot x_i|$, for all data points.

A **criterion** that complies with the above is the **sum of error squares** (SSE)

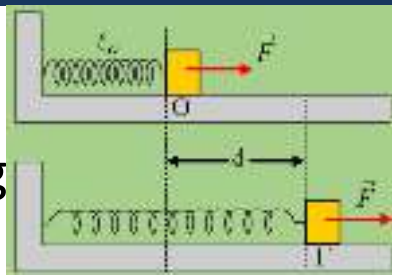
$$J(\lambda) = \sum_{i=1}^{n} (y_i - \lambda \cdot x_i)^2$$

Minimizing $J(\lambda)$ wrt λ $\frac{dJ(\lambda)}{d\lambda} = 0 \Leftrightarrow \lambda = \frac{\sum_{i=1}^{n} x_i y_i}{\sum_{i=1}^{n} x_i^2} \Rightarrow \hat{k} = \frac{1}{\lambda} = \frac{\sum_{i=1}^{n} x_i^2}{\sum_{i=1}^{n} x_i y_i}$

Since k has been **estimated** $(\hat{k})$**,** we can **estimate** the displacement y for a given force magnitude x, *without the need to perform the relative experiment*.

A simplified model paradigm (spring expansion) (cont.)

So what we did up to now:



- We **modeled** the **phenomenon** under study (spring expansion), throug the Hooke's inverse law.

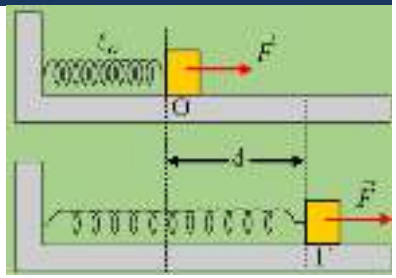
- We **used** a **noisy data** set (S) to **estimate** the **parameters** of the model (k), utilizing a suitable **criterion function.**

- We **use** the **estimate** of k ($\hat{k}$) to **estimate** the displacement y for a given force magnitude x, *without the need to perform the relative experiment*.

- *This is what is called "learning from the data":*
  - *Parameter estimation (k)*
  - *Output estimation (y) of the model for given input (x).*

The above approach is called "**parametric**".

A simplified model paradigm (spring expansion) (cont.)



In the "**nonparametric**" approach only the issue of the

output estimation (y) for given input (x)

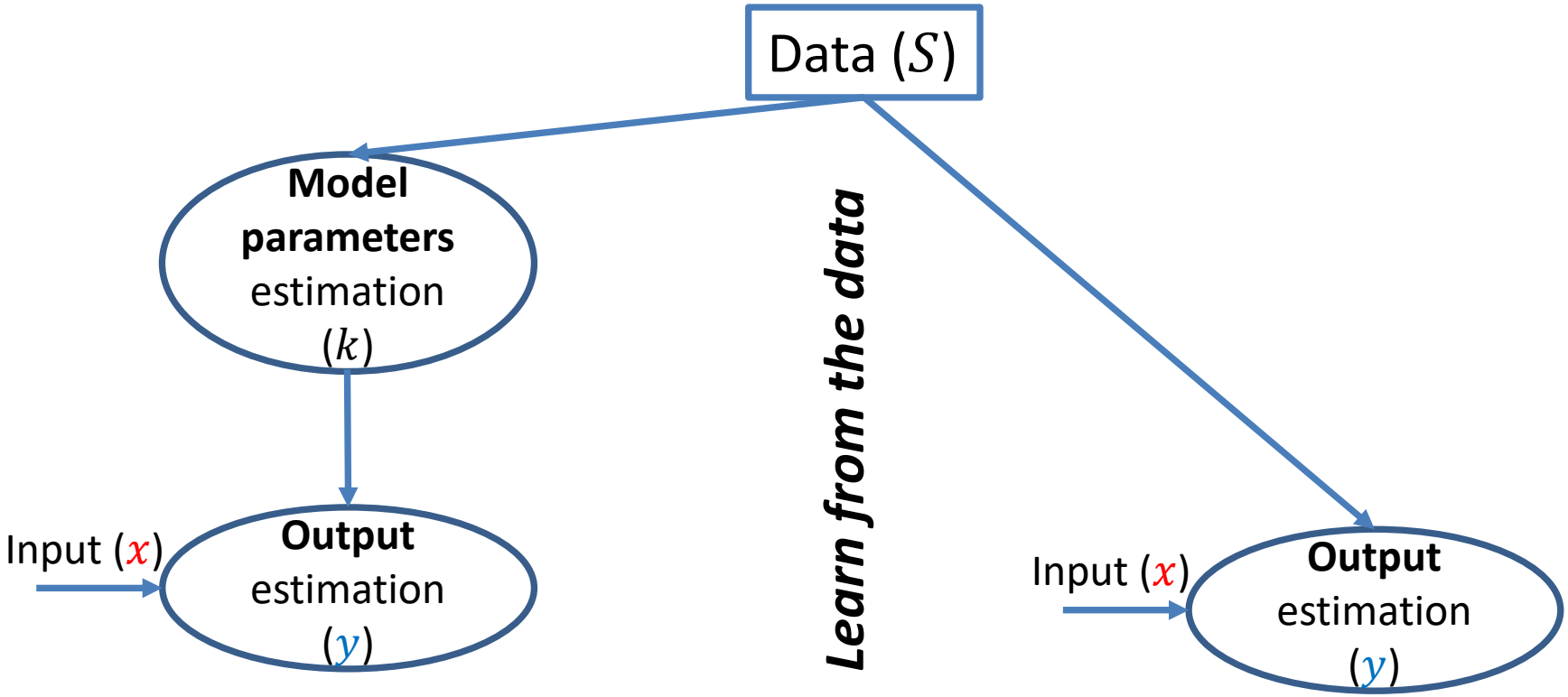
is considered and is **based directly** on the data set

$$S = \{(y_i, x_i), i = 1, \dots, n\}$$

A simple example (for the ***spring expansion*** problem)**:**

- For a given x, **determine** its k-closest neighbors in S, say $x_1, \dots, x_k$
- **Consider** their corresponding y values, that is $y_1, \dots, y_k$
- **Estimate** the y value associated with x as the **mean** of $y_1, \dots, y_k$.

“Learn from the data”



Parametric modeling

Nonparametric modeling

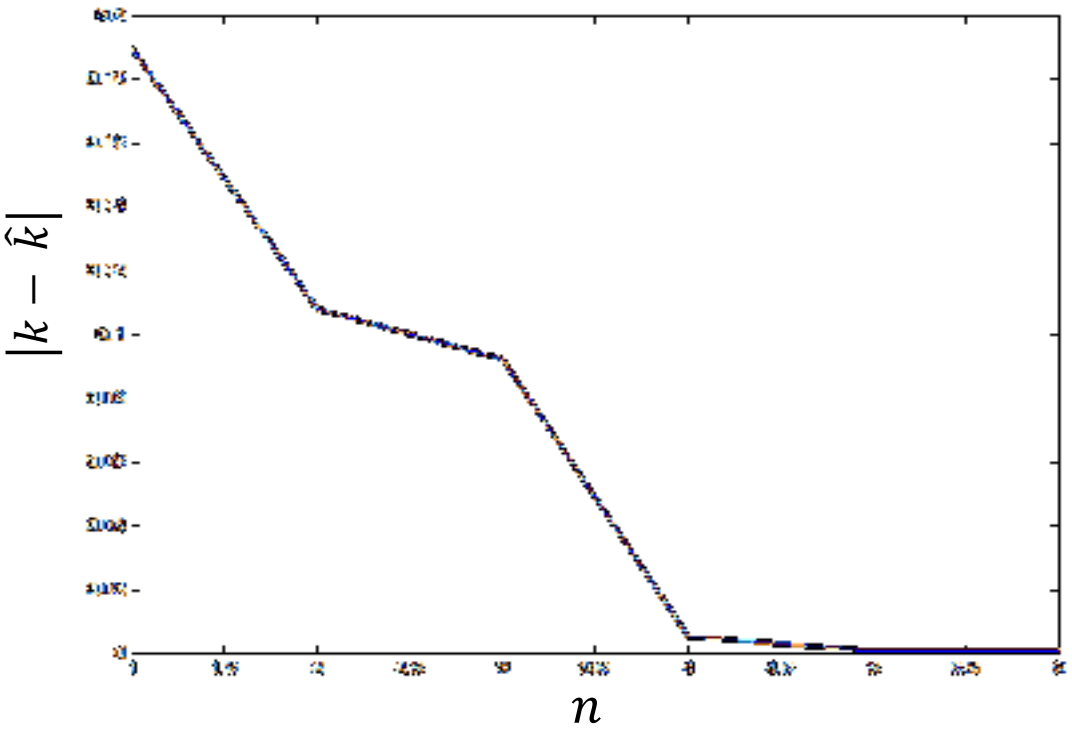
Number of data

Question: *What is the **appropriate size** n for the data set S that gives **more reliable model parameter estimations**?*

Answer: The **higher** the n the **more reliable** the model parameter **estimates** are.

Example:

- **Consider** a **spring** with $k = 10$.
- **Generate** data sets (S_n) of sizes $n = 10, 100, 1000, 10000, 100000, 1000000$
- **Use** each S_n to **estimate** k $(\hat{k})$
- **Compute** the **difference** $|k - \hat{k}|$



Number of data

A significant clarification:

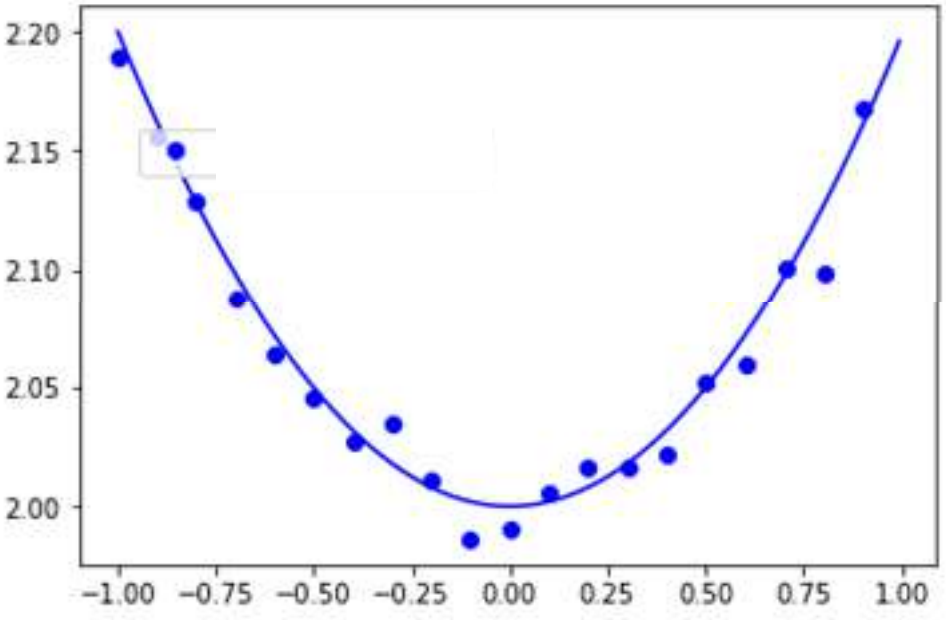
⇨The **data set** should contain **samples** from the **whole** "**range of activity**" of the phenomenon under study.

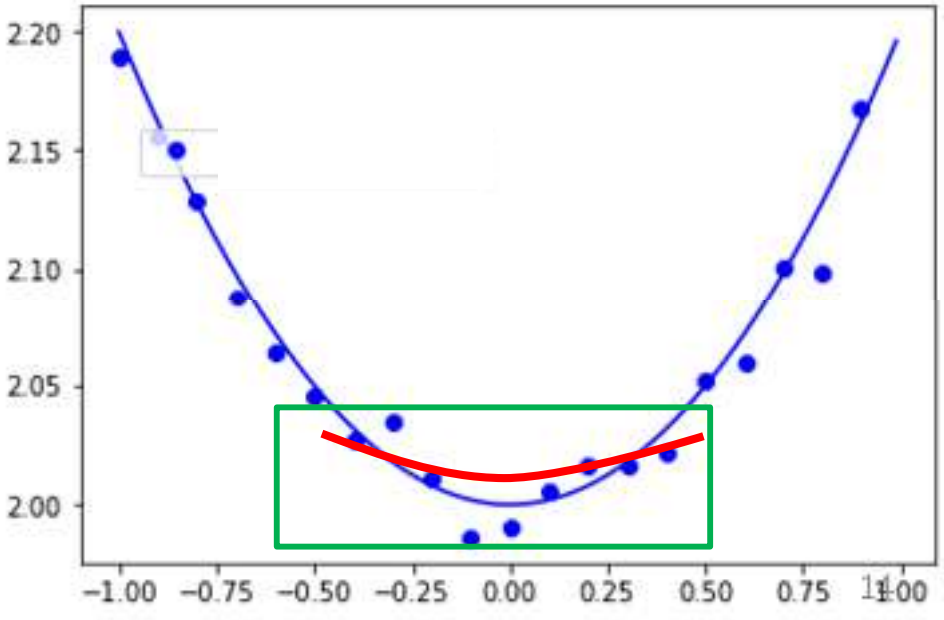
Example: Consider a **quadratic model**.
Left case: data from the **whole "range of activity"** are available

⇨ **Reliable** parameter estimations can be performed.

Right case: data from a **subset of the whole "range of activity"** are available

⇨ **Less Reliable** parameter estimations can be performed





The case where a more loose info for the model is available

In the **spring** example the associated **model** was **completely known** ($y = \frac{1}{k}x$).

What if we had a more **loose information** about the **true model**?

Paradigm:

- A **body** moves on a **straight line** (always in the same direction) and it **actually** performs a **smoothly accelerating** motion ($s = s_0 + v_0 t + \frac{1}{2}\gamma t^2 \longrightarrow y = \theta_0 + \theta_1 x + \theta_2 x^2$).
- However, we **only know** that $s = f(t)$ is a 6th degree polynomial at the most ($s = \theta_0 + \theta_1 t + \theta_2 t^2 + \theta_3 t^3 + \theta_4 t^4 + \theta_5 t^5 + \theta_6 t^6 \longrightarrow$

$$y = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4 + \theta_5 x^5 + \theta_6 x^6 = [\theta_0, \theta_1, \theta_2, \theta_3, \theta_4, \theta_5, \theta_6] \cdot \begin{bmatrix} 1 \\ x \\ x^2 \\ x^3 \\ x^4 \\ x^5 \\ x^6 \end{bmatrix} = \boldsymbol{\theta}^T \boldsymbol{x}).$$

How do we proceed with the model parameter estimation?

We proceed as previously. We **utilize**

- a **data set** $S = \{(y_i, \boldsymbol{x}_i), i = 1, \dots, n\}$
- a **criterion function**, e.g., $J(\boldsymbol{\theta}) = \sum_{i=1}^{n} (y_i - \boldsymbol{\theta}^T \boldsymbol{x}_i)^2$

Minimizing $J(\boldsymbol{\theta})$ wrt $\boldsymbol{\theta}$, we take

Least Squares (**LS**) estimation:

$$\widehat{\boldsymbol{\theta}} = (X^T X)^{-1}(X^T \boldsymbol{y})$$

$$X = \begin{bmatrix} \boldsymbol{x}_1^T \\ \boldsymbol{x}_2^T \\ \vdots \\ \boldsymbol{x}_n^T \end{bmatrix} = \begin{bmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^6 \\ 1 & x_2 & x_2^2 & \cdots & x_2^6 \\ \vdots & & & \vdots & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^6 \end{bmatrix}, \boldsymbol{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

The case where a more loose info for the model is available…

Paradigm:

- A **body** moves on a **straight line** (always in the same direction) and it **actually** performs a **smoothly accelerating** motion ($s = s_0 + v_0 t + \frac{1}{2}\gamma t^2 \longrightarrow y = \theta_0 + \theta_1 x + \theta_2 x^2$).
- However, we **only know** that $s = f(t)$ is a 6th degree polynomial at the most ($s = \theta_0 + \theta_1 t + \theta_2 t^2 + \theta_3 t^3 + \theta_4 t^4 + \theta_5 t^5 + \theta_6 t^6 \longrightarrow$ $y = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4 + \theta_5 x^5 + \theta_6 x^6 = \boldsymbol{\theta}^T \boldsymbol{x}$).

Ideally, we expect that $\hat{\theta}_3, \ldots, \hat{\theta}_6 \approx 0$. This will actually **reveal** also the **true model order** (since only $\hat{\theta}_0, \hat{\theta}_1, \hat{\theta}_2$ are expected to be non zero).

But, is this the case?

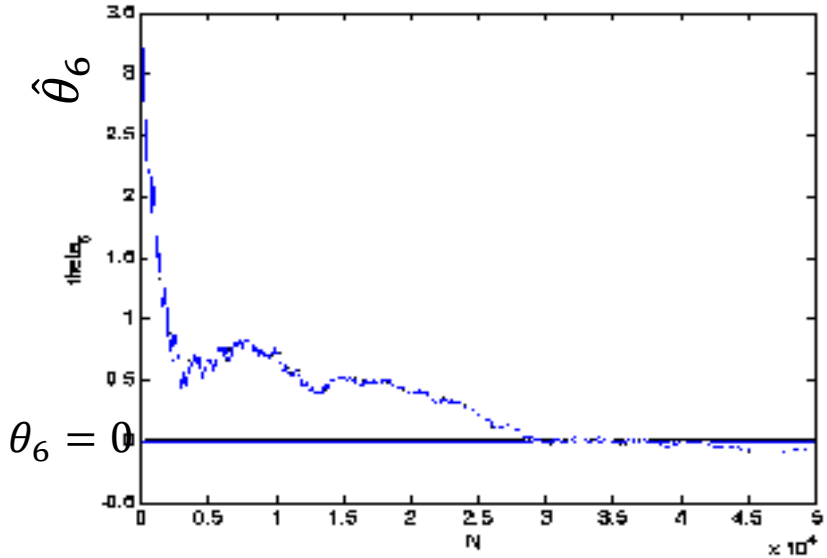
The case where a more loose info for the model is available…

Example: Assume that

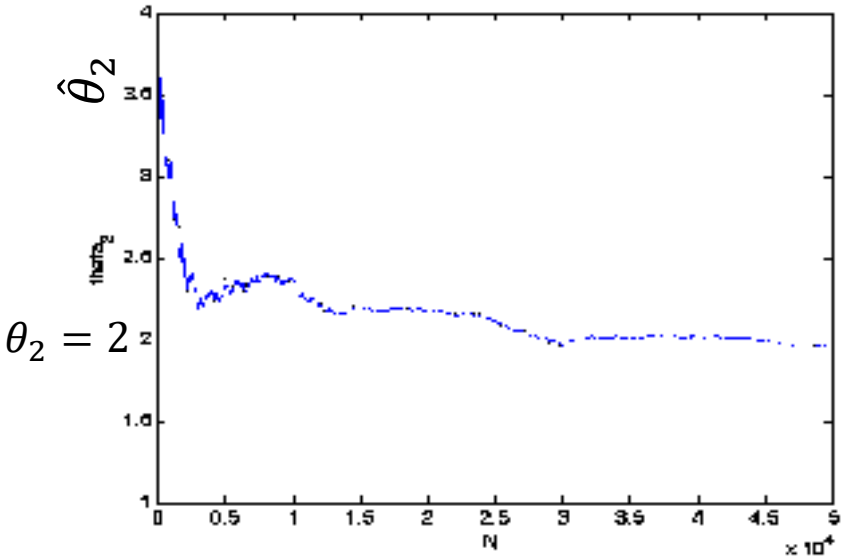
- the **true model** is $y = 1 + x + 2x^2$
- we consider the model $y = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4 + \theta_5 x^5 + \theta_6 x^6$

Applying the previous rationale for data sets of different sizes, we conclude again that:
*The **larger** (and more "representative") the **data set** at hand the **more reliable** the **parameter estimates** are.*

In addition:
*For large enough data sets, the **true model order** is also **revealed**.*



Data set size (n)



Data set size (n)

More loose info for the model & data finiteness

Example: Assume that

- the true model is $y = 1 + x + 2x^2$
- we consider the model $y = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4 + \theta_5 x^5 + \theta_6 x^6$

However, things are not so bright as they seem.

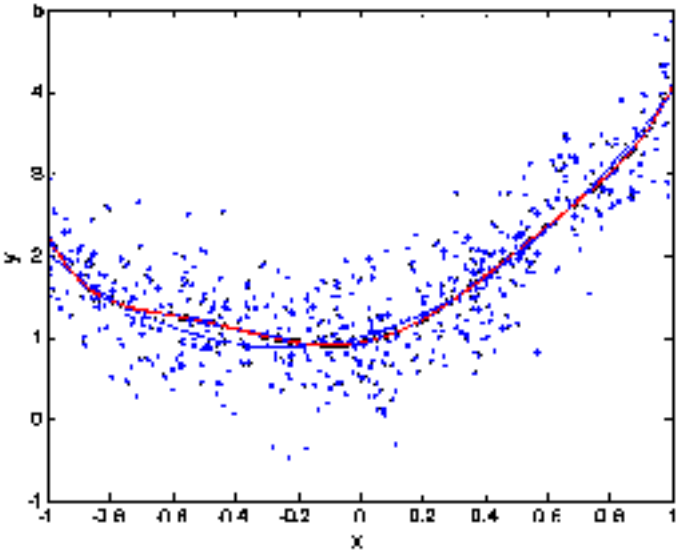
A **common problem** in practice: **Data finiteness**

E.g., for $n = 500$, it is

	θ_0	θ_1	θ_2	θ_3	θ_4	θ_5	θ_6
True value	1	1	2	0	0	0	0
LS est.	0.94	0.68	3.72	0.87	−4.74	−0.63	3.25

Thus,

- The **estimates** are **far** from the **true values** and
- the **model order** is **not revealed**



- **Observation:** The model has greater "representational power" (6th degree polynomial) than is needed (2nd degree polynomial. Thus, *it has the ability to adapt on the peculiarities of the specific **finite** data set.*
- **A way out: Restrict** its representational power.
- **How?** By **imposing restrictions** on the size of the norm of the model parameter vector $\boldsymbol{\theta} = [\theta_0, \theta_1, \theta_2, \theta_3, \theta_4, \theta_5, \theta_6]^T$

More loose info for the model & data finiteness

Example: Assume that

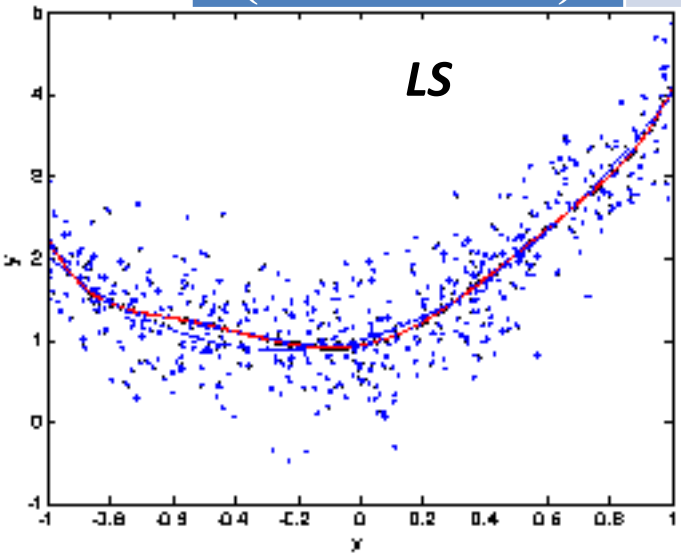
- the true model is $y = 1 + x + 2x^2$
- we consider the model $y = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4 + \theta_5 x^5 + \theta_6 x^6$

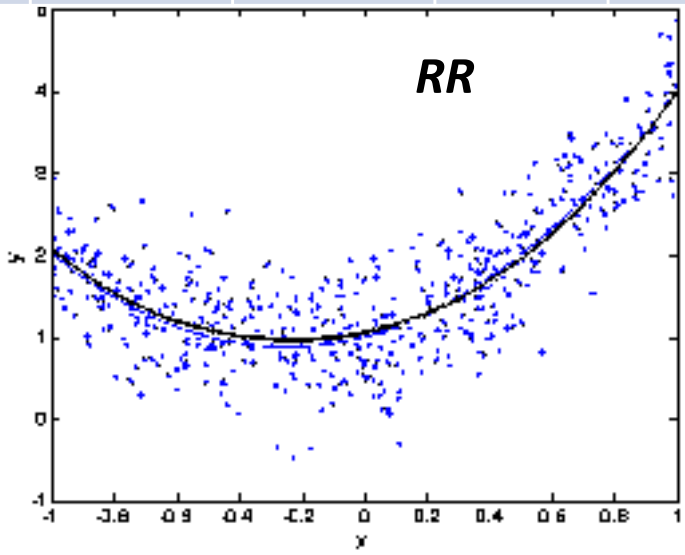
A famous paradigm: Ridge regression (***RR***)

The **criterion function** is now $J(\boldsymbol{\theta}) = \sum_{n=1}^{N}(y_n - \theta_0 - \boldsymbol{\theta}^T \boldsymbol{x}_n)^2 + \lambda||\boldsymbol{\theta}||^2$

Solution: $\widehat{\boldsymbol{\theta}} = (X^T X + \lambda I)^{-1}(X^T \boldsymbol{y})$ (see notes for a **more accurate** description)

	θ_0	θ_1	θ_2	θ_3	θ_4	θ_5	θ_6
True value	1	1	2	0	0	0	0
LS est. (norm= 7.02)	0.94	0.68	3.72	0.87	−4.74	−0.63	3.25
RR est. ($\lambda = 1$) (norm= 2.27)	1.06	0.79	1.81	0.33	0.14	−0.18	0.03





Parametric models in more detail

Parametric model: A parametric set of functions $g(\cdot;\boldsymbol{\theta}): \boldsymbol{x} \in R^l \rightarrow y \in R$

$$\mathcal{M} = \{g(\cdot;\boldsymbol{\theta}), \boldsymbol{\theta} \in A \subseteq R^K\}$$

An *instance* of the model results for a specific value of $\boldsymbol{\theta}$.

Example 1: The parametric set of linear functions ($\mathcal{M}_{lin}^1$) $g(\cdot;\boldsymbol{\theta}): x \in R \rightarrow y \in R$

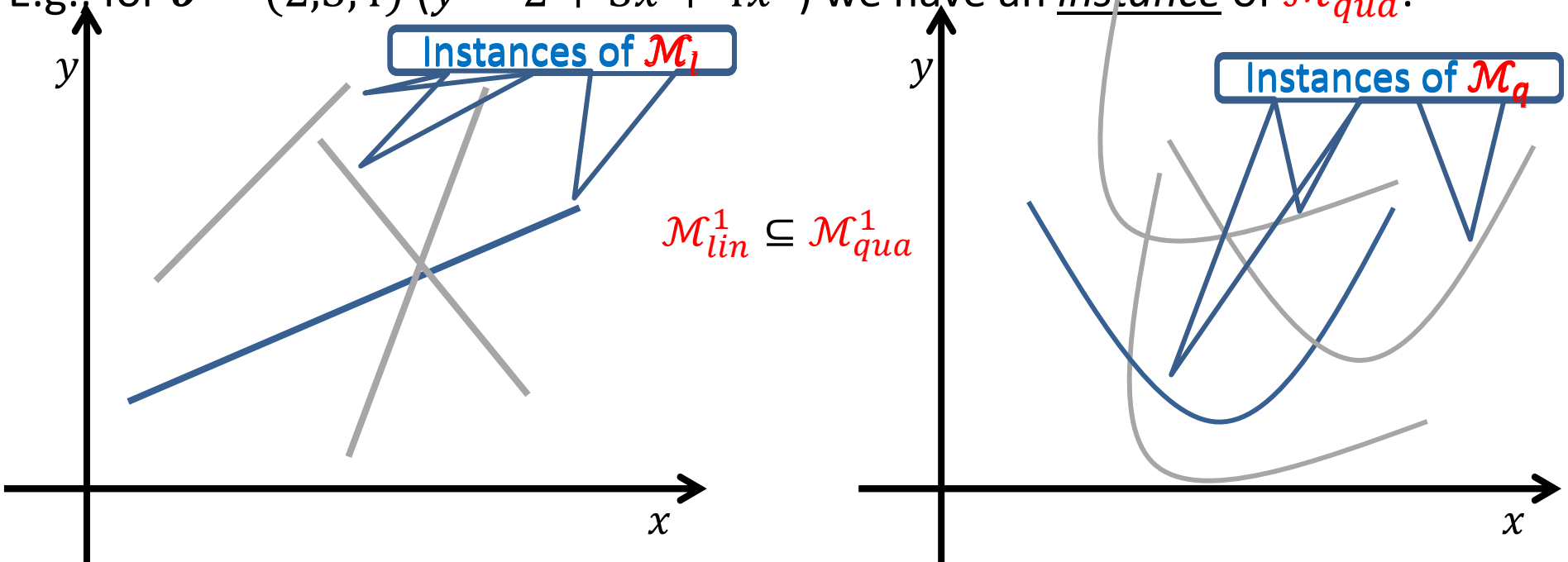
$$y \equiv g(x;\boldsymbol{\theta}) = \theta_0 + \theta_1 x, \boldsymbol{\theta} = (\theta_0, \theta_1)$$

E.g., for $\boldsymbol{\theta}(= (\theta_0, \theta_1)) = (2,3)$ $(y = 2 + 3x)$ we have an *instance* of $\mathcal{M}_{lin}^1$.

Example 2: The parametric set of quadratic functions ($\mathcal{M}_{qua}^1$) $g(\cdot;\boldsymbol{\theta}): x \in R \rightarrow y \in R$

$$y \equiv g(x;\boldsymbol{\theta}) = \theta_0 + \theta_1 x + \theta_2 x^2, \boldsymbol{\theta} = (\theta_0, \theta_1, \theta_2)$$

E.g., for $\boldsymbol{\theta} = (2,3,4)$ $(y = 2 + 3x + 4x^2)$ we have an *instance* of $\mathcal{M}_{qua}^1$.



Parametric models in more detail

Example 3: The parametric set of linear func. ($\mathcal{M}_{lin}^{2}$) $g(\cdot;\boldsymbol{\theta})$: $\boldsymbol{x} = [x_1, x_2]^T \in R^2 \rightarrow y \in R$

$$y \equiv g(x;\boldsymbol{\theta}) = \theta_0 + \theta_1 x_1 + \theta_2 x_2, \boldsymbol{\theta} = (\theta_0, \theta_1, \theta_2)$$

E.g., for $\boldsymbol{\theta}(= (\theta_0, \theta_1, \theta_2)) = (2,3,7)$ $(y = 2 + 3x_1 + 7x_2)$ we have an *instance* of $\mathcal{M}_{lin}^{2}$.

Example 4: The parametric set of linear functions ($\mathcal{M}_{lin}^{l}$)

$$g(\cdot;\boldsymbol{\theta})\text{: } \boldsymbol{x} = [x_1, \dots, x_l]^T \in R^l \rightarrow y \in R$$

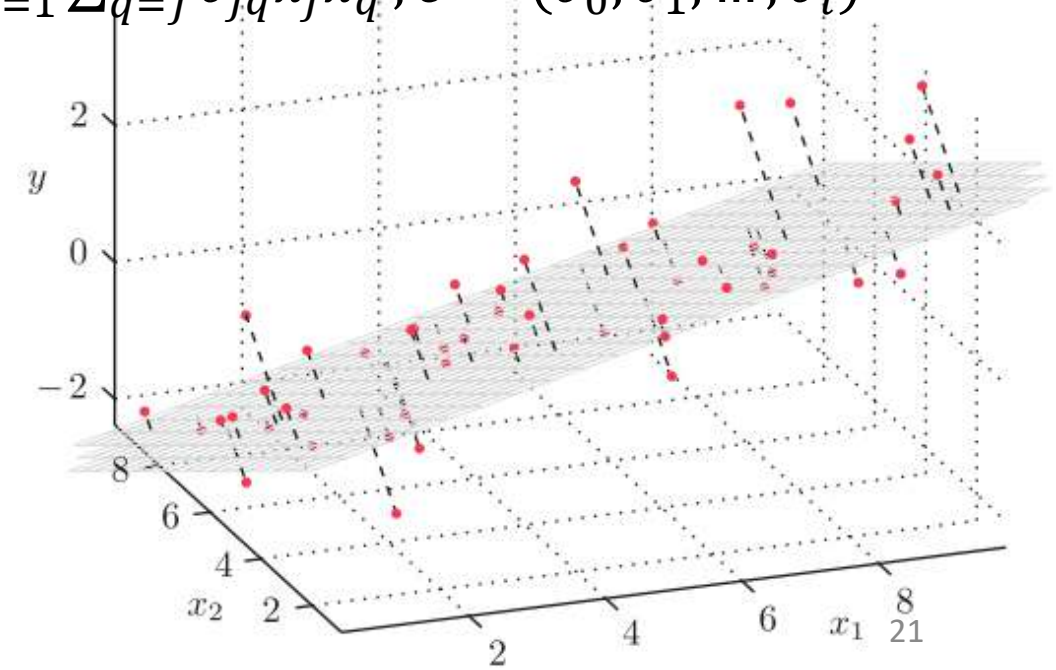
$$y \equiv g(x;\boldsymbol{\theta}) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_l x_l, \boldsymbol{\theta} = (\theta_0, \theta_1, \dots, \theta_l)$$

Example 5: The parametric set of quadratic functions ($\mathcal{M}_{qua}^{l}$)

$$g(\cdot;\boldsymbol{\theta})\text{: } \boldsymbol{x} = [x_1, \dots, x_l]^T \in R^l \rightarrow y \in R$$

$$y \equiv g(x;\boldsymbol{\theta}) = \theta_0 + \sum_{i=1}^{l} \theta_i x_i + \sum_{j=1}^{l} \sum_{q=j}^{l} \theta_{jq} x_j x_q, \boldsymbol{\theta} = (\theta_0, \theta_1, \dots, \theta_l)$$

Again $\mathcal{M}_{lin}^{l} \subseteq \mathcal{M}_{qua}^{l}$



Parametric models in more detail

Example 6: The parametric set of r-th degree polynomial func. ($\mathcal{M}_r^l$)

$$g(\cdot;\boldsymbol{\theta}): \boldsymbol{x} = [x_1, \dots, x_l]^T \in R^l \rightarrow y \in R$$

$$y \equiv g(x;\boldsymbol{\theta}) = \theta_0 + \sum_{\iota_1=1}^{l} \theta_{i_1} x_{i_1} + \sum_{\iota_1=1}^{l} \sum_{\iota_2=\iota_1}^{l} \theta_{i_1 i_2} x_{i_1} x_{i_2} + \cdots +$$
$$\sum_{\iota_1=1}^{l} \sum_{\iota_2=\iota_1}^{l} \cdots \sum_{\iota_r=\iota_{r-1}}^{l} \theta_{i_1 i_2 \dots i_r} x_{i_1} x_{i_2} \dots x_{i_r}, \quad \boldsymbol{\theta} = (\theta_0, \theta_1, \dots, \theta_l, \theta_{11}, \dots \theta_{ll}, \dots)$$

($\mathcal{M}_{lin}^l \rightarrow \mathcal{M}_1^l, \mathcal{M}_{qua}^l \rightarrow \mathcal{M}_2^l$)

It is

$$\mathcal{M}_1^l \subseteq \mathcal{M}_2^l \subseteq \cdots \mathcal{M}_r^l \subseteq \cdots$$

Remark: The **higher** the r, the **higher** the "representational power" of the polynomial.

Polynomial family

$$\mathcal{M}^l = \mathcal{M}_1^l \cup \mathcal{M}_2^l \cup \cdots \mathcal{M}_r^l \cup \cdots$$

Parametric models in more detail

Example 7: The parametric set of 2-layer feedforward Neural Networks (2-layer FNNs) with k hidden nodes ($\mathcal{F}_k^l$)

$$g(\cdot;\boldsymbol{\theta}): \boldsymbol{x} = [x_1, \dots, x_l]^T \in R^l \rightarrow y \in R$$

$$y \equiv g(x;\boldsymbol{\theta}) = \theta_0 + \sum_{j=1}^{k} \theta_j s\left(\theta_{0j} + \sum_{i=1}^{l} \theta_{ij} x_i\right), \ \boldsymbol{\theta} = (\theta_0, \theta_1, \dots, \theta_k, \theta_{01}, \dots \theta_{kl})$$

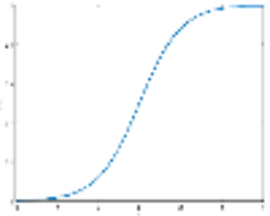
For the 2-layer FNN with $k = 2$ it is

$$y \equiv g(x;\boldsymbol{\theta}) = \theta_0 + \theta_1 \cdot s(\theta_{11}x_1 + \theta_{12}x_2 + \theta_{10}) + \theta_2 \cdot s(\theta_{21}x_1 + \theta_{22}x_2 + \theta_{20})$$

Clearly

$\mathcal{F}_1^l \subseteq \mathcal{F}_2^l \subseteq \cdots \mathcal{F}_r^l \subseteq \cdots$

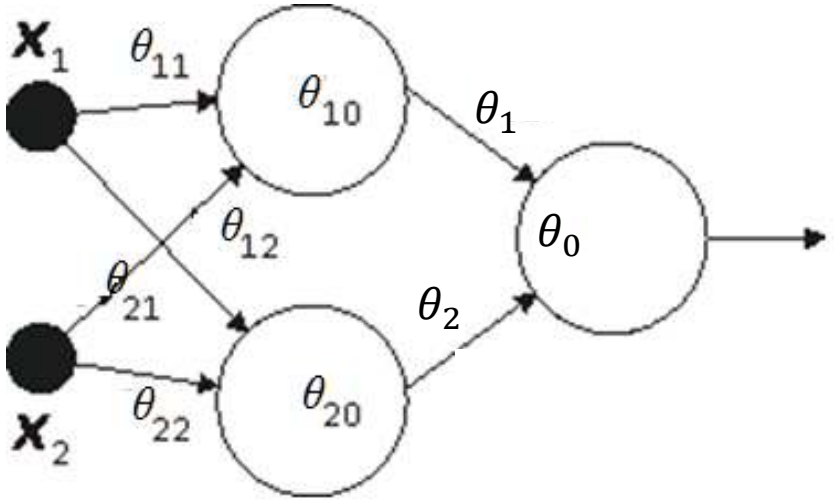
s =sigmoid function



Remark: The **higher** the k, the **higher** the "representational power" of the FNN.

Family of 2-layer FNNs

$\mathcal{F}^l = \mathcal{F}_1^l \cup \mathcal{F}_2^l \cup \cdots \mathcal{F}_r^l \cup \cdots$

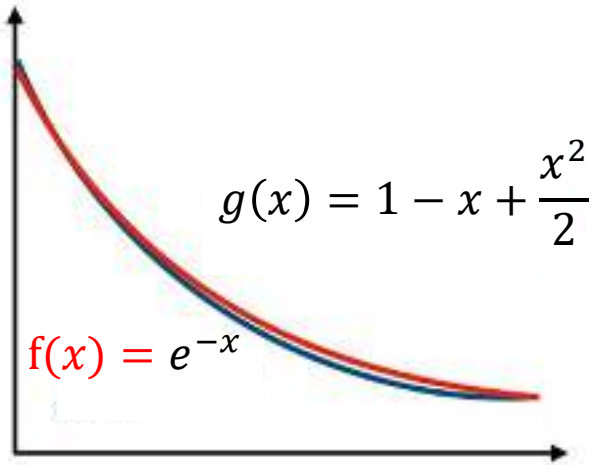


Universal approximation property (UAP)

Problem: Approximation of a **given** function $f(\boldsymbol{x})$ defined on a subset $S \subseteq R^l$ (that satisfies some rather **mild conditions**, e.g., **continuity**), by a **known form function** $g(\boldsymbol{x})$, so that, for $\varepsilon > 0$,

$$|f(\boldsymbol{x}) - g(\boldsymbol{x})| < \varepsilon, \qquad for\ all\ \boldsymbol{x} \in S.$$

Result: The
(a) **polynomial** family of functions,
(b) the **2-layer FNN** family of functions
(and several others)
share the **universal approximation property**:



For $f(\boldsymbol{x})$:
(a) there exists an integer $r = r(\epsilon)$ and an r-th order **polynomial** $g(\boldsymbol{x}; \boldsymbol{\theta}^{(r)})$, so that $\left|f(\boldsymbol{x}) - g(\boldsymbol{x}; \boldsymbol{\theta}^{(r)})\right| < \epsilon, \forall \boldsymbol{x} \in S$

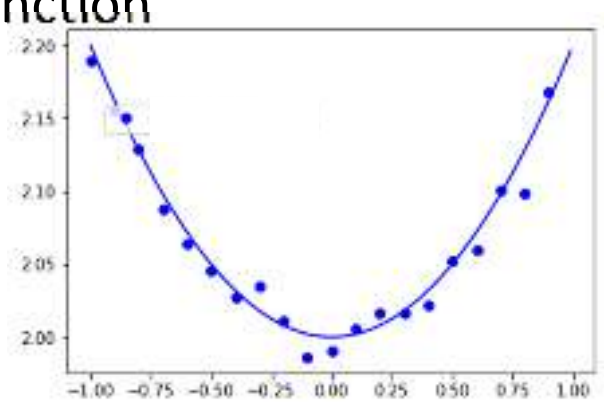
(b) there exists an integer $k = k(\epsilon)$ and a **2-layer FNN** $g(\boldsymbol{x}; \boldsymbol{\theta}^{(k)})$ with k hidden layer nodes, so that $\left|f(\boldsymbol{x}) - g(\boldsymbol{x}; \boldsymbol{\theta}^{(k)})\right| < \epsilon, \forall \boldsymbol{x} \in S$

Universal approximation property (UAP) in ML

How the universal approximation property can be useful in the ML framework?

In ML, the problem under study is modeled by an **unknown** function

$$y = f(\boldsymbol{x})$$



We have at our disposal a finite set of **noisy** points

$$S = \{(y_i, \boldsymbol{x}_i), i = 1, \dots, n\}$$

Main issue in ML: How can we **infer** the unknown function f from the data set S?

We work as follows:

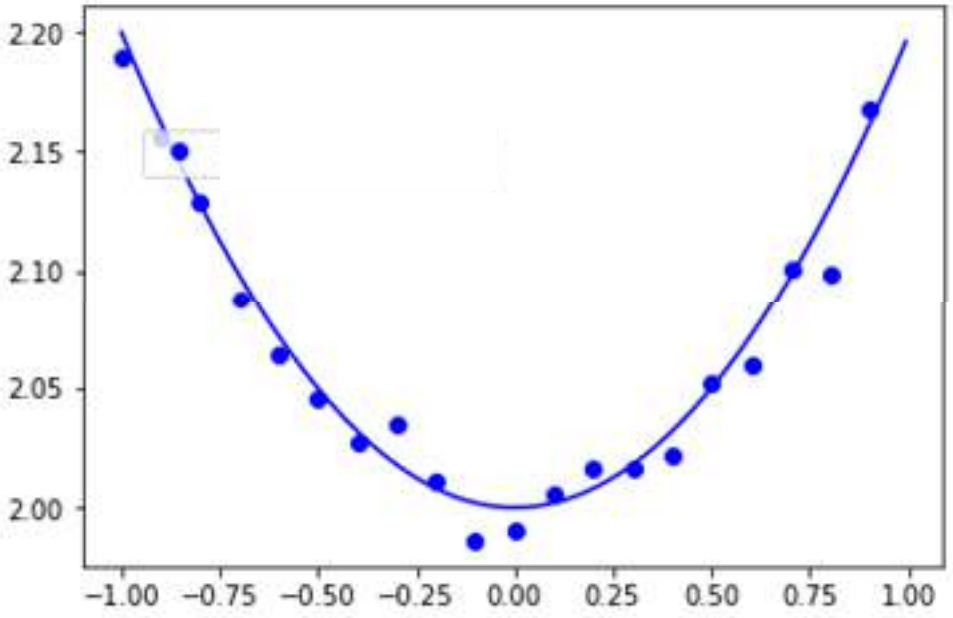
1. **Adopt** a UAP family of functions (we know that a member of this family can approximate arbitrarily well the unknown f). Assume the polynomial family.
2. **Select** an r and estimate the r -rh order polynomial, $g(\boldsymbol{x}; \boldsymbol{\theta}^{(r)})$, that best fits the data in S.
3. **If** the **result** is **satisfactory** then **stop**
4. **Otherwise** select a different value of r and **repeat** step **2**.

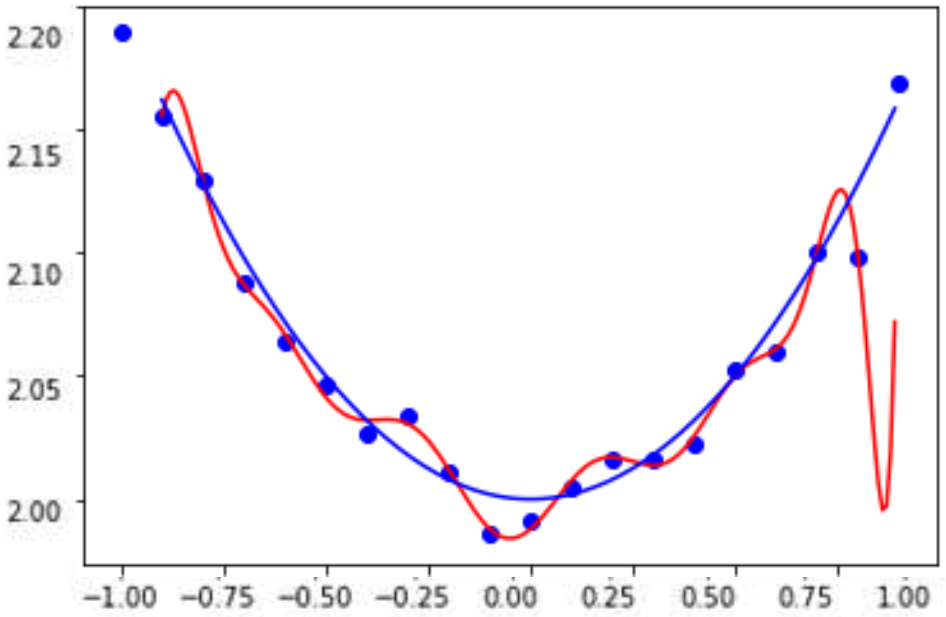
UAP and data finiteness in ML

But:

- **How** can **one be sure** that the **result** is **satisfactory**?

- Even though the approximation is good for the finite number of points, n, in S, who **guarantees** that the same holds for other points?





UAP and data finiteness in ML

Since all our info is the **finite** S we proceed as follows:

- **Split** randomly S into the training set S_{tr} (n_{tr} points) and the test set S_{te} (n_{te} points).

- **(A) Select** an r and estimate the r -rh order polynomial, $g(\boldsymbol{x};\boldsymbol{\theta}^{(r)})$, that **best fits** the data in S_{tr}, according to a **criterion** like

e.g., $L(y,\hat{y}) = (y-\hat{y})^2$
$\Omega(\cdot) = norm\ restr.\ func.$

$$J(\boldsymbol{\theta}^{(r)}) = \sum_{i=1}^{n_{tr}} L\left(y_i, g(\boldsymbol{x}_i;\boldsymbol{\theta}^{(r)})\right) + \lambda\,\Omega(\|\boldsymbol{\theta}^{(r)}\|)$$

- **Compute** the **error** associated with the **training** and the **test sets** resp.,

$$Err_{tr} = \sum_{(y_i,\boldsymbol{x}_i)\in S_{tr}} L\left(y_i, g(\boldsymbol{x}_i;\hat{\boldsymbol{\theta}})\right) \qquad Err_{te} = \sum_{(y_i,\boldsymbol{x}_i)\in S_{te}} L\left(y_i, g(\boldsymbol{x}_i;\hat{\boldsymbol{\theta}})\right)$$

- **If** both Err_{tr} and Err_{te} are "**low**" then **stop.**

- **Otherwise** select a different value of r and **repeat** from step **(A)**.

UAP and data finiteness in ML

*The cases where **not both** Err_{tr} and Err_{te} are "**low**"*

(a) "**high**" Err_{tr} and "**high**" Err_{te}

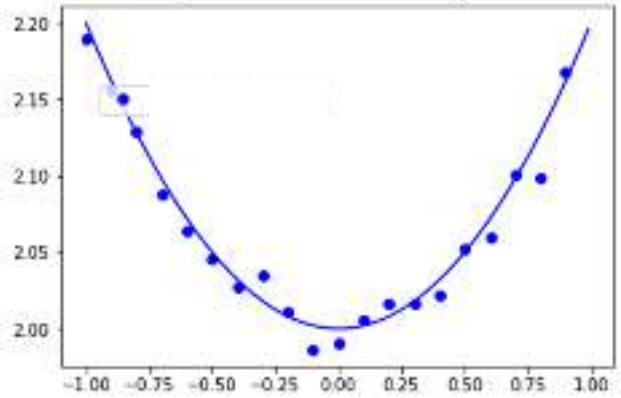
The **model** has **not enough "representational power"** to approx. the true model

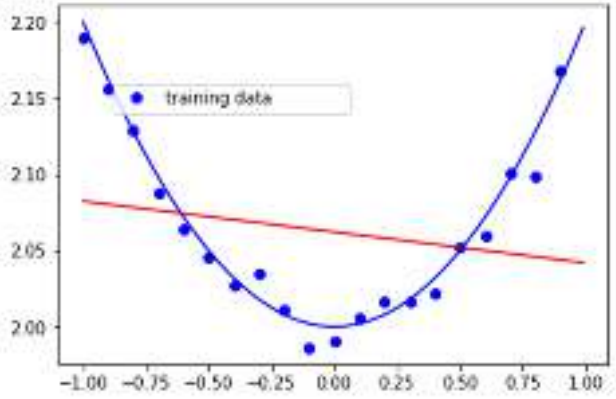
⇒ **Try** a **more complex** model.

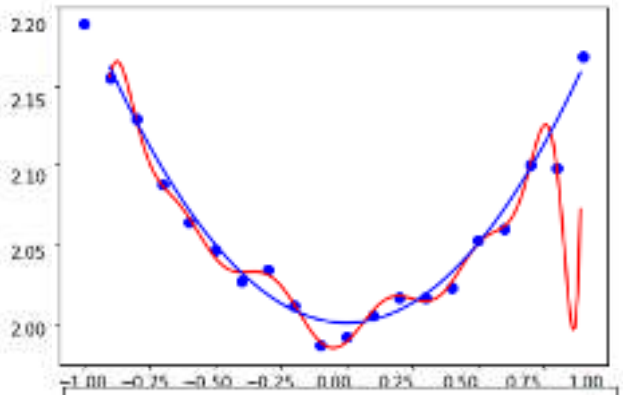
(b) "**low**" Err_{tr} and "**high**" Err_{te}

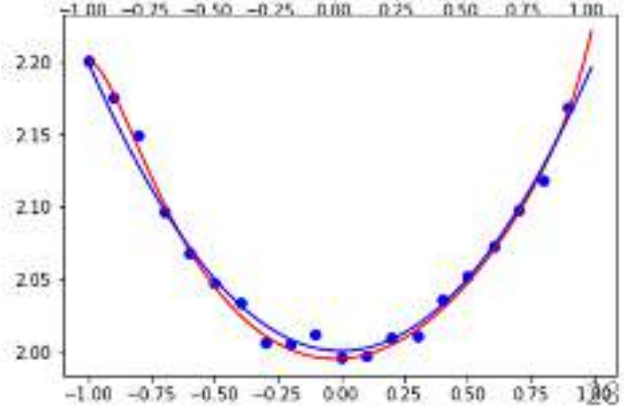
The **model** has **excessively enough "representational power"** to approx. the true model

⇒ **Try** a **less complex** model.









Golden rule: **Determine** the **member** of the adopted **UAP** whose "**representational power**" is close to the **true model complexity**

⇨**Both** Err_{tr} and Err_{te} are **low**.

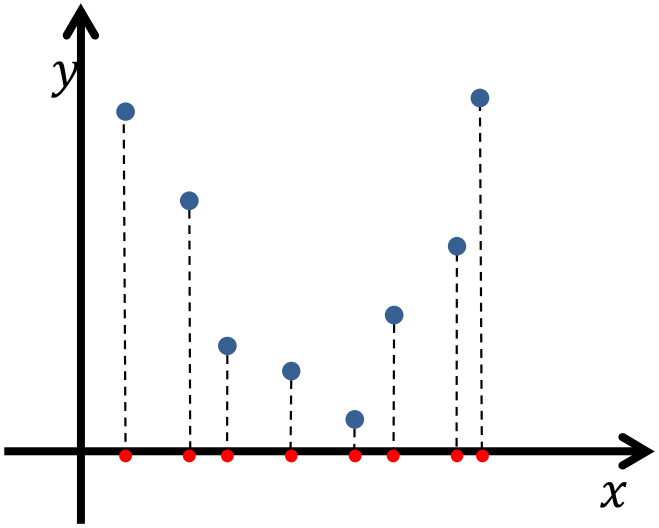
Assumptions implied but not explicitly uttered

For the **available data:**

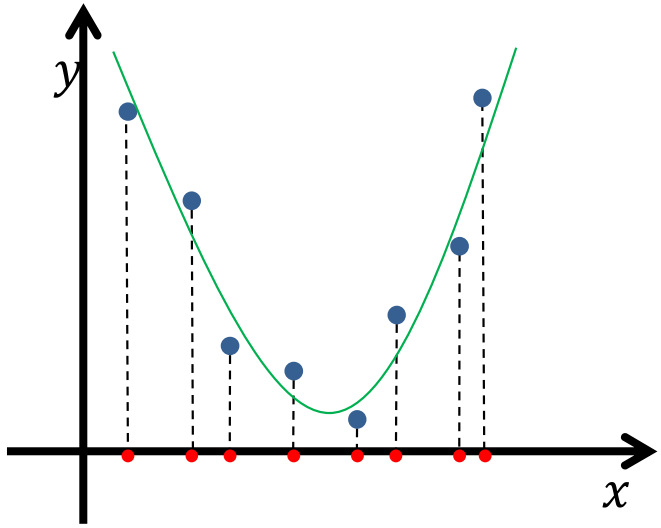
- They **cover** the whole "**range of activity**" of the **phenomenon** under study.
- They are "**enough**" to make **reliable estimations** of the involved model **parameters**.

For the **unknown function f:**

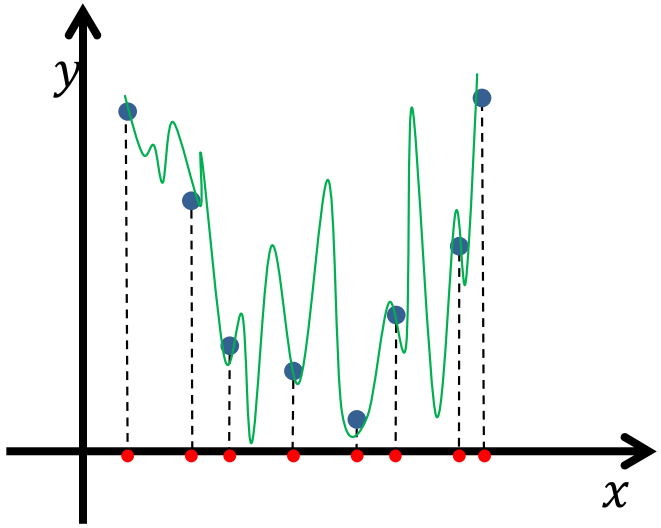
It is "**smooth**". The **smoother** the f, the more reliable the interpolation estimates are.



The data



"**Smoothed**" scenario



"**Non smoothed**" scenario

Explainability and interpretability of results

If we **know** the **true model** associated with the **phenomenon** under study (that is, know the "**physics**" of the phenomenon),
then we can **understand** the **reasoning** behind the **model results**.

What if the **true model** is **unknown**?
Adopt a **UAP** family of **functions** and we **determine** the **instance** that "**best**" **approximates** the **unknown model.**

However, in general, the **mathematical formulation** of the **true model** is **not the same** with that of the adopted **UAP family**.
⇒ the **reasoning** behind a result is not **understandable** by humans.

Interpretability: the **degree** to which a **human** can **understand** the **reasoning** behind a **decision** taken by machine learning model.
Explainability: has to do with the **degree** to which the **internal mechanics** of a machine learning model can be **understood** by humans.

White-Box (WB) methods: their **internal rationale** can be easily **explained** in human terms and their decisions can be easily justified.
Black-Box (BB) methods: **more complex** and **more difficult** to **explain** how their **internal structure** works **BUT** usually **achieve** *higher* **performance** than WB methods

A final note

*"to **built** a machine learning **model**, use **every** piece of **information** you have at your disposal"*

(**data** and/or additional **knowledge** about the **"physics"** of the problem)

Contents

Part I: Introductory concepts

- **Model definition**
- Definition of ML via simplified examples
- Significant issues: *noise/uncertainty* existence, *number of data points*, *data finiteness*
- **How** to **approximate** a **true model** and relevant issues

Part II: Basic problems in ML

- Regression
- Supervised classification
- Unsupervised classification (clustering)

Part III: Some ML models in more detail

- k-nearest neighbor (*k-NN*) classifier
- *Decision trees* and *Random Forests*
- Feedforward Neural Networks (*FNNs*)
- Recurrent Neural Networks (*RNNs*) – Long-Short Term Memory (*LSTM*) networks
- Some basic concepts on *time series*

The main problems in ML

Regression:

A set of independent variables $\boldsymbol{x}$ are related with a set of dependent variables y via a function of the form $y = f(\boldsymbol{x}) + e$, where e is the "**uncertainty**".

Aims: Given a **finite** set of data/measurements $S = \{(y_i, \boldsymbol{x}_i), i = 1, \dots n\}$

1: **Approximate** $f(\cdot)$, based on S.

2: Given a value of $\boldsymbol{x}$ **estimate** the corresponding value of y, based on

(a) either the approximation of f based on S or **(b)** the S itself.

Regressor: The function $\boldsymbol{f}(\cdot)$.

Classification:

(I) **Supervised:**

A number of classes $\omega_1, \dots \omega_M$ to which **a set of "entities"** will be classified.

Aims: Given a **finite** set of data/measurements $S = \{(y_i, entity_i), i = 1, \dots, n\}$

1: **Model/Delineate** the classes, based on S.

2: Given an entity **assign** it to the "most appropriate" class.

Classifier: The **rule** that **assigns entities** to **classes**.

(II) **Unsupervised (Clustering):**

A **finite** set of **entities** $S = \{entity_i, i = 1, \dots N\}$ is available.

Aim:

Group (a) "similar" entities to the same cluster and **(b)** "less similar" entities to different clusters. The resulting **groups** correspond to **unlabeled classes** (**clusters**).

The problems in ML

Regression:

- (a) What is the model that generates the data?
- (b) How can we use the known data to perform estimations on unknown data?

Supervised Classification:

- (a) What is the model that generates/discriminates the data?
- (b) How can we use the known data to perform estimations on unknown data?

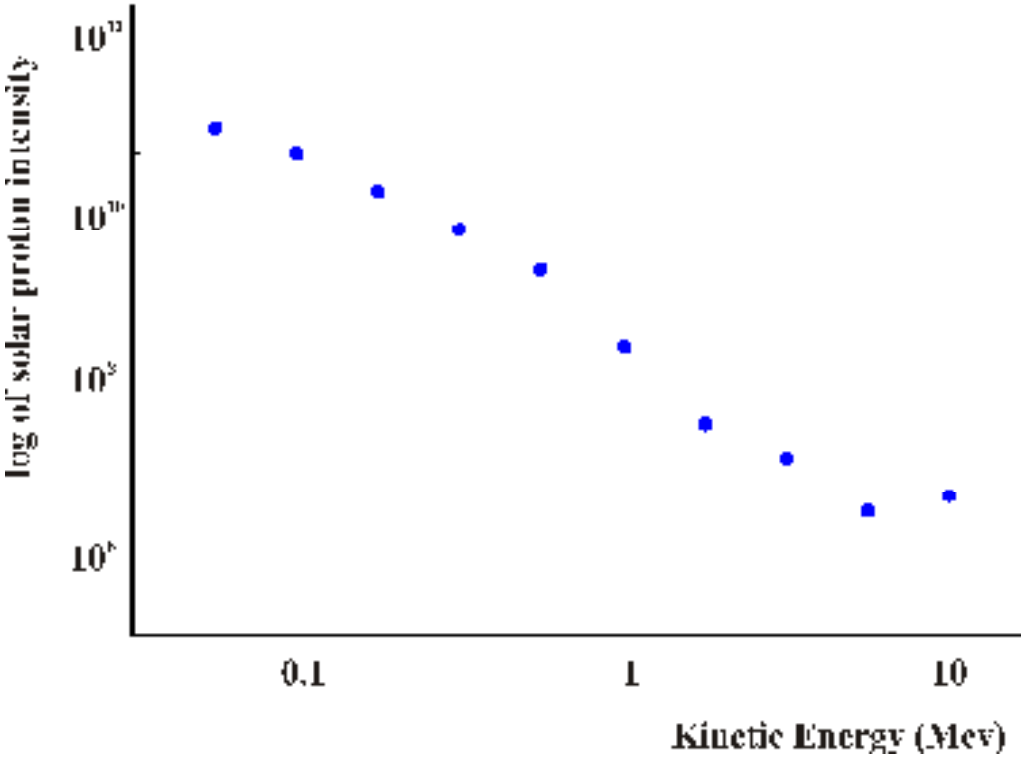
Clustering

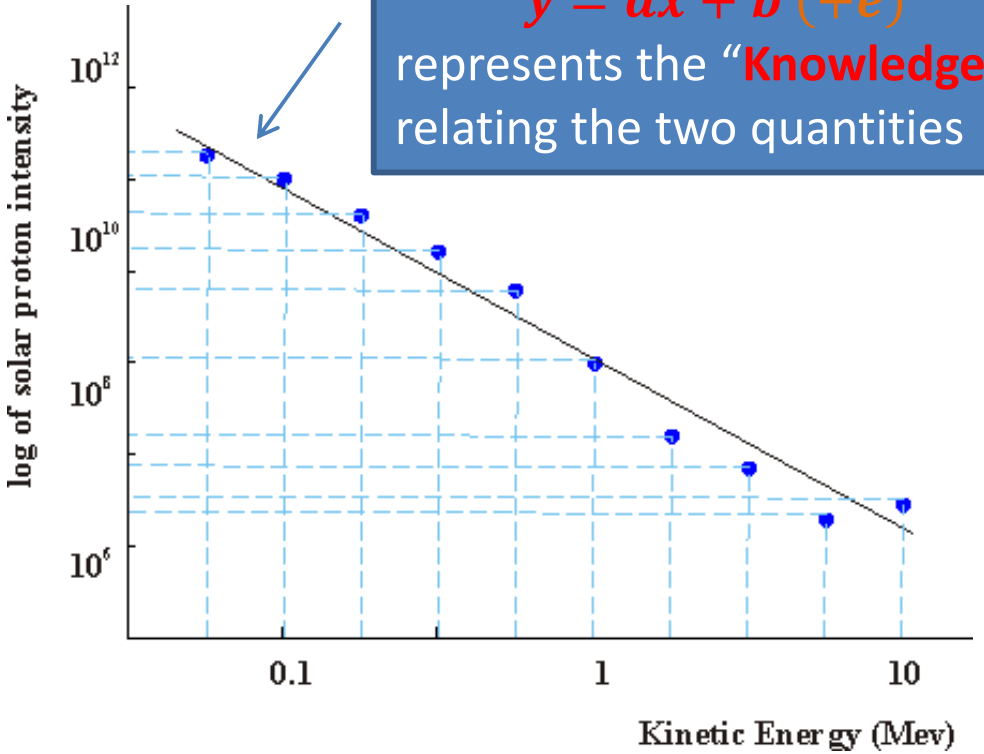
- What are the physical groups formed by the data?

The problems in ML

Regression

(a) What is the model that generates the data?





The line
$y = ax + b\ (+e)$
represents the "**Knowledge**" relating the two quantities

Regression:
- (a) What is the model that generates the data?
- (b) How can we use the known data to perform estimations on unknown data?

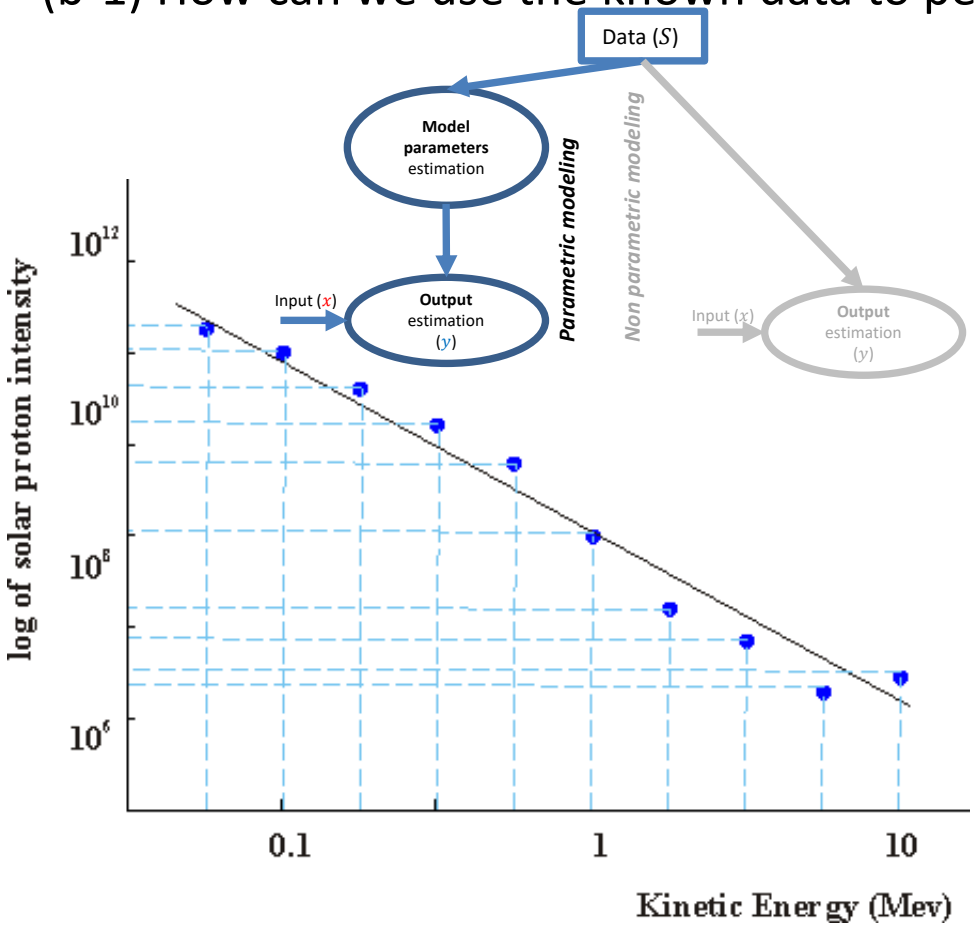
Supervised Classification:
- (a) What is the model that generates/discriminates the data?
- (b) How can we use the known data to perform estimations on unknown data?

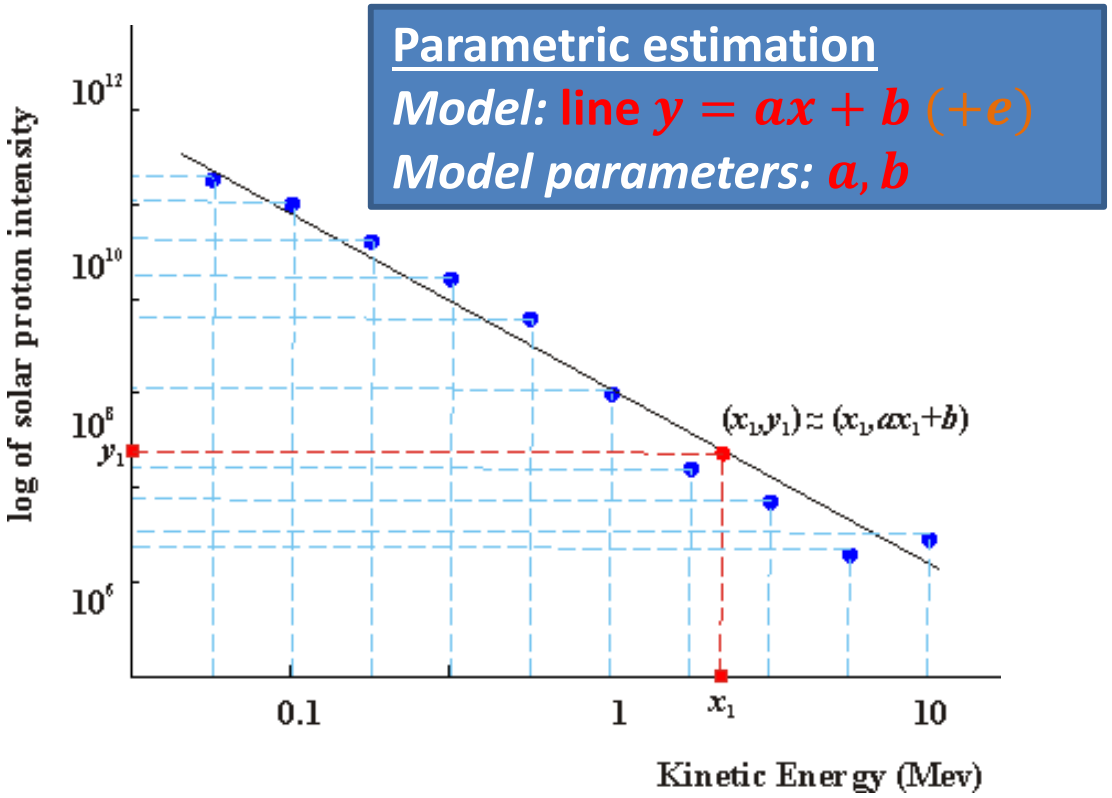
Clustering
- What is the general spread pattern of the data?

The problems in ML

Regression

(b-1) How can we use the known data to perform estimations on unknown data?





Regression:

- (a) What is the model that generates the data?
- (b) How can we use the known data to perform estimations on unknown data? (parametric estimation)

Supervised Classification:

- (a) What is the model that generates/discriminates the data?
- (b) How can we use the known data to perform estimations on unknown data?

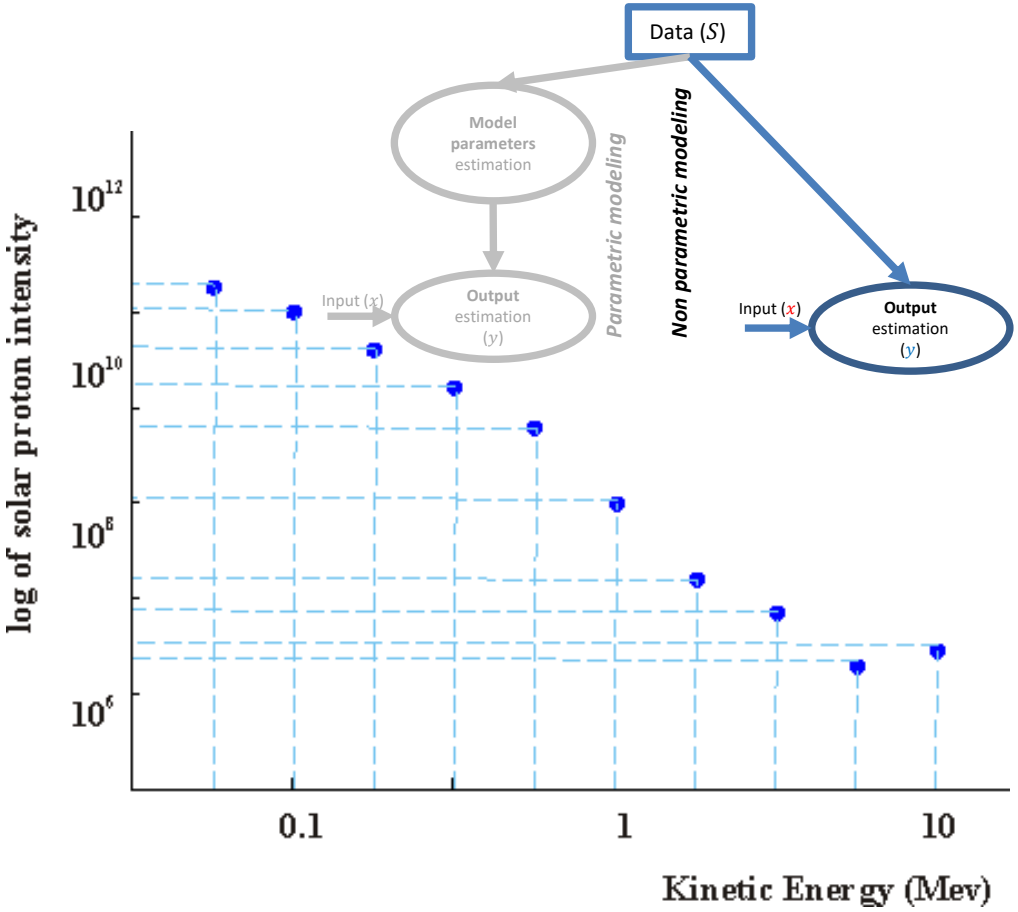
Clustering

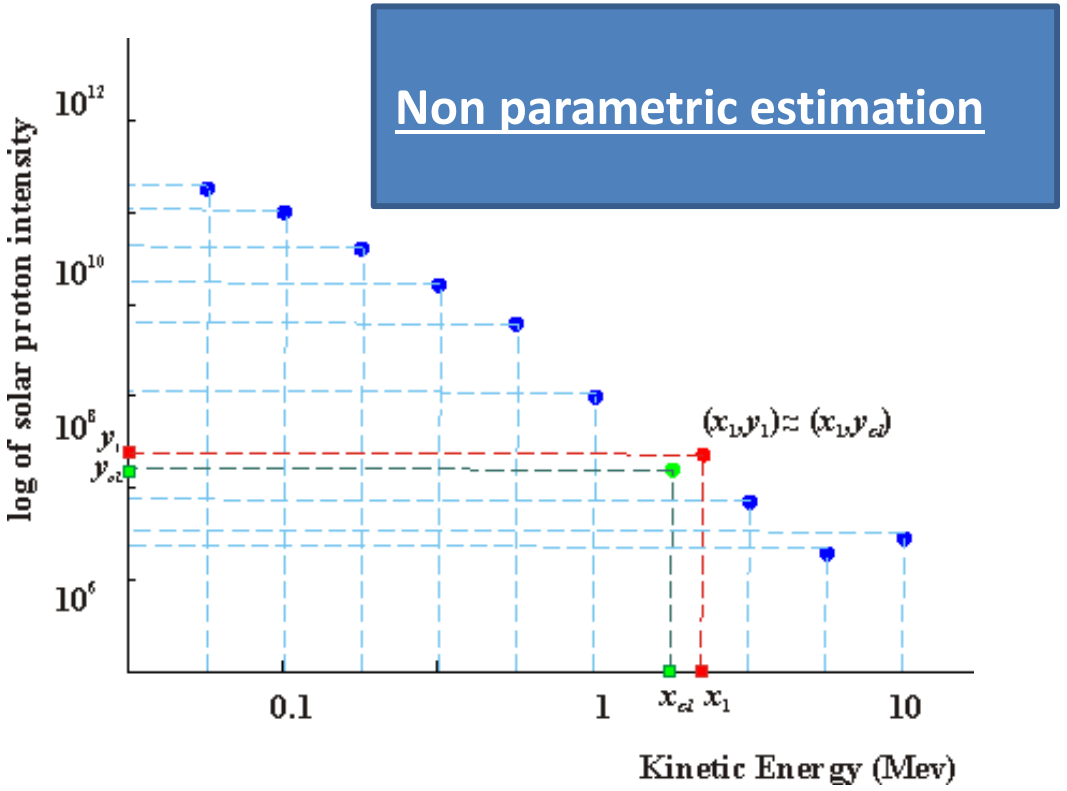
- What is the general spread pattern of the data?

The problems in ML

Regression

(b-2) How can we use the known data to perform estimations on unknown data?





Regression:

- (a) What is the model that generates the data?
- (b) How can we use the known data to perform estimations on unknown data? (non-parametric estimation)

Supervised Classification:

- (a) What is the model that generates/discriminates the data?
- (b) How can we use the known data to perform estimations on unknown data?

Clustering

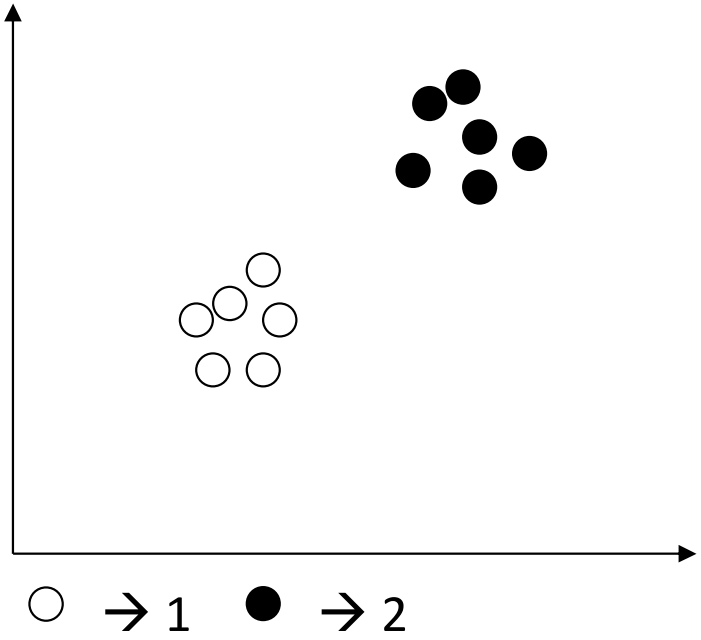
- What is the general spread pattern of the data?

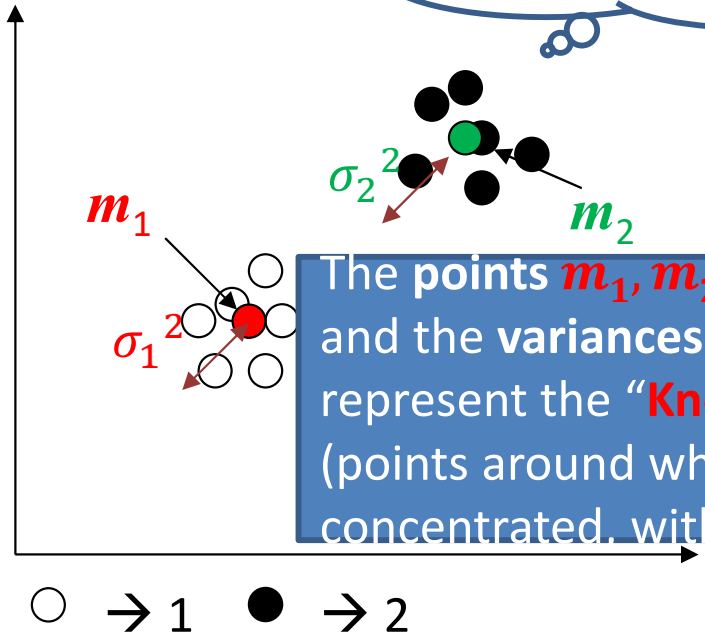
The problems in ML

Supervised Classification

(a) What is the model that generates the data?

A number of classes $\omega_1, \ldots \omega_M$ to which **a set of "entities"** will be classified





The **points** m_1, m_2 and the **variances** $\sigma_1{}^2, \sigma_2{}^2$ represent the "**Knowledge**" (points around which classes are concentrated, with given variance)

Regression:
- (a) What is the model that generates the data?
- (b) How can we use the known data to perform estimations on unknown data?

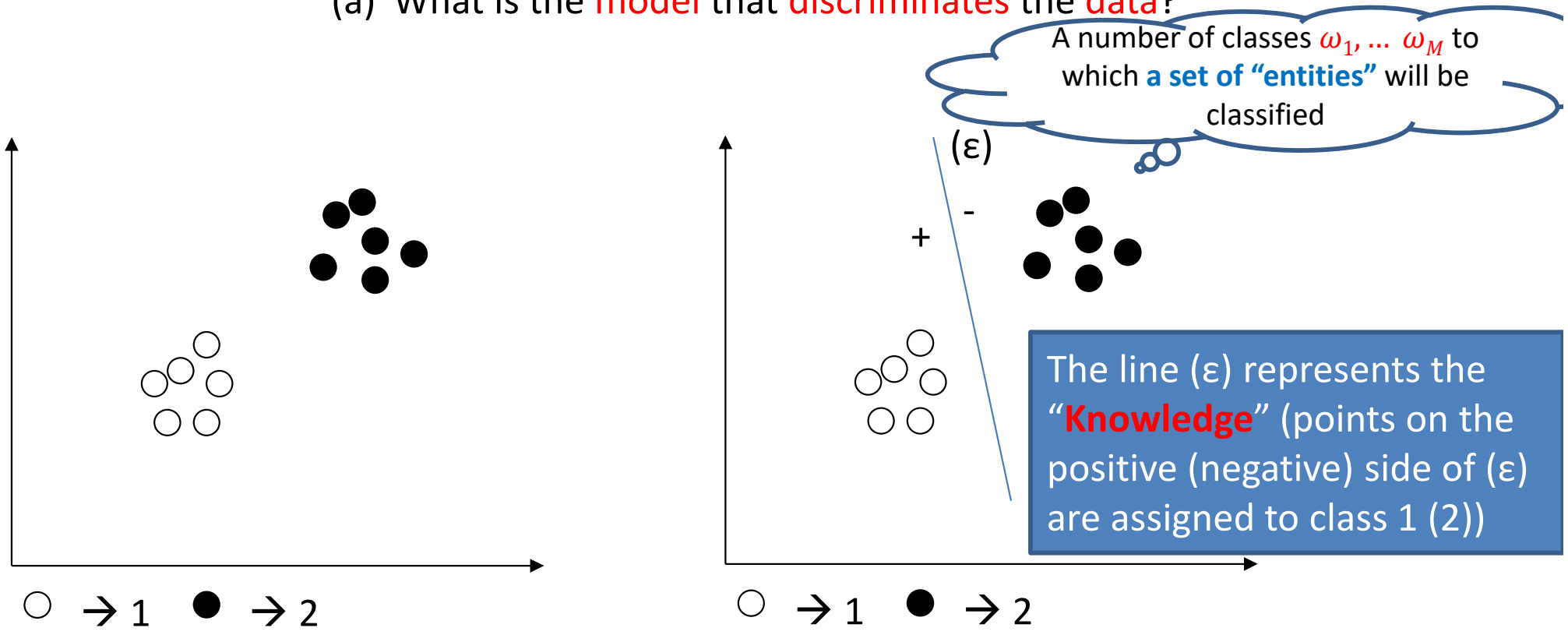
Supervised Classification:
- (a) What is the model that generates/discriminates the data?
- (b) How can we use the known data to perform estimations on unknown data?

Clustering
- What is the general spread pattern of the data?

The problems in ML

Supervised Classification

(a) What is the model that discriminates the data?



The line (ε) represents the "**Knowledge**" (points on the positive (negative) side of (ε) are assigned to class 1 (2))

Regression:
- (a) What is the model that generates the data?
- (b) How can we use the known data to perform estimations on unknown data?

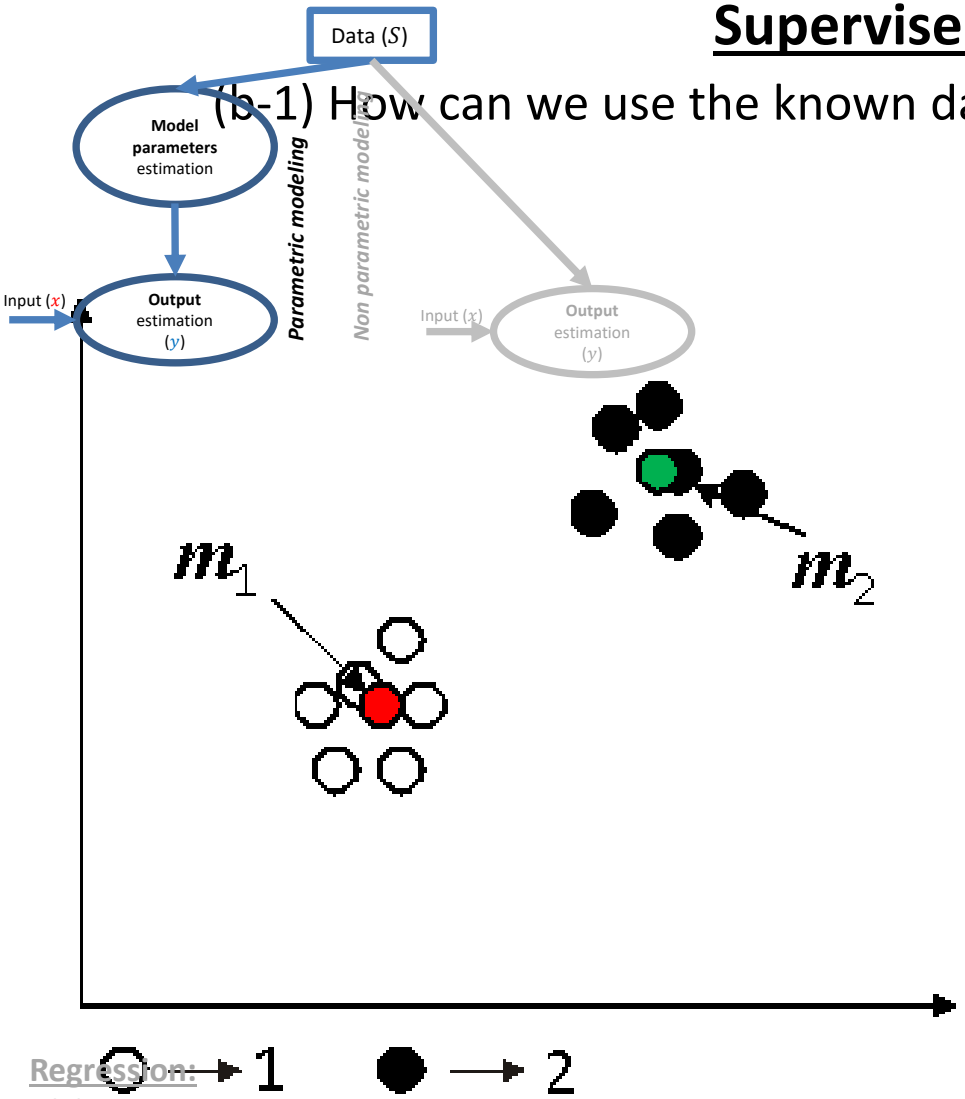
Supervised Classification:
- (a) What is the model that generates/discriminates the data?
- (b) How can we use the known data to perform estimations on unknown data?

Clustering
- What is the general spread pattern of the data?

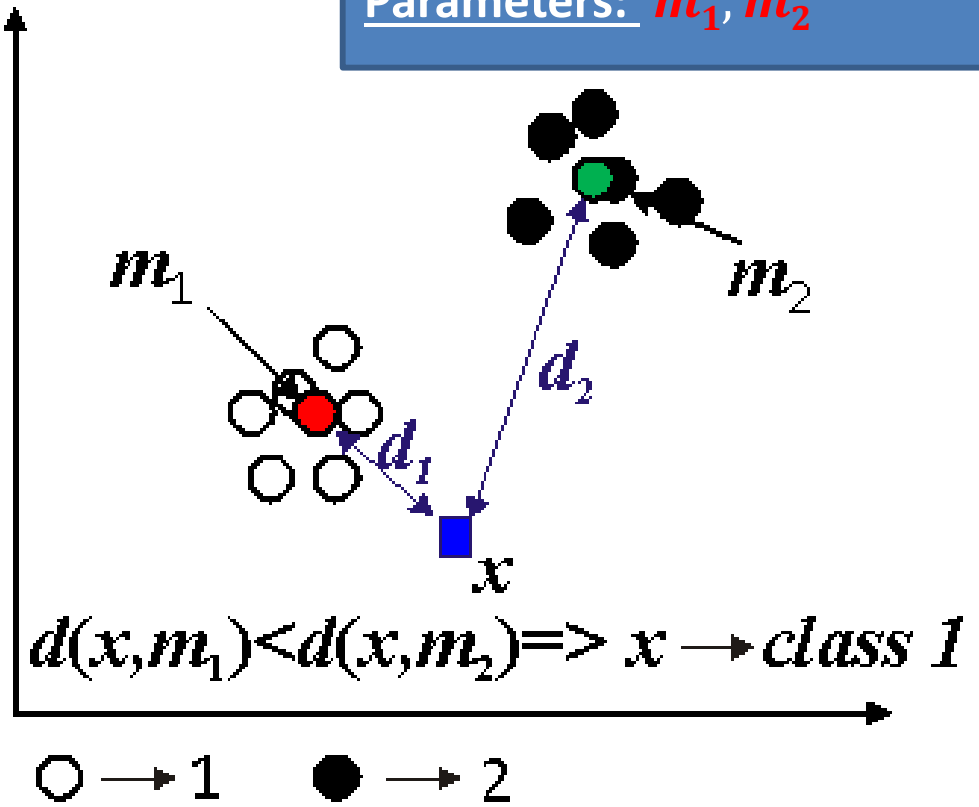
The problems in ML

Supervised Classification

(b-1) How can we use the known data to perform estimations on unknown data?



Parametric estimation
Parameters: m_1, m_2



Regression:
- (a) What is the model that generates the data?
- (b) How can we use the known data to perform estimations on unknown data?

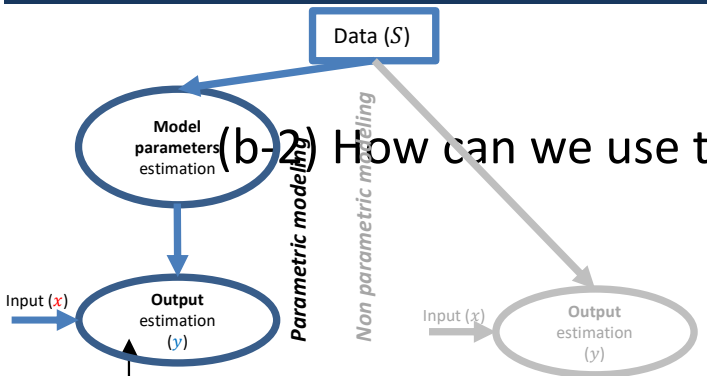
Supervised Classification:
- (a) What is the model that generates/discriminates the data?
- (b) How can we use the known data to perform estimations on unknown data? (parametric estimation: generation)

Clustering
- What is the general spread pattern of the data?

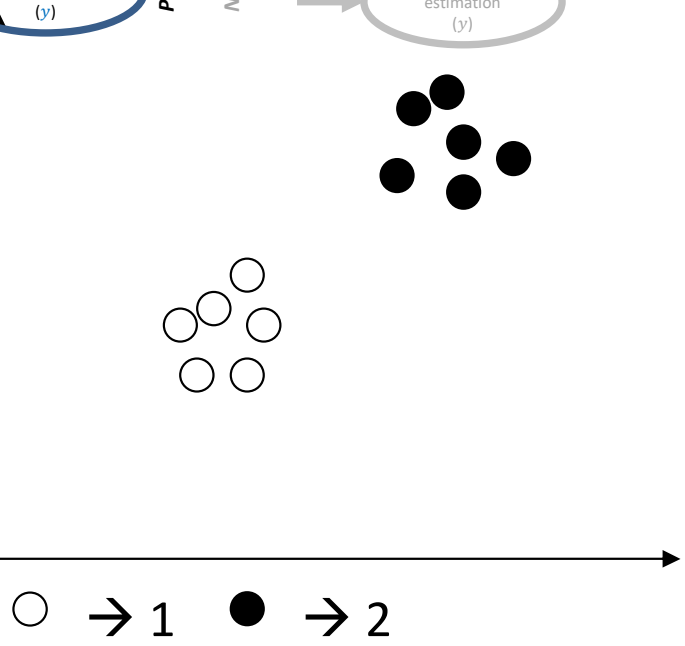
The problems in ML

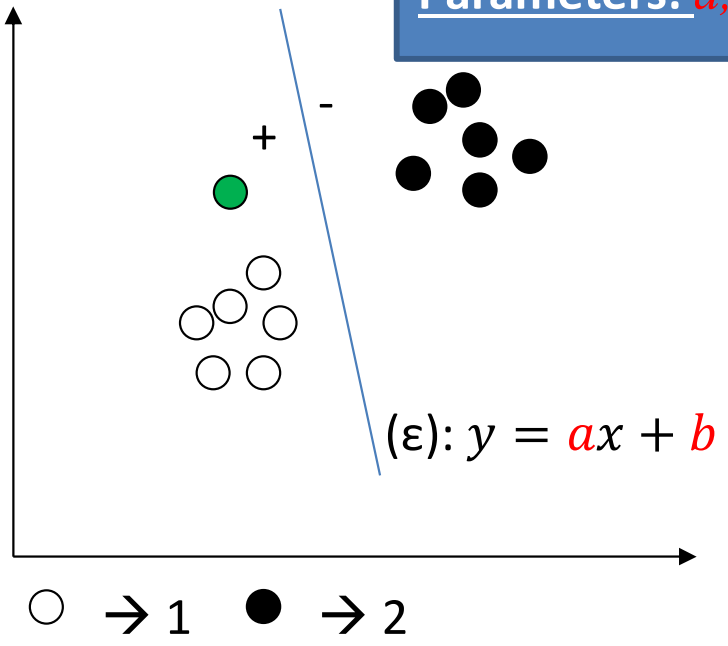
Supervised Classification

(b-2) How can we use the known data to perform estimations on unknown data?



Parametric estimation
Parameters: $a,\ b$





Assign the unknown point to class 1, since it lies on the positive side of (ε)

Regression:
- (a) What is the model that generates the data?
- (b) How can we use the known data to perform estimations on unknown data?

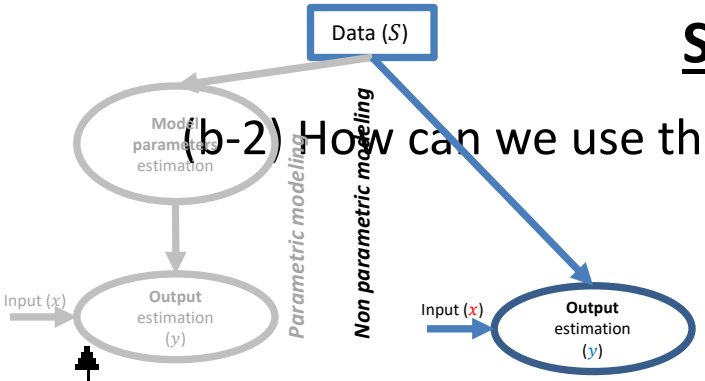
Supervised Classification:
- (a) What is the model that generates/discriminates the data?
- (b) How can we use the known data to perform estimations on unknown data? (parametric estimation: discrimination)

Clustering
- What is the general spread pattern of the data?

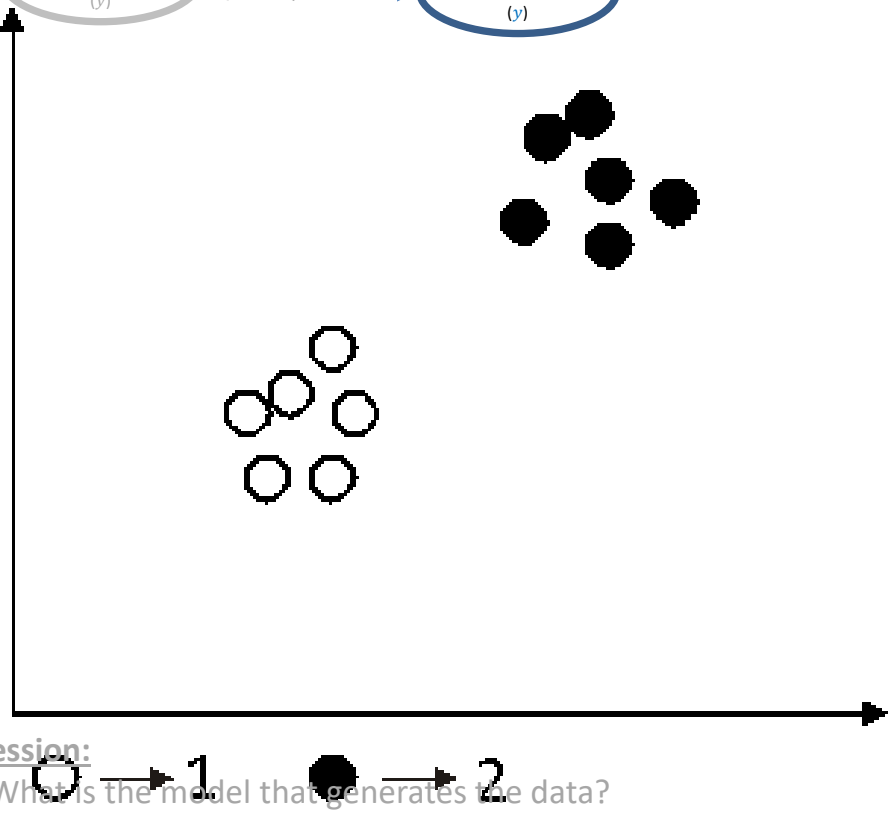
The problems in ML

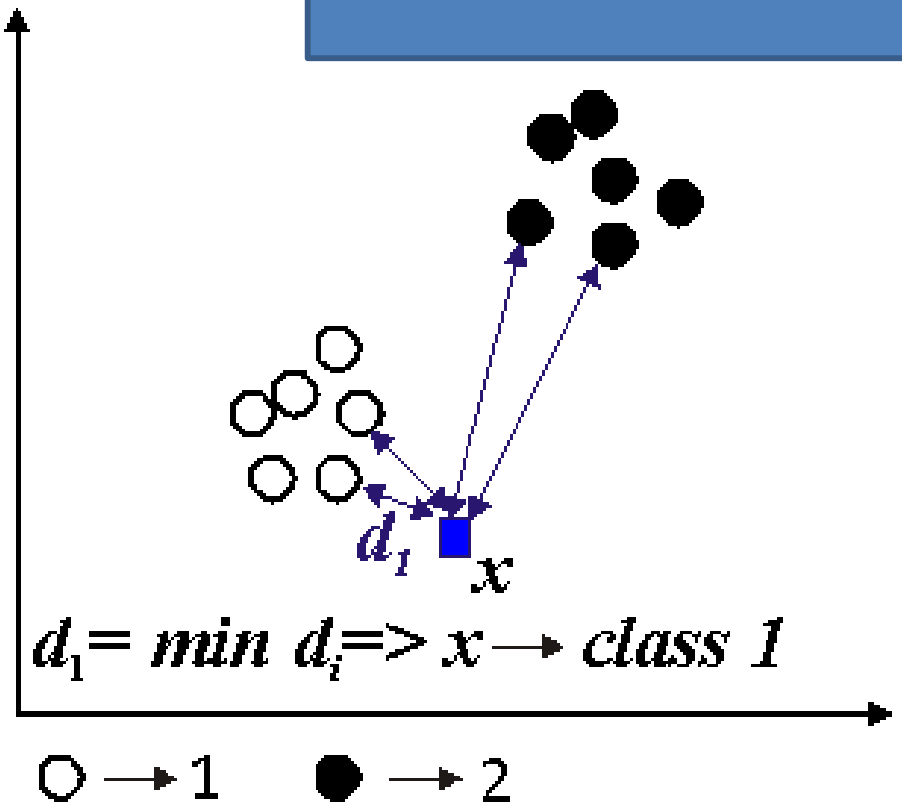
Supervised Classification

(b-2) How can we use the known data to perform estimations on unknown data?



Non Parametric estimation





Regression:
- (a) What is the model that generates the data?
- (b) How can we use the known data to perform estimations on unknown data?

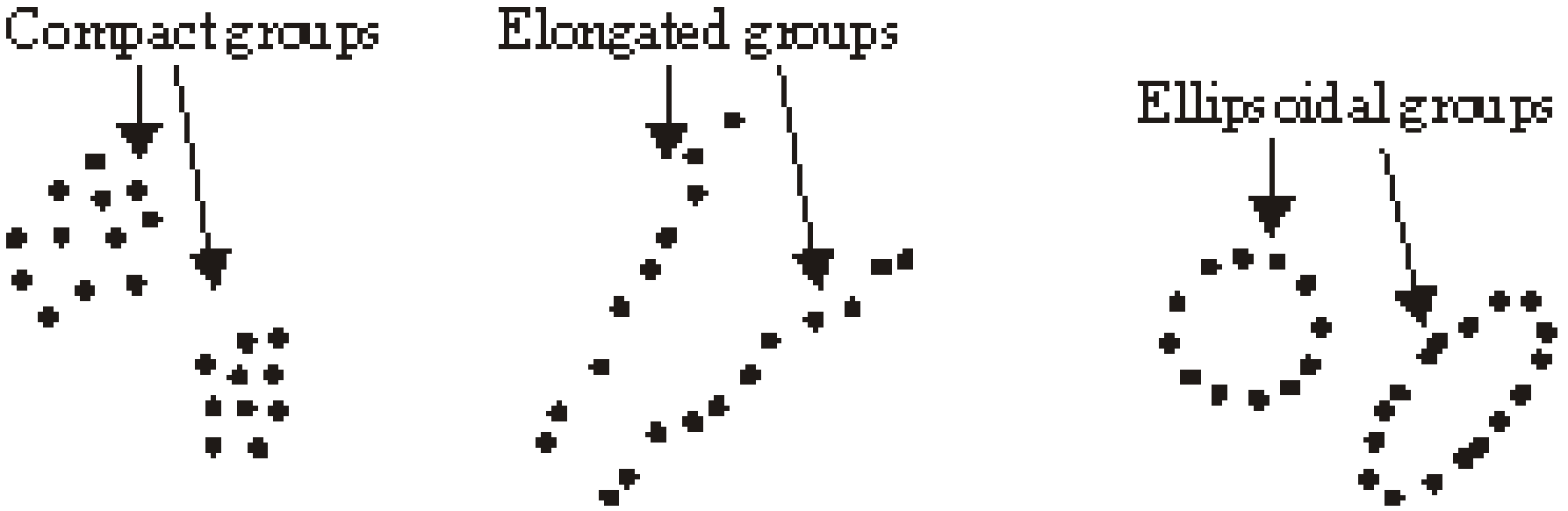
Supervised Classification:
- (a) What is the model that generates/discriminates the data?
- (b) How can we use the known data to perform estimations on unknown data? (non-parametric estimation)

Clustering
- What is the general spread pattern of the data?

The problems in ML

Unsupervised classification (Clustering)

What are the physical groups formed by the data?



Regression:
- (a) What is the model that generates the data?
- (b) How can we use the known data to perform estimations on unknown data?

Supervised Classification:
- (a) What is the model that generates/discriminates the data?
- (b) How can we use the known data to perform estimations on unknown data?

Clustering
- What is the general spread pattern of the data?

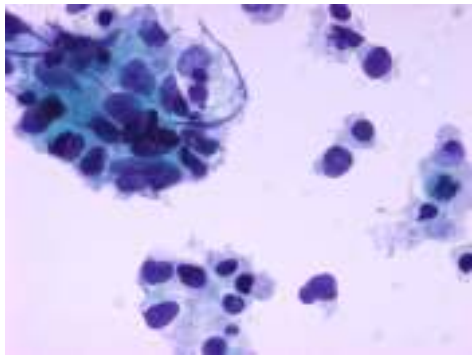
Entities

NOTE:

•In **classification/clustering** the main underlying concept is that of the **entity**.

•Entities (as well as the classes) are defined according to the classification/clustering application at hand.

Classification applications



Cytology

Entities: *human cells nuclei*

Classes: *benign*, *malignant*

Industry production line

Entities: E.g., *screws*

Classes: *defective*, *non-defective*



Optical character recognition

Entities: Alphanumerical characters, punctuation

Classes: *26 classes of lowercase lett.*+*26 classes of uppercase lett.*+*10 classes of digits* +*punctuation classes*.

Entities

NOTE:

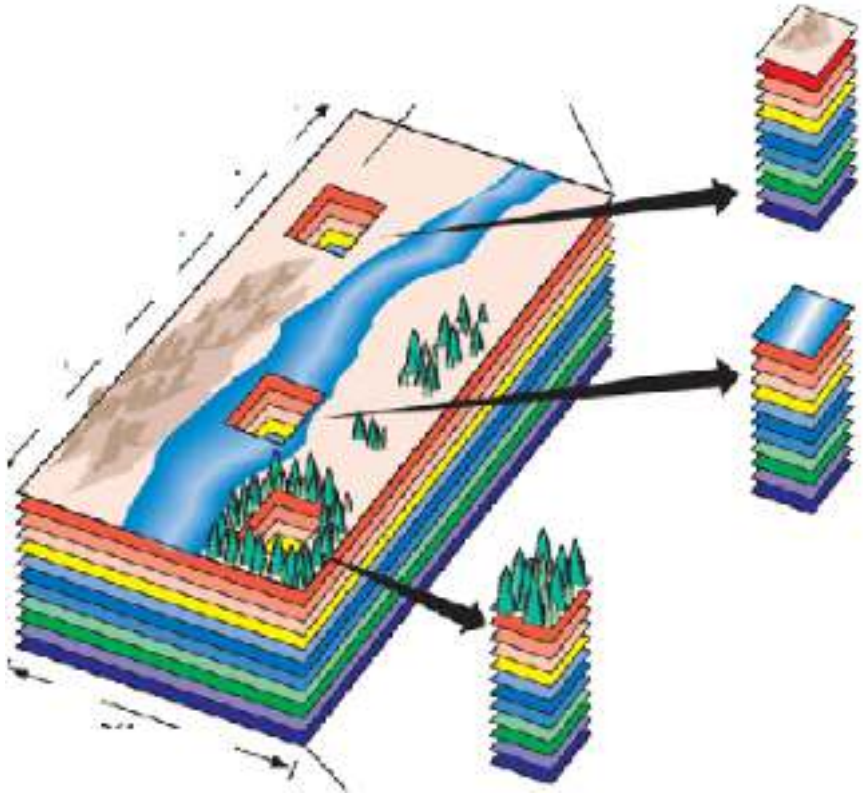
- In **classification/clustering** the main underlying concept is that of the **entity**.
- Entities are defined according to the classification/clustering application at hand.

Classification applications (cont.)

Hyperspectral image classification

Entities: *image pixels*

Classes: *area types* (e.g., soil, water, vegetation etc).



Entity representation

AN IMPORTANT ISSUE: ENTITY REPRESENTATION

How the entities involved an a **classification problem** are represented?

Since we deal with statistical processing methods, we need "numerical" representations.

We work as follows:
For each entity, we measure a set of L quantities (measurements - features), **the same for all entities**.

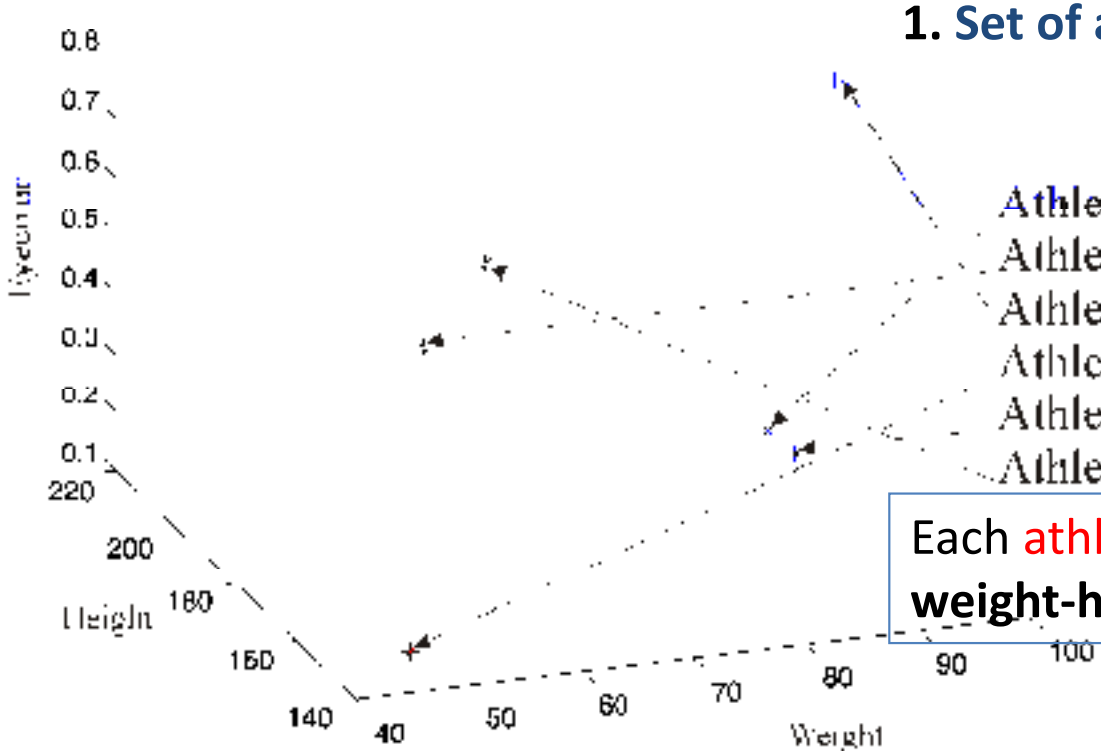
Then, each entity is represented as a **point** in an L-dimensional space (feature space).

Entity representation

AN IMPORTANT ISSUE: ENTITY REPRESENTATION (cont.)

Examples:

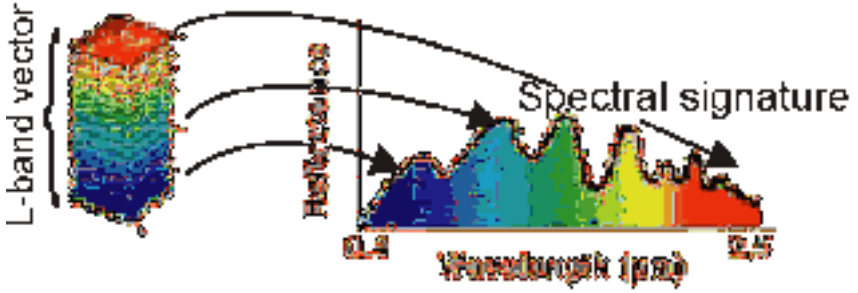
1. Set of athletes: Features weight, height, eyecolor



	Weight	Height	Eyecolor
Athlete1	90	190	.2
Athlete2	50	155	.6
Athlete3	100	205	.7
Athlete4	48	152	.1
Athlete5	95	200	.1
Athlete6	57	160	.7

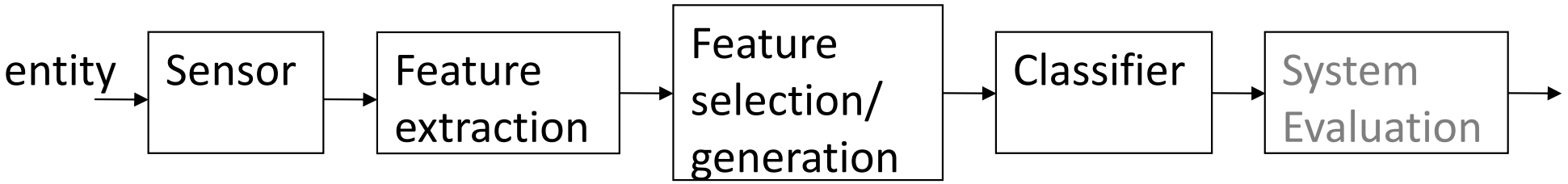
Each athlete is represented by a 3-dim. vector in the **weight-height-eyecolor** (feature) **space**.

2. Set of pixels (HSI): Features spectral measurements

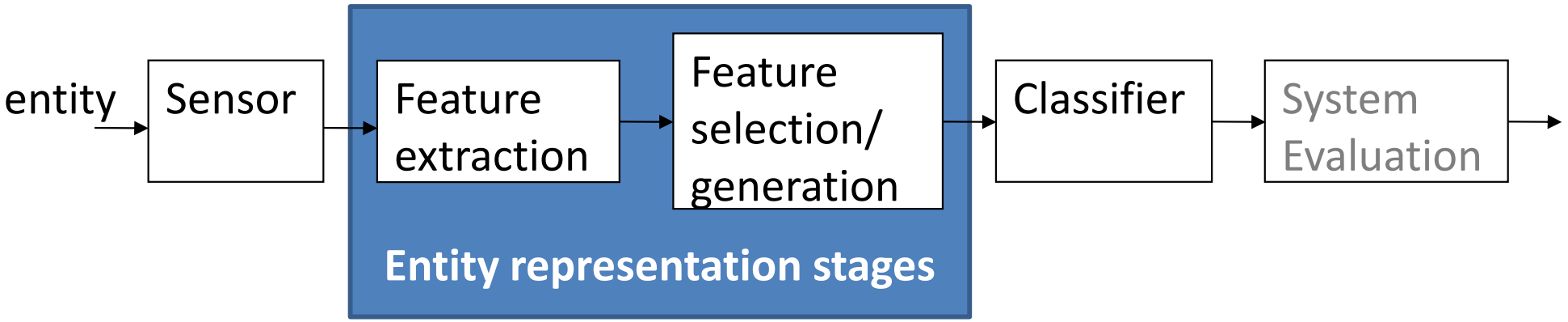


Each pixel is represented by an L-dim. vector (typically L~200) in the **spectral** (feature) **space**.

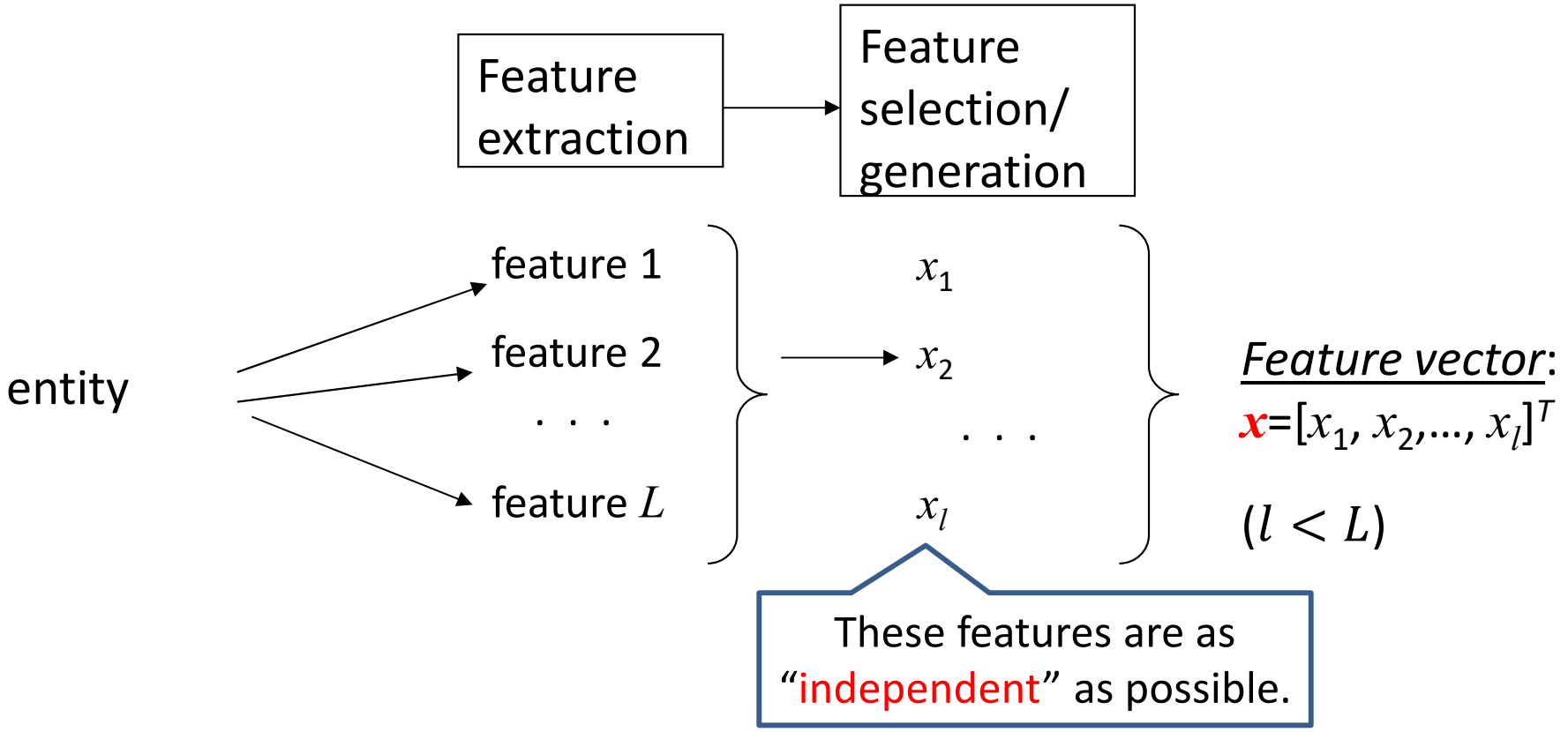
Typical (supervised) classification system structure



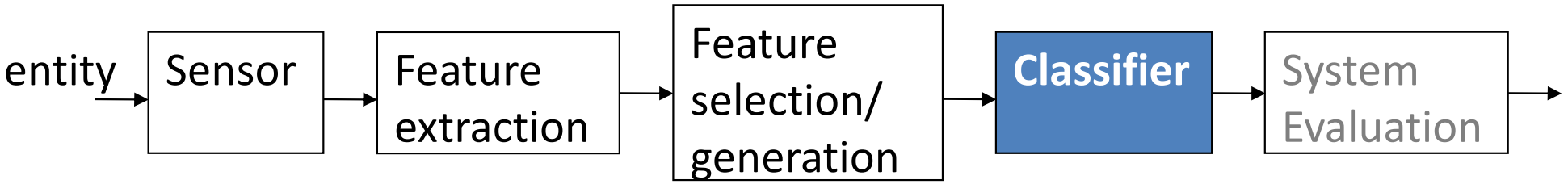
Typical (supervised) classification system structure



Entity representation (how it works)

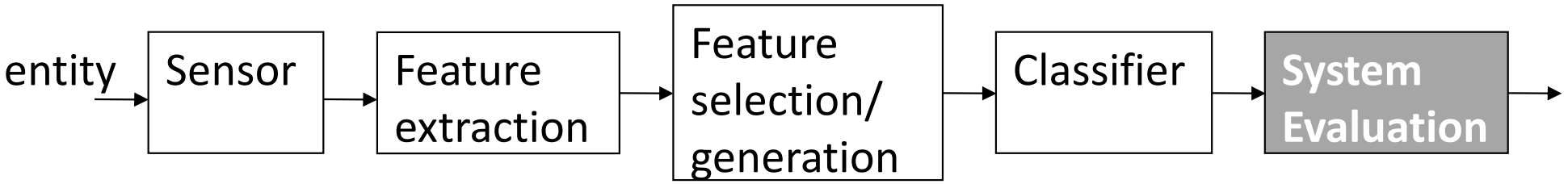


Typical (supervised) classification system structure



Classifier: *A set of rules/functions which* ***assigns*** *a given* ***feature vector*** $\boldsymbol{x}$ *(representing a specific entity),* ***to a class****.*

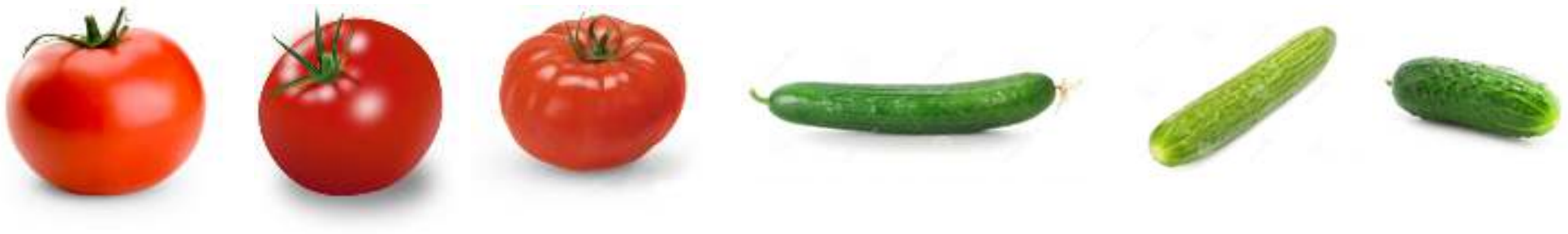
Typical (supervised) classification system structure



System evaluation: *Evaluation of the ability of the system to deal with the task under study (**active** only during the **designing phase**).*

Typical (supervised) classification system structure

Example: Design a classifier that recognizes if an object (entity) is a tomato or a cucumber.
Entities - Objects: tomatoes, cucumbers
Classes: «***tomato***» , «***cucumber***»



Design:

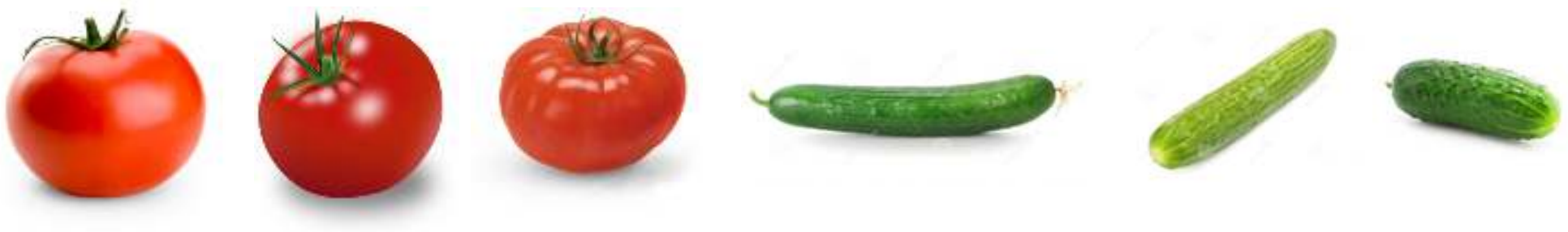
Sensor: E.g. a *photo-camera*. Every object is represented by its **RGB** photo.

Feature selection/generation:

Classification:

Typical (supervised) classification system structure

Example: Design a classifier that recognizes if an object (entity) is a tomato or a cucumber.
Entities - Objects: tomatoes, cucumbers
Classes: «***tomato***» , «***cucumber***»



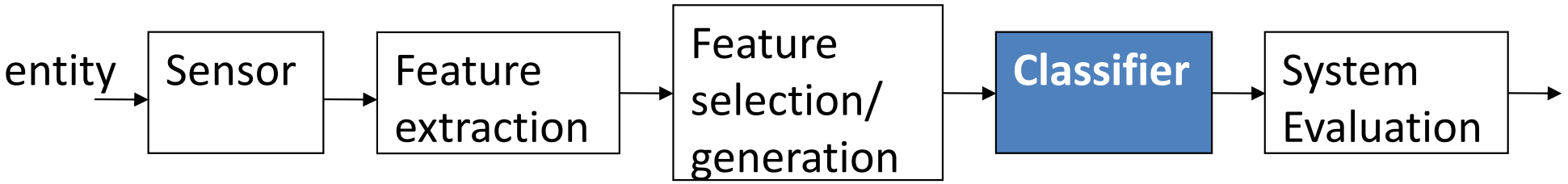
Design:

Sensor: E.g. a *photo-camera*. Every object is represented by its **RGB** photo.

Feature selection/generation: E.g. (a) color (value of R (red) band) και (b) shape (λ = major to minor axis ratio of the ellipse that approximates the shape of the entity under study. It is $\lambda \cong 1$, for circular shapes, $\lambda \gg 1$, for elongated ellipses). (**Representing** each entity via the two-dimensional vector $[R, \lambda]$)

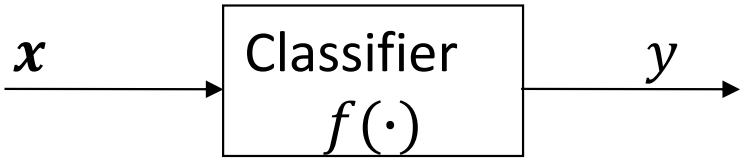
Classification: Object classification based on the corresponding feature vector $[R, \lambda]$, via the rule: **If** $R \gg$ **AND** $\lambda \cong 1$ **then** «tomato», **else** «cucumber».

Typical (supervised) classification system structure



Classifier: *A set of rules/functions which* ***assigns*** *a given* ***feature vector*** $\boldsymbol{x}$ *(representing a specific entity),* ***to a class****.*

Classifier design issue: Representing schematically a classifier as



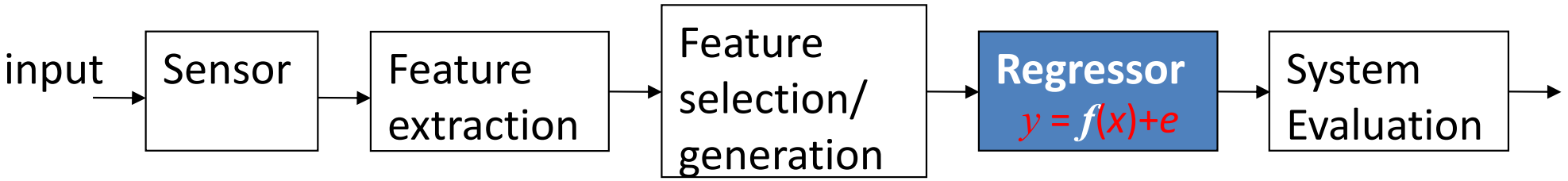
$$y = f(\boldsymbol{x})$$

how can the mapping f(.) be learned?

$f(\cdot)$: is learned based on a given **finite** set X of *data* (*feature*) *vectors* ($\boldsymbol{x}_n$) for which the associated *output* y_n is known.

$$S = \{(y_n, \boldsymbol{x}_n), \boldsymbol{x}_n \in R^l, y_n \in \{1,2,\dots,M\}, n = 1,\dots,N\}$$

Typical regression system structure



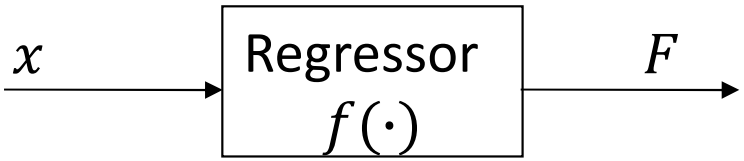
Regressor: *The **rule/model/functions** that, given an input vector x, it estimates the associated output vector y.*

Typical regression system structure

Example 1: On a given spring, we perform different forces (F) that cause different expansions (x), as shown in the following table:

x (cm)	F (N)
2	101
3	203
4	298
5	402
6	499

Design a **regressor** $F = f(x)$ that models for relation between F and x.



Design:

*In this case a **single feature** (**expansion** – independent variable) is encountered.*

Estimation of $f(\cdot)$:

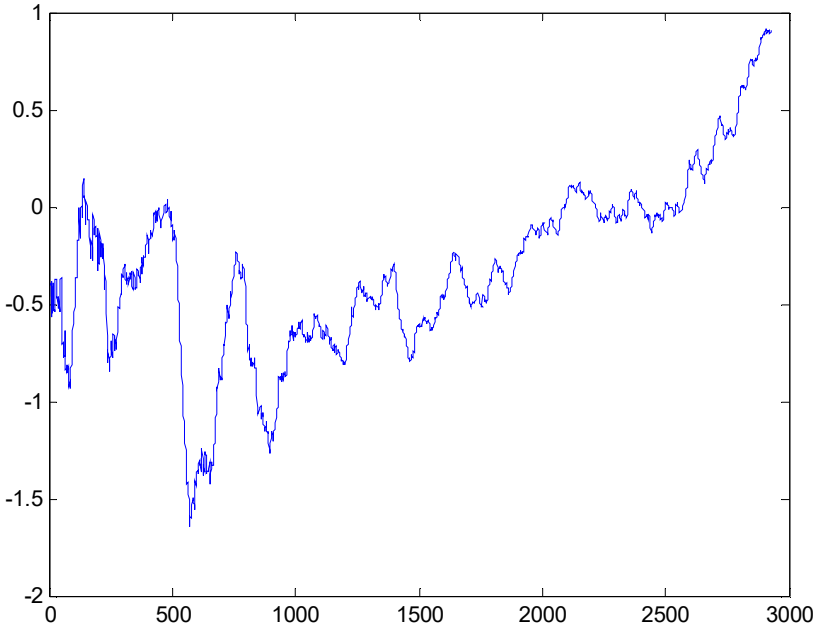
Based on the data set

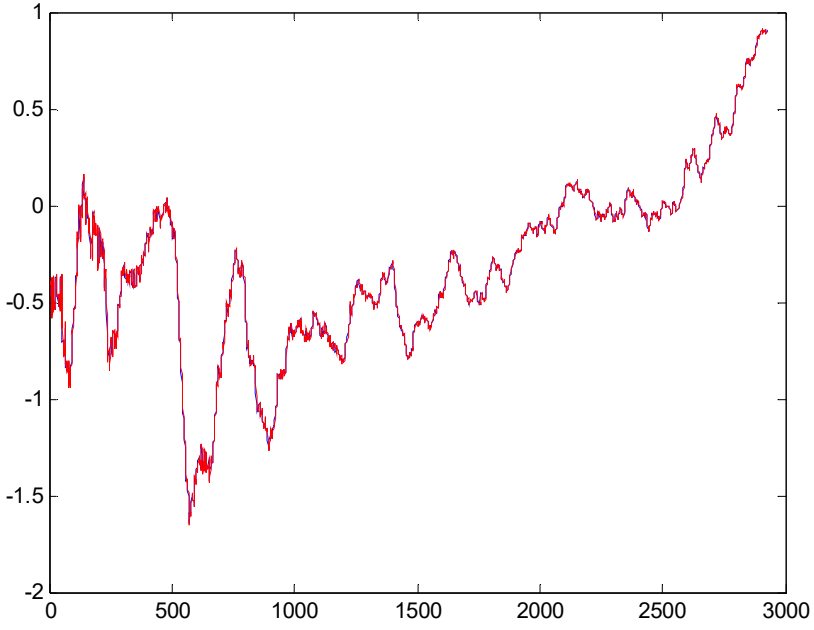
$$S = \{ (2, 101), (3, 203), (4, 298), (5, 402), (6, 499) \}$$

Prediction: Given a x^*, estimate $F^* = f(x^*)$ (e.g. $F^* = w \cdot x^*$)

Typical regression system structure

Example 2 (time series prediction): Design a **regressor** that performs **one-step ahead predictions** in a temperature changes time series $x(n), n = 1, \dots, N$.
(Temperatures are in Celsius and reported as anomalies relative to the Jan 1951-Dec 1980 average.)





Design:

Sensor: E.g. a thermometer.

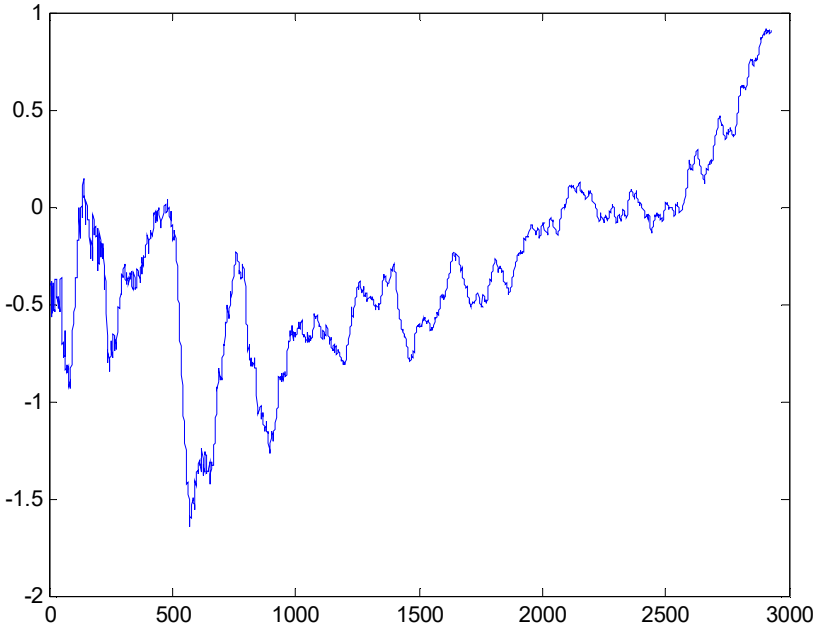
Feature generation: For prediction of the time series value at time t, the **feature vector** used is $\boldsymbol{x}(t) = [x(t-1), \dots, x(t-k)]$. ($k$-dimensional vector)

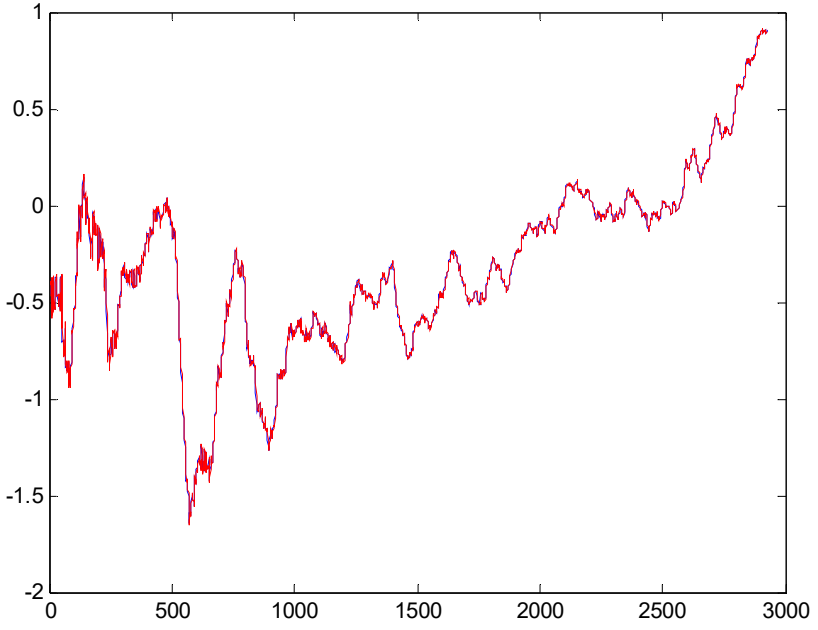
Prediction: $x_{est}(t) = f(\boldsymbol{x}(t))$ (e.g. $x_{est}(t) = \boldsymbol{w}^T\boldsymbol{x}(t) + w_0$)

Typical regression system structure

Example (cont.): Design a **regressor** that performs **one-step ahead predictions** in a temperature changes time series $x(n), n = 1, \dots, N$.

(Temperatures are in Celsius and reported as anomalies relative to the Jan 1951-Dec 1980 average.)





Estimation of *f*(.):

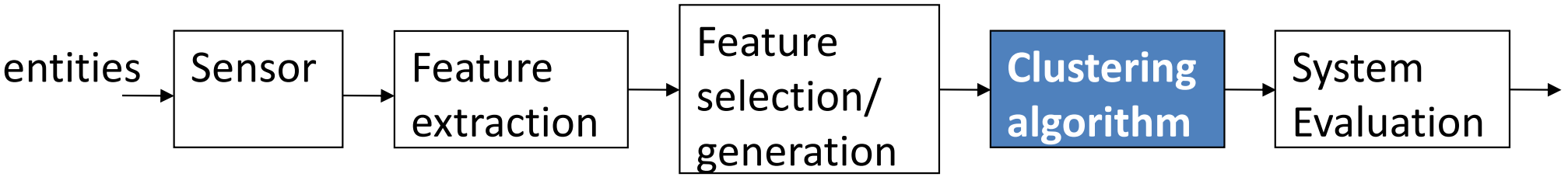
Via a data set of the form

$$S = \{((x(1), \dots, x(k)), x(k+1)), ((x(2), \dots, x(k+1)), x(k+2)), \dots, ((x(n-k), \dots, x(n-1)), x(n))\ \}$$

or

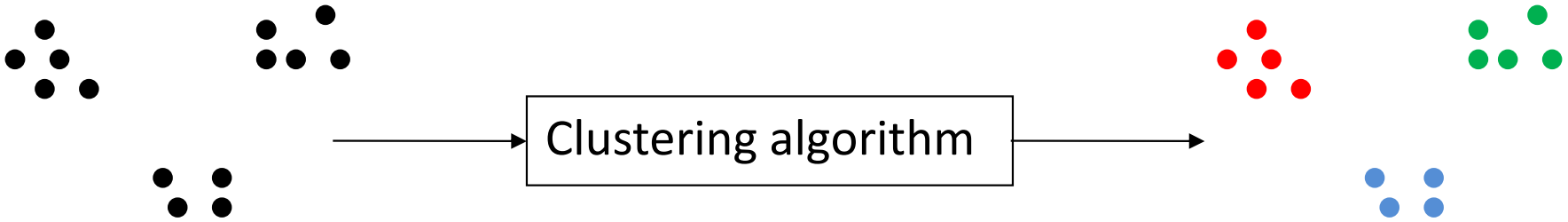
$$S = \{\ (\boldsymbol{x}(k+1), x(k+1)), (\boldsymbol{x}(k+2), x(k+2)), \dots, (\boldsymbol{x}(n), x(n))\ \}$$

Typical (unsupervised) classification system structure



Clustering algorithm: *An algorithm that groups "similar" entities to the same cluster (group) and "different" entities to different clusters.*

- Each **cluster** may be viewed as an **unlabeled class**.



The algorithm is applied a data set:

$$S = \{ \boldsymbol{x}_n, \boldsymbol{x}_n \in R^l, n = 1, \ldots, N \}$$

Contents

Part I: Introductory concepts

- **Model definition**
- Definition of ML via simplified examples
- Significant issues: *noise/uncertainty* existence, *number of data points*, *data finiteness*
- **How** to **approximate** a **true model** and relevant issues

Part II: Basic problems in ML

- Regression
- Supervised classification
- Unsupervised classification (clustering)

Part III: Some ML models in more detail

- k-nearest neighbor (*k-NN*) classifier
- *Decision trees* and *Random Forests*
- Feedforward Neural Networks (*FNNs*)
- Recurrent Neural Networks (*RNNs*) – Long-Short Term Memory (*LSTM*) networks
- Some basic concepts on *time series*

Classification – k-nearest neighbor

***k*-nearest neighbor classifier**

-It is a non-parametric classifier.

-Let

$$S = \{(y_i, \boldsymbol{x}_i), \boldsymbol{x}_i \in R^l, y_i \in \{1,2, \dots, M\}, i = 1, \dots, n\}$$

be a given data set containing vectors from M classes, $\omega_1, \dots, \omega_M$.

- **Aim:** Assign a given point $\boldsymbol{x}$ to a class, based on S.

The **classifier**:

- Choose an (odd) value for k.
- For the given $\boldsymbol{x}$ do
  - Determine its k **nearest neighbors** in the data set S.
  - Among these k points let k_j be those that stem from the j-th class, $j = 1, \dots, M$.
  - Assign $\boldsymbol{x}$ to the class ω_q, for which $k_q = max_{j=1,\dots,M} k_j$.

no training phase is encountered in k-NN.

A drawback: Increased computational complexity

($O(N)$ for the classification of a single vector)

Classification – k-nearest neighbor

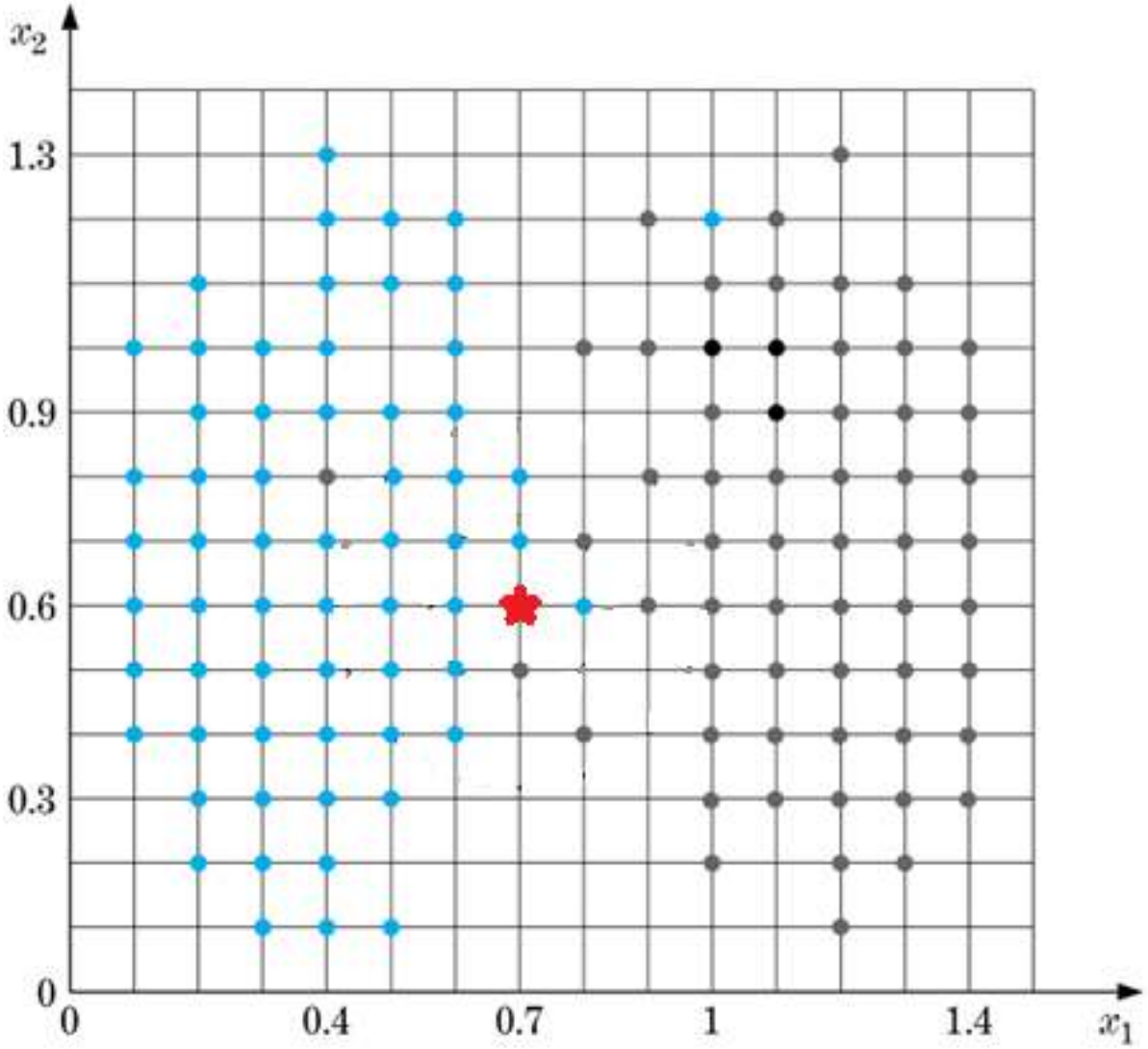
***k*-nearest neighbor classifier**

Example: Classify the "**star**" point to either the "**blue**" or the "**black**" class, according to the 7-NN classification rule.

Answer:

- Considering all the data points from **both** classes, among the ***seven*** nearest neighbors of the "**star**" point, there are $k_{blue} = 5$ data points from the **blue** class and $\boldsymbol{k_{black} = 2}$ data points from the **black** class.

- Since $k_{blue} = \max\{k_{blue}, \boldsymbol{k_{black}}\}$, the "**star**" point is assigned to the "**blue**" class.



Decision Trees

- It is a family of **classifiers**.
- They are multistage decision systems where classes are rejected in a sequential manner, until an acceptable class is reached. To this end:

– Usually, at each node a decision is taken **based** on an individual feature, asking questions of the form:

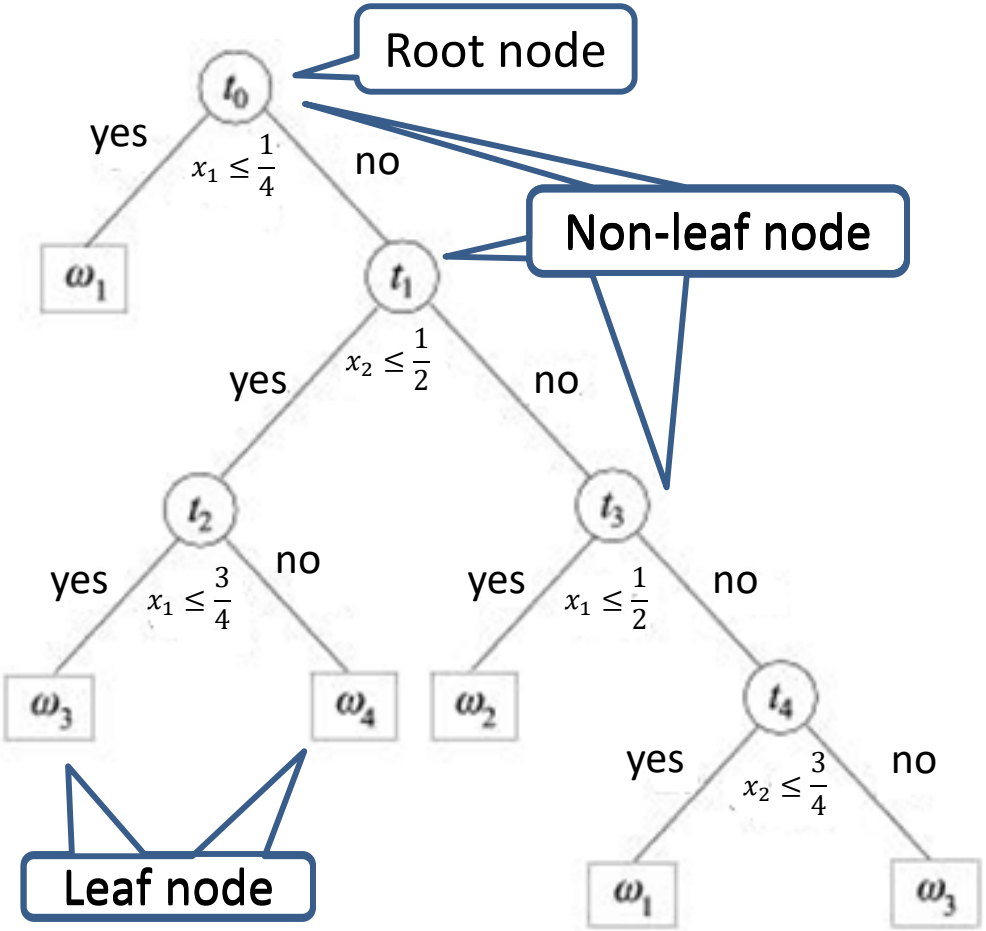
"is the value of the i-th feature less than α ($x_i \leq \alpha$)?"

where α is a preselected threshold (it is estimated during the training phase).

Decision Trees

– The following figure corresponds to such a paradigm:
  - Trees of this type (left figure) are known as Ordinary Binary Classification Trees (OBCT).

– Other types of node rules are also possible yet they are less popular.



Decision Trees

The **generation** of a **Decision Tree** (**DT**) is based on a data set

$$S = \{(y_i, \boldsymbol{x}_i), i = 1, \dots, n\} = \{(y_i, (x_{i1}, \dots, x_{il})), i = 1, \dots, n\}$$

To establish a DT we need to define:

- The **structure** of the **DT** (number of non-leaf and leaf nodes, layers of nodes)
- The **rules** for each **node**.

The purity concept:

Divide a given set S to two nonempty, disjoint subsets S_y and S_n, so that *at least one of them contains points from fewer classes than those contained in S*.

Example:

$S = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3, \boldsymbol{x}_4, \boldsymbol{x}_5, \boldsymbol{x}_6, \boldsymbol{x}_7, \boldsymbol{x}_8\}$

No gained purity

$S_y = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3, \boldsymbol{x}_4\}, S_n = \{\boldsymbol{x}_5, \boldsymbol{x}_6, \boldsymbol{x}_7, \boldsymbol{x}_8\}$

Less gained purity

$S_y = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3, \boldsymbol{x}_4, \boldsymbol{x}_5, \boldsymbol{x}_6, \boldsymbol{x}_7\}, S_n = \{\boldsymbol{x}_8\}$

More gained purity

$S_y = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_5, \boldsymbol{x}_6\}, S_n = \{\boldsymbol{x}_3, \boldsymbol{x}_4, \boldsymbol{x}_7, \boldsymbol{x}_8\}$

Decision Trees

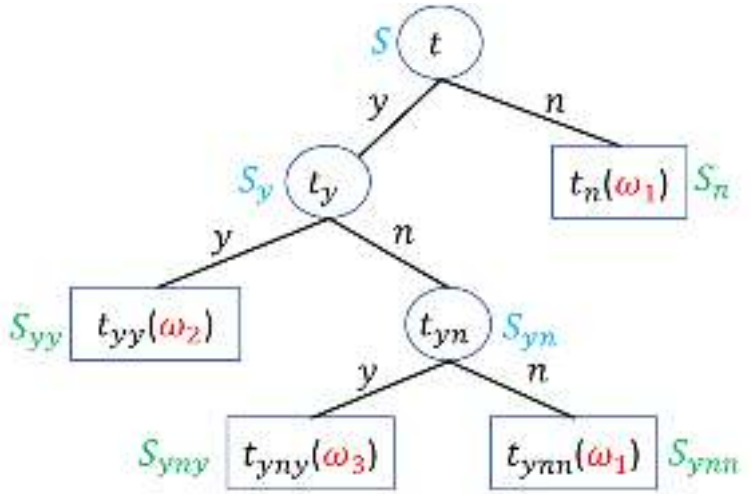
$$S = \{(y_i, \boldsymbol{x}_i), i = 1, \dots, n\} = \{(y_i, (x_{i1}, \dots, x_{il})), i = 1, \dots, n\}$$

The "building" procedure is **sequential**:

Starting point: The **rule** of the **root node**, $\boldsymbol{t}$ (associated with S):

- Consider all the following **candidate rules**

Data vector	1st coord. (x_1)	...	l^{th} coord. (x_l)
$\boldsymbol{x}_1$	$x_1 \leq x_{11}$	...	$x_l \leq x_{1l}$
$\boldsymbol{x}_2$	$x_1 \leq x_{21}$	...	$x_l \leq x_{2l}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\boldsymbol{x}_n$	$x_1 \leq x_{n1}$	...	$x_l \leq x_{nl}$



- **Associate** with the **root node** the **rule** that leads to the **maximum possible purity gain**, say $x_q \leq x_{jq}$
- **Split** S into S_y and S_n according to the rule $x_q \leq x_{jq}$
- **Define** two **children nodes** of the root, $\boldsymbol{t_y}$ and $\boldsymbol{t_n}$ associated with S_y and S_n, resp.
- **If** S_y and/or S_n are "**pure enough**" the respective **nodes** become **leaf nodes**.

Almost **all points** stem from a **single class.**

The above **procedure** is **applied recursively** until **no non-leaf nodes exist**.

Decision Trees

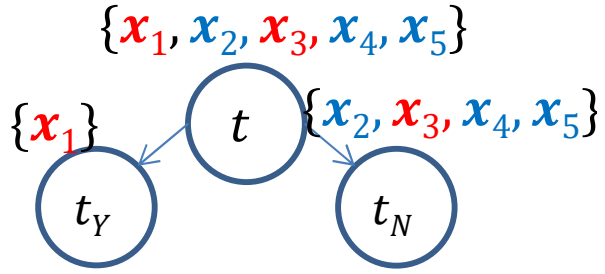
Example: Consider a two-class case with the following **5** points:

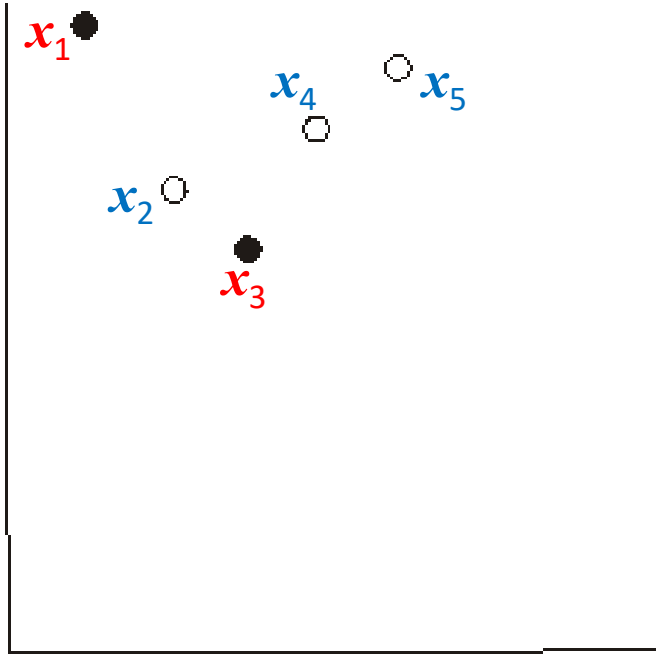
$\boldsymbol{x}_1 = (1,10) - \omega_1,\ \ \boldsymbol{x}_2 = (2,7) - \omega_2,\ \ \boldsymbol{x}_3 = (3,6)\ - \omega_1, \boldsymbol{x}_4 = (4,8) - \omega_2, \boldsymbol{x}_5 = (5,9) - \omega_2.$

The possible rules to partition the data set into two disjoint sets are

$x_1 \leq 1, x_1 \leq 2, x_1 \leq 3, x_1 \leq 4, x_1 \leq 5$

$x_2 \leq 10, x_2 \leq 7, x_2 \leq 6, x_2 \leq 8, x_2 \leq 9$





Combining classifiers: Random Forests (RF)

Random forests: The idea here is to **create** a set of B decision tree classifiers, so that they are as **independent** from each other as possible. Given a set of n l-dimensional data vectors, S, we have:

Classifiers creation phase (decision trees):

- **Generate** B data sets, $S_1, \ldots, S_B$, each one of size N by applying random sampling with replacement on S.
- Based on $S_1, \ldots, S_B$, **create** the respective decision trees, $T_1, \ldots, T_B$. For the determination of the rule of a specific node in each one of the trees, **consider** only $\lambda (<< l)$ features, **randomly selected** from the whole set of features.

Operational phase:

For a given $\boldsymbol{x}$:

- **Classify** it using $T_1, \ldots, T_B$.
- **Combine** the above results in order to produce the final classification for $\boldsymbol{x}$, e.g., by the **majority voting rule**.

Feedforward Neural networks (FNNs)

-They are parametric models.

-Their structuring element (**perceptron** structure) is the **neuron** (**node**), which:

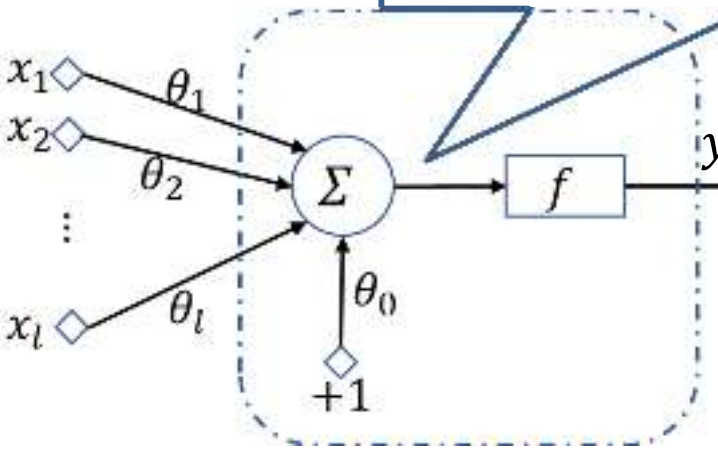
(a) **weights** its inputs,

(b) it **sums** them and

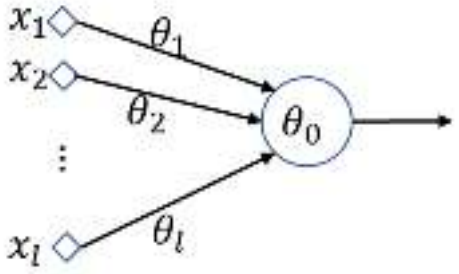
(c) it **passes** the result through a nonlinear function $f(\cdot)$ (output function).

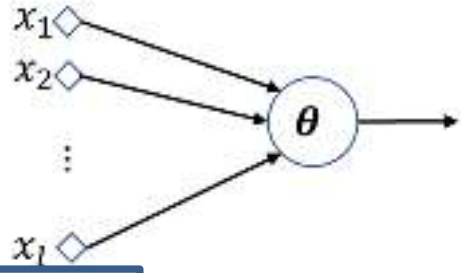
The perceptron structure:

$$\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_l x_l \equiv \boldsymbol{\theta}^T \boldsymbol{x},$$
$$\boldsymbol{x} = [1, x_1, x_2, \ldots, x_l]^T,\ \boldsymbol{\theta} = [\theta_0, \theta_1, \theta_2, \ldots, \theta_l]^T$$



$$y = f(\boldsymbol{\theta}^T \boldsymbol{x}) = f\left(\sum_{i=0}^{l} \theta_i x_i\right)$$



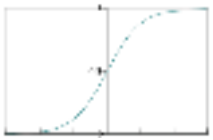


Typical choices for $f(\cdot)$

(a) Sigmoid $f(z) = \frac{1}{1+e^{-az}}$ or $f(z) = \frac{1-e^{-az}}{1+e^{-az}}$,

(b) ReLU $f(z) = \begin{cases} z, & if\ z \geq 0 \\ 0, & if\ z < 0 \end{cases}$

(c) Linear $f(z) = z$

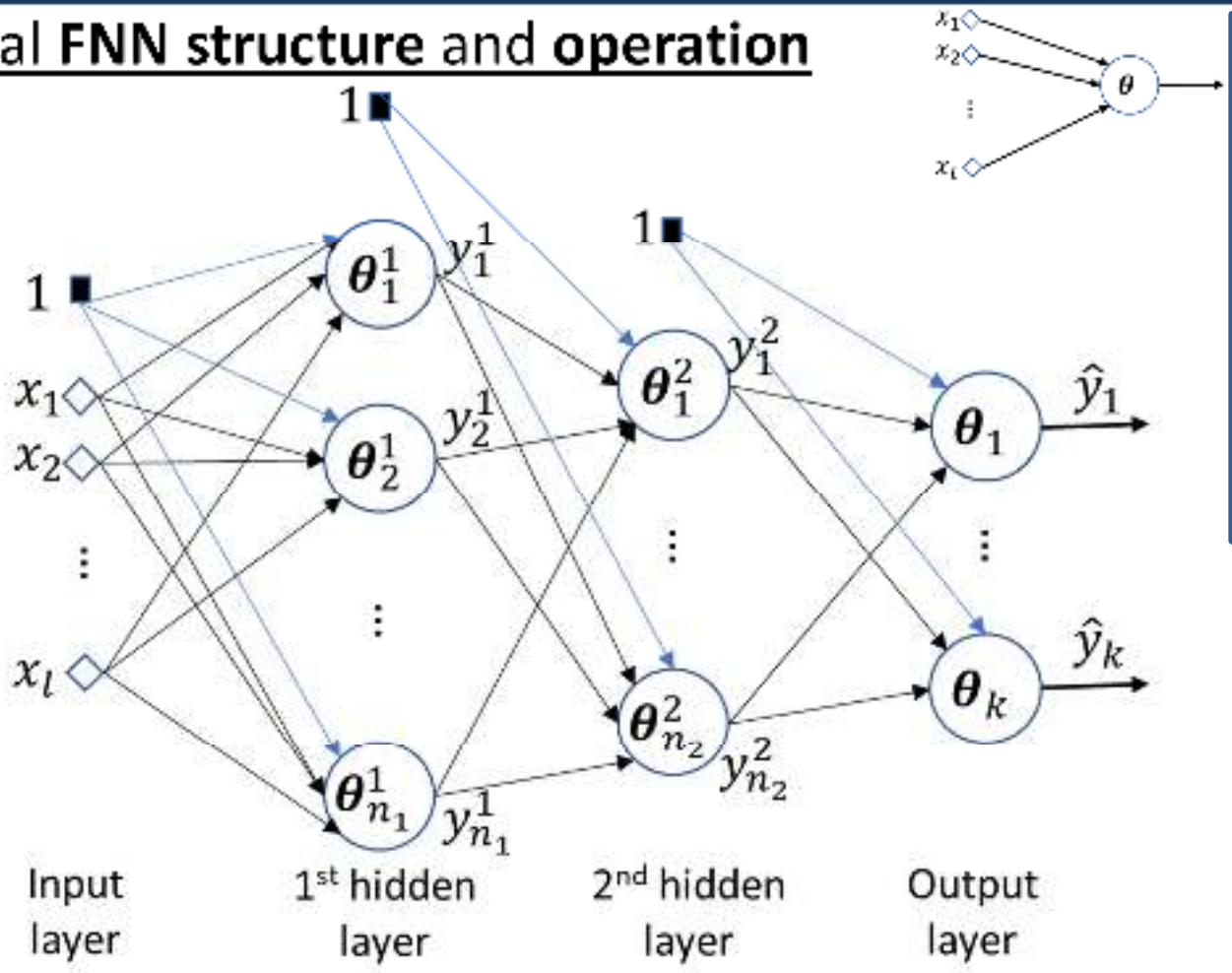






Feedforward Neural networks (FNNs)

Typical **FNN structure** and **operation**



Parameter vector of the i-th node in the j-th layer

$$\boldsymbol{\theta}_i^j = \left[\theta_{0i}^j, \theta_{1i}^j, \theta_{2i}^j, \dots, \theta_{n_{i-1},i}^j\right]^T$$

Layer parameter matrices

$$\Theta^1 = \left[\boldsymbol{\theta}_1^1, \boldsymbol{\theta}_2^1, \dots, \boldsymbol{\theta}_{n_1}^1\right],$$

$$\Theta^2 = \left[\boldsymbol{\theta}_1^2, \boldsymbol{\theta}_2^2, \dots, \boldsymbol{\theta}_{n_2}^2\right]$$

$$\Theta = \left[\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \dots, \boldsymbol{\theta}_k\right]$$

They can be **used** both in **regression** and **classification** tasks.

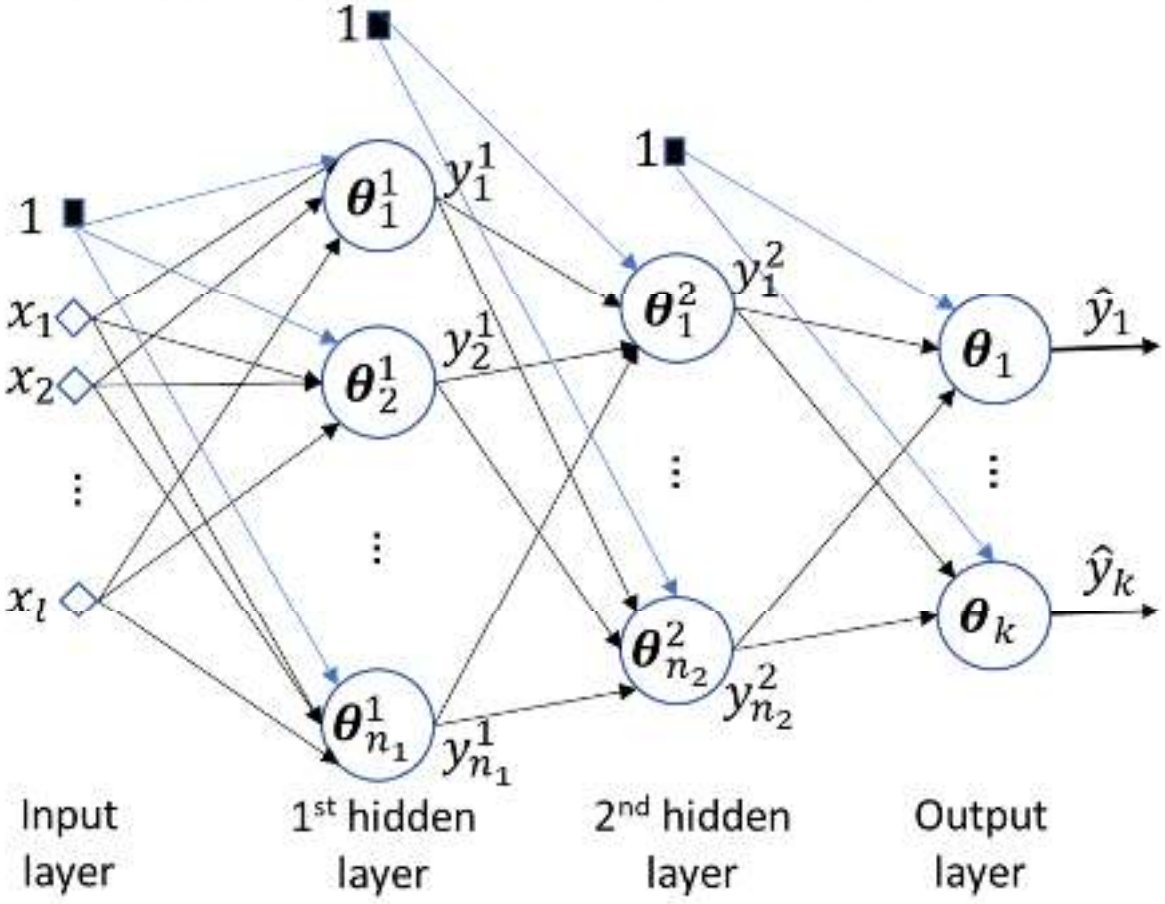
E.g., for a **3-class** classification task, we have **3 output nodes** and the classes are coded as [0,0,1], [0,1,0], [1,0,0]

$$\boldsymbol{x} \to \boldsymbol{y}^1 = \begin{bmatrix} f\left(\boldsymbol{\theta}_1^{1\,T}\boldsymbol{x}\right) \\ f\left(\boldsymbol{\theta}_2^{1\,T}\boldsymbol{x}\right) \\ \vdots \\ f\left(\boldsymbol{\theta}_{n_1}^{1\,T}\boldsymbol{x}\right) \end{bmatrix} \to \boldsymbol{y}^2 = \begin{bmatrix} f\left(\boldsymbol{\theta}_1^{2\,T}\boldsymbol{y}^1\right) \\ f\left(\boldsymbol{\theta}_2^{2\,T}\boldsymbol{y}^1\right) \\ \vdots \\ f\left(\boldsymbol{\theta}_{n_2}^{2\,T}\boldsymbol{y}^1\right) \end{bmatrix} \to \hat{\boldsymbol{y}} = \begin{bmatrix} f(\boldsymbol{\theta}_1^T\boldsymbol{y}^2) \\ f(\boldsymbol{\theta}_2^T\boldsymbol{y}^2) \\ \vdots \\ f(\boldsymbol{\theta}_k^T\boldsymbol{y}^2) \end{bmatrix}$$

$$\boldsymbol{x} \to \boldsymbol{y}^1 = f\left(\Theta^{1\,T}\boldsymbol{x}\right) \to \boldsymbol{y}^2 = f\left(\Theta^{2\,T}\boldsymbol{y}^1\right) \to \hat{\boldsymbol{y}} = f(\Theta^T\boldsymbol{y}^2)$$

Feedforward Neural networks (FNNs)

FNN training (parameter estimation)



$$\boldsymbol{x} \rightarrow \boldsymbol{y}^1 = f\left(\Theta^{1^T}\boldsymbol{x}\right) \rightarrow \boldsymbol{y}^2 = f\left(\Theta^{2^T}\boldsymbol{y}^1\right) \rightarrow \hat{\boldsymbol{y}} = f(\Theta^T\boldsymbol{y}^2)$$

- $\Theta' = \{\Theta^1, \Theta^2, \dots, \Theta\}$
- $S = \{(\boldsymbol{y}_i, \boldsymbol{x}_i), \boldsymbol{x}_i \in R^l, \boldsymbol{y}_i \in R^k, i = 1, \dots, n\}$
- **Cost function** $J(\Theta') = \sum_{i=1}^{n} L(\boldsymbol{y}_i, \hat{\boldsymbol{y}}_i)$
  (e.g., $L(\boldsymbol{y}, \hat{\boldsymbol{y}}) = \sum_{j=1}^{k}\left(y_j - \hat{y}_j\right)^2$)

Aim: $minimize_{\Theta'} J(\Theta')$

It is carried out **iteratively** according to the **BackPropagation** (**BP**) philosophy:

- **Initialization** of Θ'
- At *each iteration*
  - **Forward pass:** the FNN outputs are computed for all $\boldsymbol{x}_i$'s
  - **Backward pass:** the parameter updating quantities are **computed** for the nodes in the **output layer**, then for the **last hidden layer**,…, then for the **1st hidden layer**.
  - **Then**, the **parameter updating** is **performed**.
- Until *convergence*

Recurrent Neural networks (RNNs)

- In some applications, **sequential data dependence** is **inherited** (e.g., time series forecasting)

- **Sequential data dependence (SDD):** The **ordering** of **data presentation** $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_{i-1}, \boldsymbol{x}_i, \boldsymbol{x}_{i+1}, \ldots, \boldsymbol{x}_n$ in a system **matters**. For time series applications, the **indices** have **temporal content** (time instances).

- Although **FNNs** can be used for this kind of data, they **cannot exploit** the **SDD** knowledge.

- **Recurrent Neural Networks** (**RNNs**) have the ability to **exploit** the SDD knowledge.

Two such RNNs are:

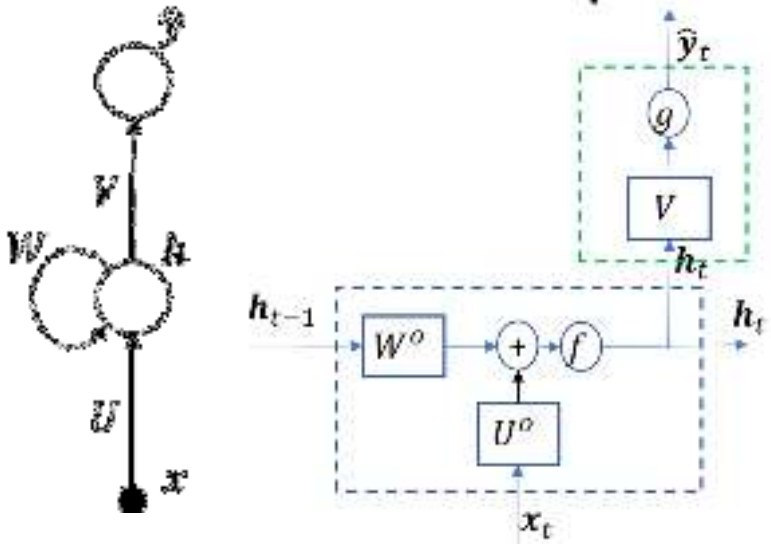
- (**Vanilla**) **RNN**
- **Long-Short Term Memory** (**LSTM**) networks

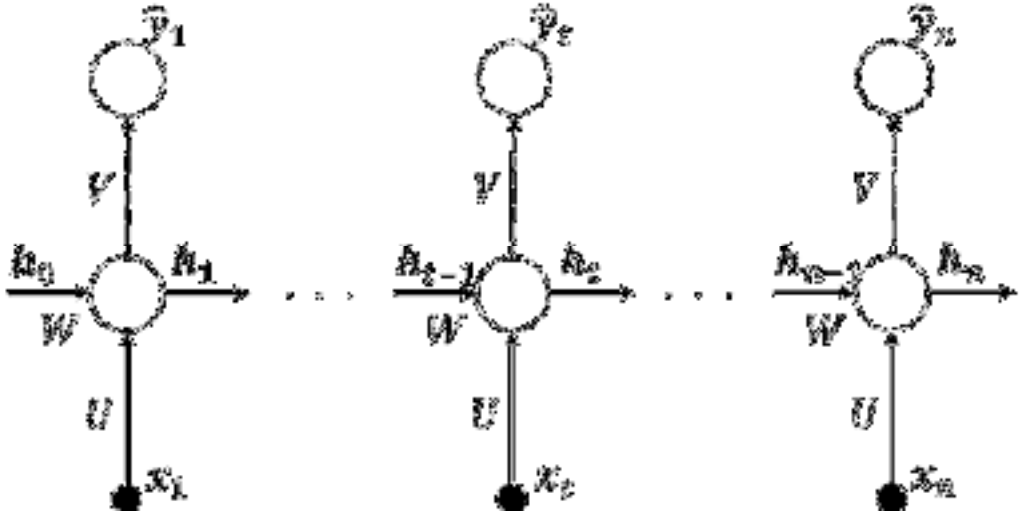
(Vanilla) Recurrent Neural networks (RNNs)

- **Sequential data dependence (SDD):** The **ordering** of **data presentation** $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_{i-1}, \boldsymbol{x}_i, \boldsymbol{x}_{i+1}, \ldots, \boldsymbol{x}_n$ in a system **matters**.

- ***RNN state***: The **RNN state** at time t "**aggregates**" the **RNN history** up to this time. It is modeled by the state vector $\boldsymbol{h}_t$.

- **Important note:** In **SDD** cases, the **network output** at time t should not be considered independently from the outputs of the other time instances. One should consider **the outputs for all consecutive time instances** as a whole.





$$\boldsymbol{h}_t = f(U\boldsymbol{x}_t + W\boldsymbol{h}_{t-1} + \boldsymbol{b})$$

$$\hat{\boldsymbol{y}}_t = g(V\boldsymbol{h}_t + \boldsymbol{c})$$

Temporal "unfolding"

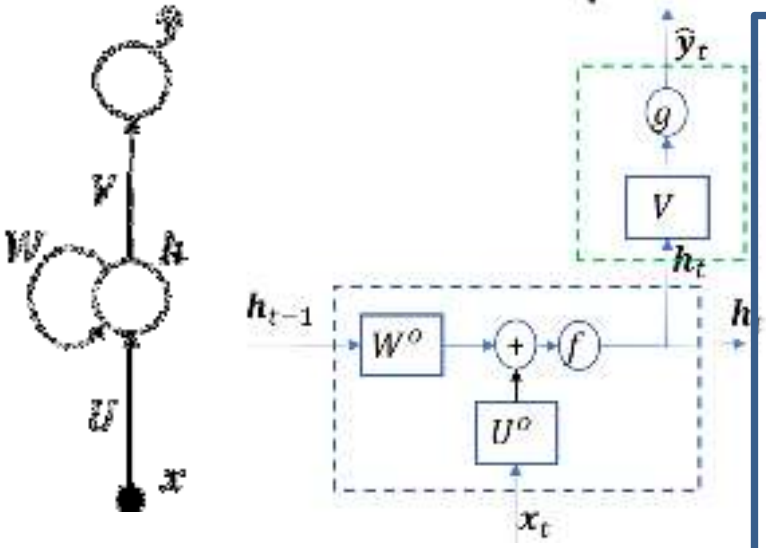
Parameters: $U, W, \boldsymbol{b}, V, \boldsymbol{c}$

(Vanilla) Recurrent Neural networks (RNNs)

- **Sequential data dependence (SDD):** The **ordering** of **data presentation** $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_{i-1}, \boldsymbol{x}_i, \boldsymbol{x}_{i+1}, \ldots, \boldsymbol{x}_n$ in a system **matters**.

- ***RNN state***: The **RNN state** at time t "**aggregates**" the **RNN history** up to this time. It is modeled by the state vector $\boldsymbol{h}_t$.

- **Important note:** In **SDD** cases, the **network output** at time t should not be considered independently from the outputs of the other time instances. One should consider **the outputs for all consecutive time instances** as a whole.



$$\boldsymbol{h}_t = f(U\boldsymbol{x}_t + W\boldsymbol{h}_{t-1} + \boldsymbol{b})$$

$$\hat{\boldsymbol{y}}_t = g(V\boldsymbol{h}_t + \boldsymbol{c})$$

Parameters: $U, W, \boldsymbol{b}, V, \boldsymbol{c}$

Parameter estimation:

- $S = \{(\boldsymbol{y}_t, \boldsymbol{x}_t), t = 1, \ldots, n\}$
- **Cost function**

$$J(U, W, V, \boldsymbol{b}, \boldsymbol{c}) = \sum_{t=1}^{n} L(\boldsymbol{y}_t, \hat{\boldsymbol{y}}_t)$$

(e.g., $L(\boldsymbol{y}_t, \hat{\boldsymbol{y}}_t) = \|\boldsymbol{y}_t - \hat{\boldsymbol{y}}_t\|_2^2$

Aim: $minimize_{U,W,V,\boldsymbol{b},\boldsymbol{c}} J(U, W, V, \boldsymbol{b}, \boldsymbol{c})$

It is carried out **iteratively** according to the **Back-Propagation Through Time** (**BPTT**): It follows the BP philosophy but, additionally, it can handle SDD cases.

Long-Short Term Memory (LSTM) networks

- **Drawback of vanilla RNNs:** They **cannot deal** well with **long-term** data **dependencies**.

- **LSTM** networks deal with such situations.

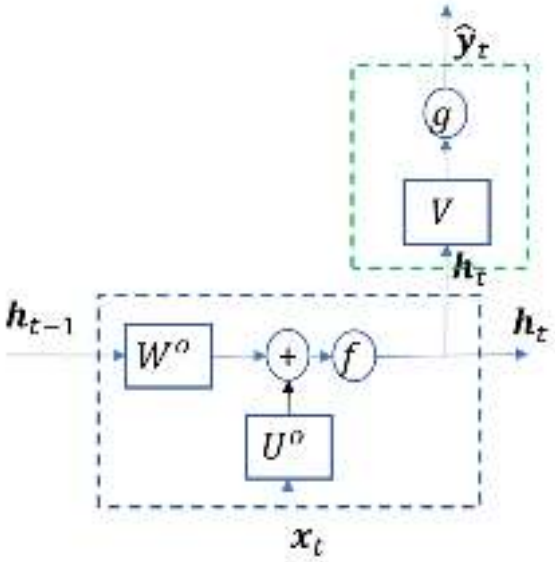
- **LSTM** networks are **RNNs** in their nature.

- They use two (instead of one in vanilla RNNs) different aspects:
  - The **cell**/unit **state** (modeled by the vector $\boldsymbol{s}$)
  - The **hidden variables** concept (modeled by the vector $\boldsymbol{h}$)

- The **cell state vector** $\boldsymbol{s}$ handles **long-term dependencies** (so that to *forget less valuable* and *retain more valuable* information from the past).

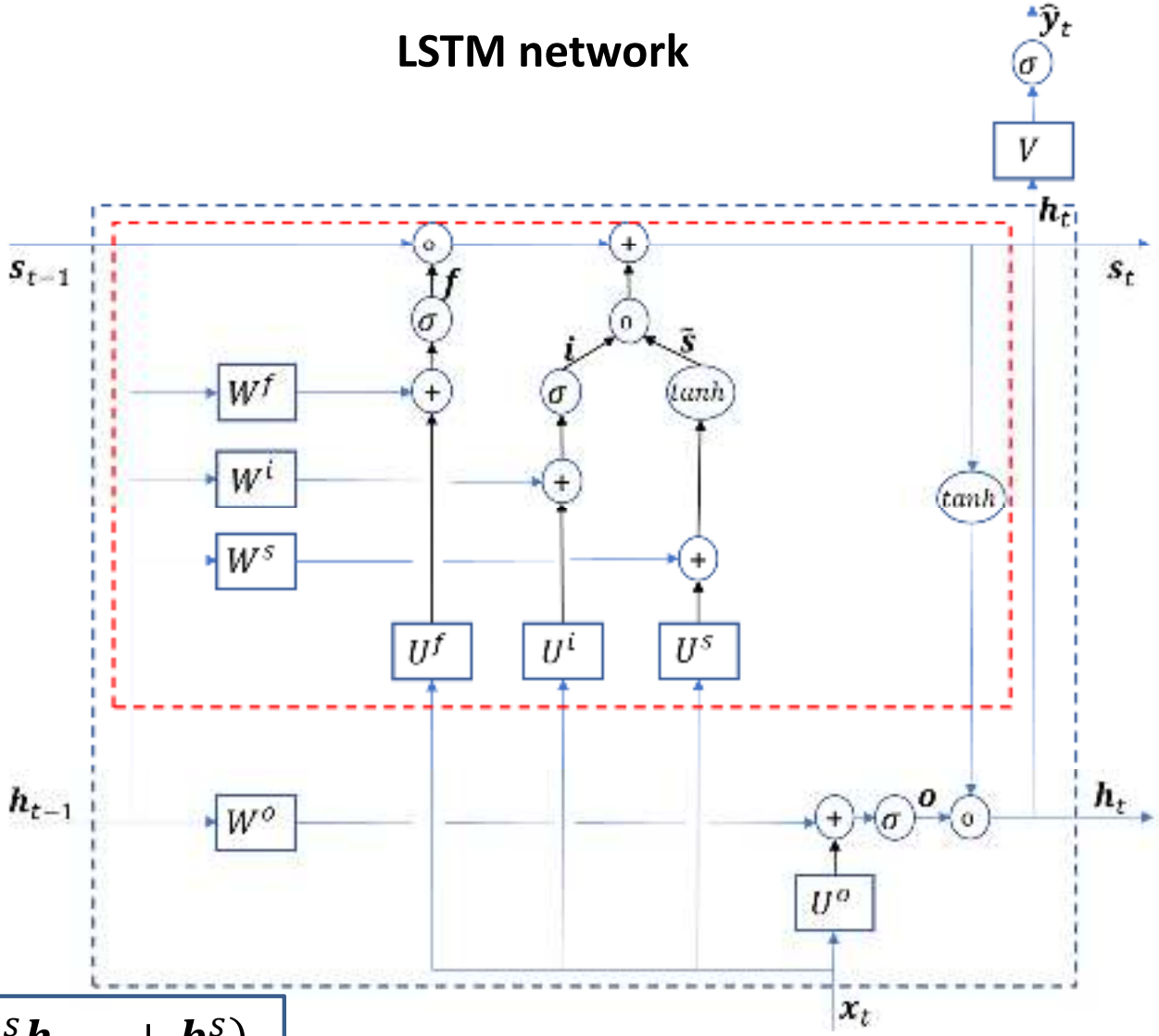
Long-Short Term Memory (LSTM) networks

Vanilla RNN



$$\boldsymbol{h}_t = f(U\boldsymbol{x}_t + W\boldsymbol{h}_{t-1} + \boldsymbol{b})$$
$$\hat{\boldsymbol{y}}_t = g(V\boldsymbol{h}_t + \boldsymbol{c})$$

LSTM network



$$\boldsymbol{x}_t$$
$$\boldsymbol{h}_{t-1}$$

$$\tilde{\boldsymbol{s}} = tanh(U^s\boldsymbol{x}_t + W^s\boldsymbol{h}_{t-1} + \boldsymbol{b}^s)$$
$$\boldsymbol{f} = \sigma\left(U^f\boldsymbol{x}_t + W^f\boldsymbol{h}_{t-1} + \boldsymbol{b}^f\right)$$
$$\boldsymbol{i} = \sigma\left(U^i\boldsymbol{x}_t + W^i\boldsymbol{h}_{t-1} + \boldsymbol{b}^i\right)$$

$$\boldsymbol{s}_t = \boldsymbol{f}\ o\ \boldsymbol{s}_{t-1} + \boldsymbol{i}\ o\ \tilde{\boldsymbol{s}}$$

$$\boldsymbol{o} = \sigma(U^o\boldsymbol{x}_t + W^o\boldsymbol{h}_{t-1} + \boldsymbol{b}^o)$$

$$\boldsymbol{h}_t = \boldsymbol{o}\ o\ tanh(\boldsymbol{s}_t)$$

$$\hat{\boldsymbol{y}}_t = g(V\boldsymbol{h}_t + \boldsymbol{c})$$

Long-Short Term Memory (LSTM) networks

LSTM operation:

NOTE: All subsequent **operations** are in an **elementwise manner.**

(A) Update of s_t

•The cell state vector s_t is a **combination** of s_{t-1} and the **new information** $\tilde{s}$:

$$s_t = f \; o \; s_{t-1} + i \; o \; \tilde{s}$$

• All f, i, $\tilde{s}$ are **determined** by the **current input** x_t and the previous **hidden variable** value, h_{t-1}

$$\tilde{s} = tanh(U^s x_t + W^s h_{t-1} + b^s)$$
$$f = \sigma\left(U^f x_t + W^f h_{t-1} + b^f\right)$$
$$i = \sigma\left(U^i x_t + W^i h_{t-1} + b^i\right)$$

$$\sigma(z) = \frac{1}{1+e^{-z}}$$

(B) Update of h_t

• The **hidden variable vector depends** on the **state vector** s_t:

$$h_t = o \; o \; tanh(s_t)$$

•o is **determined** by the **current input** x_t and the previous **hidden variable** value, h_{t-1}

$$o = \sigma(U^o x_t + W^o h_{t-1} + b^o)$$

(C) Computation of $\hat{y}_t$

•It depends exclusively on h_t

$$\hat{y}_t = g(V h_t + c)$$

Long-Short Term Memory (LSTM) networks

LSTM parameter estimation:

- $S = \{(\boldsymbol{y}_t, \boldsymbol{x}_t), t = 1, \dots, n\}$
- **Cost function**

  $J(U^s, U^f, U^i, U^o, W^s, W^f, W^i, W^o, \boldsymbol{b}^s, \boldsymbol{b}^f, \boldsymbol{b}^i, \boldsymbol{b}^o) = \sum_{t=1}^{n} L(\boldsymbol{y}_t, \hat{\boldsymbol{y}}_t)$

  (e.g., $L(\boldsymbol{y}_t, \hat{\boldsymbol{y}}_t) = \|\boldsymbol{y}_t - \hat{\boldsymbol{y}}_t\|_2^2$)

Aim:

$$min_{U^s, U^f, U^i, U^o, W^s, W^f, W^i, W^o, \boldsymbol{b}^s, \boldsymbol{b}^f, \boldsymbol{b}^i, \boldsymbol{b}^o} J(U^s, U^f, U^i, U^o, W^s, W^f, W^i, W^o, \boldsymbol{b}^s, \boldsymbol{b}^f, \boldsymbol{b}^i, \boldsymbol{b}^o)$$

It is carried out **iteratively** according to the **Back-Propagation Through Time** (**BPTT**): It follows the BP philosophy but, additionally, it can handle SDD cases.

A short time-series introduction

Time series process: A **sequence** of n **temporarily ordered** random variables (rvs) that are **related** to **each other**.

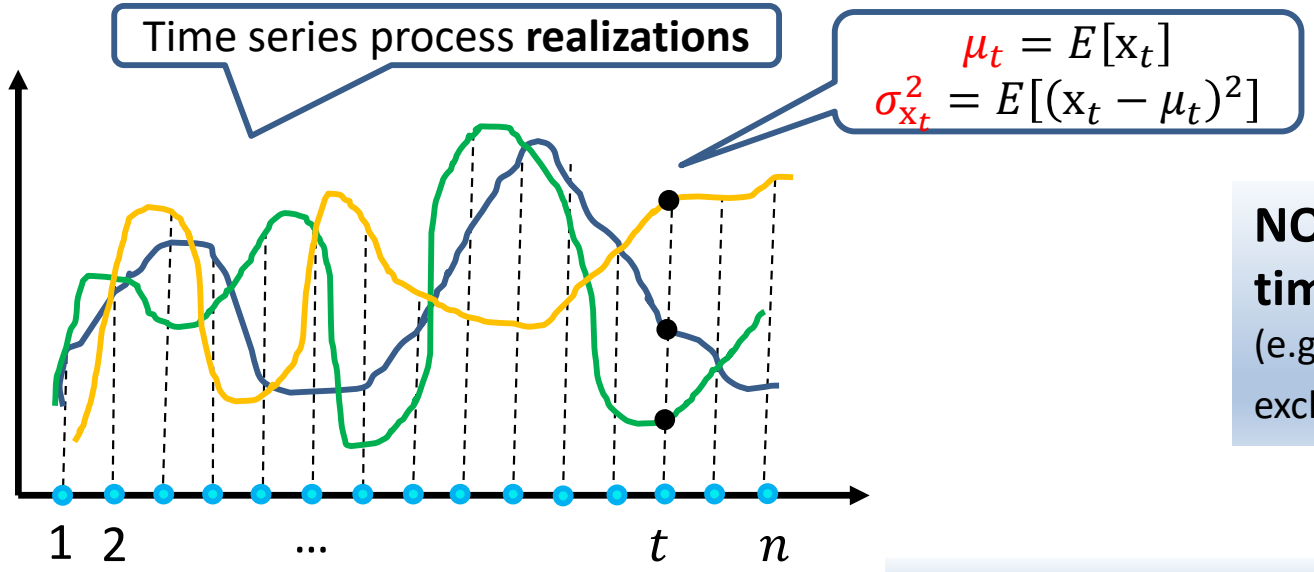
$$\{\mathrm{x}_t\}_{t=1}^{n} \equiv \mathrm{x}_{t=1}^{n} = \{\mathrm{x}_1, \dots, \mathrm{x}_t, \dots, \mathrm{x}_n\}$$

the **size** of the **time interval** between **two successive time stamps** is considered **fixed.**

x_t: the **rv** associated with the ***t*-th** time stamp.

$\mathrm{x}_{t=1}^{n}$ is **modeled** by an n-dimensional pdf $p(x_1, x_2, \dots, x_n)$.

Picking **randomly** a sequence (time series), $x_{t=1}^{n} = (x_1, x_2, \dots, x_n)$ **according** to $p(x_1, x_2, \dots, x_n)$, we have a *time series realization* of $\mathrm{x}_{t=1}^{n}$.



NOTE: In **practice**, only a single **time series realization** is **available** (e.g., the daily time series of the currency exchange US dollar/Euro, from 2020 to 2025).

ISSUE: How to perform future time-series estimation

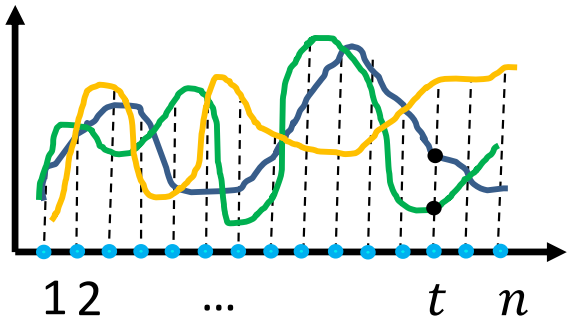
A short time-series introduction

Time series process quantitative description ($\{x_t\}_{t=1}^{n} \equiv x_{t=1}^{n} = \{x_1, \dots, x_t, \dots, x_n\}$)

A. *For individual time stamps*

Mean per time stamp: $\mu_t = E[x_t]$

Variance per time stamp: $var(x_t) = \sigma_{x_t}^2 = E[(x_t - \mu_t)^2]$



B. *For pairs of time stamps*

Autocovariance function: $\gamma(s,t) = cov(x_s, x_t) = E[(x_s - \mu_s)(x_t - \mu_t)]$

Autocorrelation function: $\rho(s,t) = \frac{\gamma(s,t)}{\sqrt{\gamma(s,s)\gamma(t,t)}}$

Partial autocorrelation function (PAC) of lag k: the **correlation** between x_t and x_{t+k}, with the **linear dependence** of x_{t+k} on $x_{t+k-1}, \dots, x_{t+1}$ being **removed**.

They measure how x_s and x_t **covariate**.

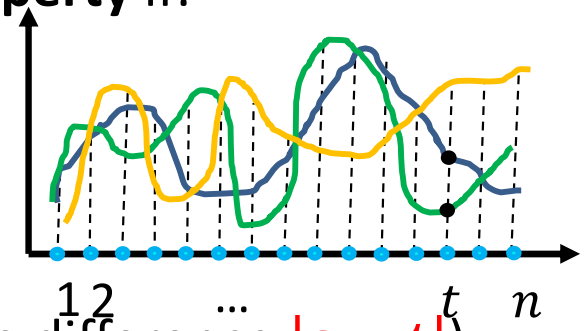
A major practical issue:

Since in **practice**, only a single **time series realization** is **available**, the above **quantities cannot be estimated reliably** (both per time stamp and per pairs of time stamps).

Way out: Focus on a **special class** of **time series processes**; Those that posses the (weak) stationarity (WS) property.

A short time-series introduction: stationary processes

A time series process $\mathrm{x}_{t=1}^{n}$ **possesses** the **weak stationarity property** if:



1. $E[\mathrm{x}_t] = \mu, t = 1, \ldots, n$

(the rvs x_t at all time stamps have **common mean value** μ)

2. $\gamma(s,t) = f(|s-t|)$

(the autocovariance function $\gamma(s,t)$ **depends exclusively** on the difference $|s-t|$)

Setting $h = |s-t|$, we write

- $\gamma(s,t) = \gamma(h)$
- $\rho(s,t) = \rho(h) = \frac{\gamma(h)}{\gamma(0)}$

- For a **WS time series process**, we can **estimate** the mean and covariance quantities from just **one realization**.

For the time series $x_{t=1}^{n}$ (a realization of the WS $\mathrm{x}_{t=1}^{n}$), it is

- $\hat{\mu} = \frac{1}{n}\sum_{t=1}^{n} x_t$, $\quad \hat{\sigma}^2 = \frac{1}{n}\sum_{t=1}^{n}(x_t - \hat{\mu})^2$
- $\hat{\gamma}(h) = \frac{1}{n-h}\sum_{t=1}^{n-h}(x_t - \hat{\mu})(x_{t+h} - \hat{\mu})$, $\quad \hat{\rho}(h) = \frac{\hat{\gamma}(h)}{\hat{\gamma}(0)}$

A short time-series introduction: stationary processes

White noise ($x_{t=1}^{n}$)**:** the **simplest** WS time series process.

- the rvs $x_1, x_2, \ldots, x_n$ are **independent** from each other and are **identically distributed** ($p(x_1, x_2, \ldots, x_n) = p(x_1)p(x_2) \ldots p(x_n)$).
- $p(x_i) = wn(0, \sigma_w^2)$ (zero mean with variance σ_w^2)
- $\gamma_w(h) = cov(w_{t+h}, w_t) = \begin{cases} \sigma_w^2, & h = 0 \\ 0, & h \neq 0 \end{cases}$
- $\rho_w(h) = \begin{cases} 1, & h = 0 \\ 0, & h \neq 0 \end{cases}$

Autoregressive process $AR(p)$: A **stationary** time series process x_{t-1}^{n} is called ***autoregressive*** of order p, $AR(p)$, if it is

$$x_t = a + \varphi_1 x_{t-1} + \varphi_2 x_{t-2} + \cdots + \varphi_p x_{t-p} + w_t$$

with

(a) $w_t \sim wn(0, \sigma_w^2)$ and

(b) $\varphi_1, \varphi_2, \ldots, \varphi_p$ being **constant parameters** ($\varphi_p \neq 0$)

($a = \mu(1 - \varphi_1 - \cdots - \varphi_p)$)

In words: x_t **depends linearly** on $x_{t-1}, x_{t-2}, \ldots, x_{t-p}$ **+** a randomly generated quantity modelled by the white noise process, w_t.

A short time-series introduction: stationary processes

Moving average process $MA(q)$**:** A ***moving average*** time series process x_{t-1}^{n} of order q, $MA(q)$, is defined as

$$x_t = w_t + \theta_1 w_{t-1} + \theta_2 w_{t-2} + \cdots + \theta_q w_{t-q}$$

with $w_t \sim wn(0, \sigma_w^2)$ and $\theta_1, \theta_2, \ldots, \theta_q$ being **constant parameters** ($\theta_q \neq 0$).

In words: **Linear combinations** of the **white noise** process **generate** $x_{t=1}^{n}$.

Autoregressive moving average process $ARMA(p, q)$**:** A **stationary** time series process $x_{t=1}^{n}$ is called ***autoregressive moving average*** $ARMA(p, q)$, if it holds

$$x_t = a + \varphi_1 x_{t-1} + \cdots + \varphi_p x_{t-p} + w_t + \theta_1 w_{t-1} + \cdots + \theta_q w_{t-q}$$

with

(a) $w_t \sim wn(0, \sigma_w^2)$

(b) $\varphi_1, \varphi_2, \ldots, \varphi_p, \theta_1, \theta_2, \ldots, \theta_q$ being **constant parameters** ($\varphi_p \neq 0, \theta_q \neq 0$),

(c) p and q being called autoregressive and moving average **orders**

($a = \mu(1 - \varphi_1 - \cdots - \varphi_p)$)

It is: $ARMA(p, 0) \rightarrow AR(p)$ and $ARMA(0, q) \rightarrow MA(q)$

A short time-series introduction: stationary processes

The above refer to stationary processes.
What if the process under consideration is not stationary?

Fortunately, in some cases, a **nonstationary** TS can be **transformed** to a **stationary**.

Some definitions:

(1st order) Differencing operator of $x_{t=1}^{n}$: $\nabla x_t = x_t - x_{t-1}$

2nd order Differencing operator of $x_{t=1}^{n}$: $\nabla^2 x_t = \nabla x_t - \nabla x_{t-1} (= x_t - 2x_{t-1} + x_{t-2})$.

k-th order Differencing operator of $x_{t=1}^{n}$: $\nabla^k x_t = \nabla^{k-1} x_t - \nabla^{k-1} x_{t-1}$

Some results:

- If $x_{t=1}^{n}$ is **stationary**, then $\nabla x_{t=1}^{n}$ is also **stationary**.

- If $x_{t=1}^{n}$ and $y_{t=1}^{n}$ are **uncorrelated stationary** time series processes, then $(x + y)_{t=1}^{n}$ is also a **stationary** time series process.

A short time-series introduction: stationary processes

In several situations, for a time series process $x_{t=1}^{n}$ we have

$$x_t = z_t + y_t$$

with

$z_{t=1}^{n}$ a **nonstationary trend** component and

$y_{t=1}^{n}$ a **zero mean stationary** component.

Some special cases for $z_{t=1}^{n}$:

(a) $z_t = b_0 + b_1 t$ (**nonstationary**):

- x_t is **nonstationary**.
- $\nabla x_t = x_t - x_{t-1} = (z_t + y_t) - (z_{t-1} + y_{t-1}) = (z_t - z_{t-1}) + (y_t - y_{t-1}) = b_1 + \nabla y_t$ is **stationary.**

(b) $z_t = \sum_{j=0}^{r} b_j t^j$ $z_t = b_0 + b_1 t + \cdots + b_r t^r$ (**nonstationary**):

- x_t is **nonstationary**.
- $\nabla^r x_t$ is **stationary.**

(c) $z_t = z_{t-1} + v_t$: z_t is a slowly **varying non-stationary** stochastic process (v_t is a stationary process)

- x_t is **nonstationary**.
- $\nabla x_t = x_t - x_{t-1} = v_t + \nabla y_t$ is **stationary.**

A short time-series introduction: stationary processes

ARIMA processes: They are **not stationary** but **they may become** through the application of the difference operation (one or more times).

AutoRegressive Integrated Moving Average process $ARIMA(p,d,q)$: A time series process $\mathrm{x}_{t=1}^{n}$ is called $ARIMA(p,d,q)$, if the process $\nabla^d \mathrm{x}_{t=1}^{n}$ is $ARMA(p,q)$.

In practice, we have at our disposal a **single** time series $x_{t=1}^{n}$.
Issues:

- How can we **model** $x_{t=1}^{n}$?
- How can we make future **predictions**?

A short time-series introduction: stationary processes

A working plan:

Stage I:

- **Check** $x_{t=1}^{n}$ for **stationarity** (using statistical hypothesis testing, e.g., ADF)
- **If** the test suggests **stationarity** move to **Stage II**.
- **Otherwise**, apply differencing operator on $x_{t=1}^{n}$ and **check** $\nabla x_{t=1}^{n}$ for **stationarity.**
- **Repeat** the **previous step** until the **stationarity** hypothesis is **not rejected.**

Stage II:

Determine the p, q parameters of $ARMA(p, q)$ model that "**best fits**" $x_{t=1}^{n}$ (or a $\nabla^d x_t$):

First path:

- **Adopt** a criterion Q that **compromises** between **(a)** *model simplicity* and **(b)** *degree of fitness* of the model to $x_{t=1}^{n}$ (or $\nabla^d x_t$).
- **Evaluate** Q for a **range** of (p, q) values and **select** the $ARMA(p, q)$ model that corresponds to the **optimum** value of Q (Q may be e.g., AIC or BIC).

Second path:

- Other methods are based on the **autocorrelation** and the **partial autocorrelation** functions.

Stage III:

Estimate φ_i, $i = 1, \dots, p$, and θ_j, $j = 1, \dots, q$, of the selected $ARMA$ model (e.g., using the Durbin-Levinson algorithm)

Stage IV:

Perform *future predictions*, utilizing the established *ARMA* model.

Some practical guidelines

Classification task:

"Static" regression tasks: No temporal dependencies are present
Working Steps:
Determine the exact input and output information that describe the task under consideration.
Aggregate all input variables to form the l-dim. input vector ($\boldsymbol{x}$)
Aggregate all output variables to form the k-dim. output vector ($\boldsymbol{y}$)

Data set creation: For each "piece of data", measure the input vector value ($\boldsymbol{x}_i$) and the associated output vector value ($\boldsymbol{y}_i$).

Adopt a model that maps l-dim. input space to the k-dim. output space.