CIFAR
CANADIAN INSTITUTE FOR ADVANCED RESEARCH

CRM

FIELDS

Deep Learning Summer School 2015

Introduction to Machine Learning

by Pascal Vincent

MILA Montreal Institute for Learning Algorithms

August 4, 2015

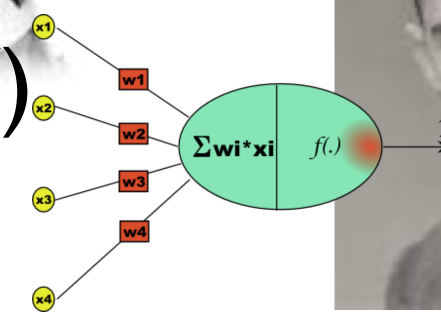
Université de Montréal

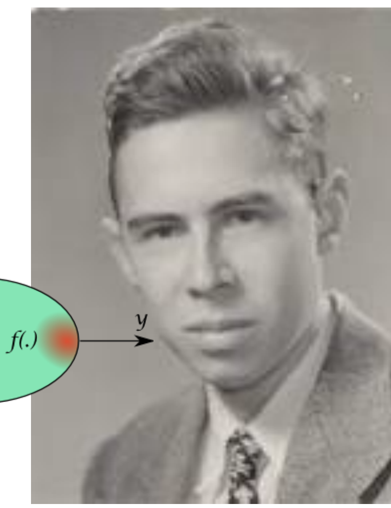
Département d'informatique et de recherche opérationnelle

What is machine learning ?

Historical perspective

- Born from the ambitious goal of **Artificial Intelligence**
- Founding project:
  The **Perceptron** (Frank Rosenblatt 1957)
  First artificial neuron **learning** form examples





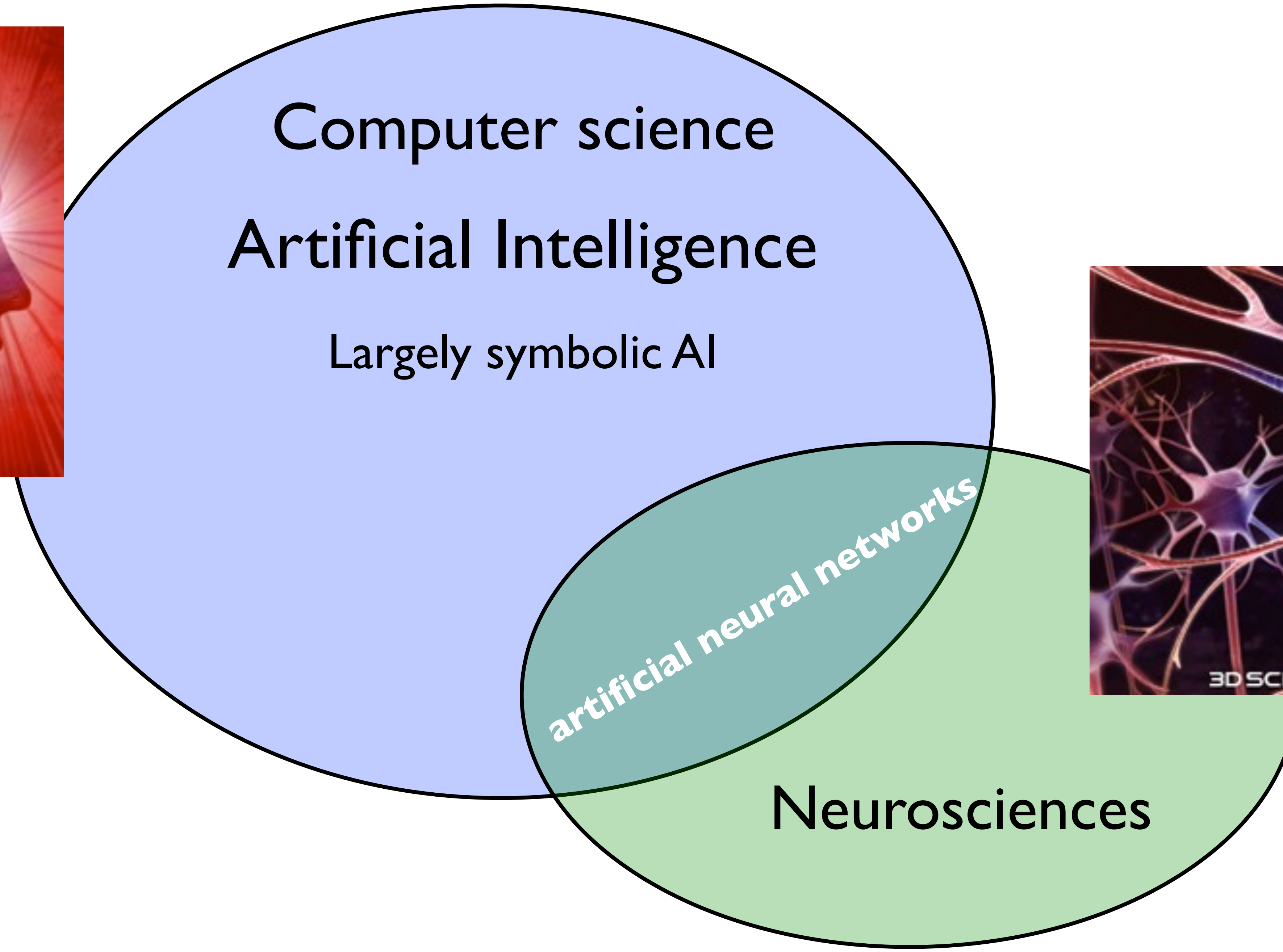
- Two historically opposed approaches to AI:

Neuroscience inspired:
➪ neural nets learning from examples for artificial perception

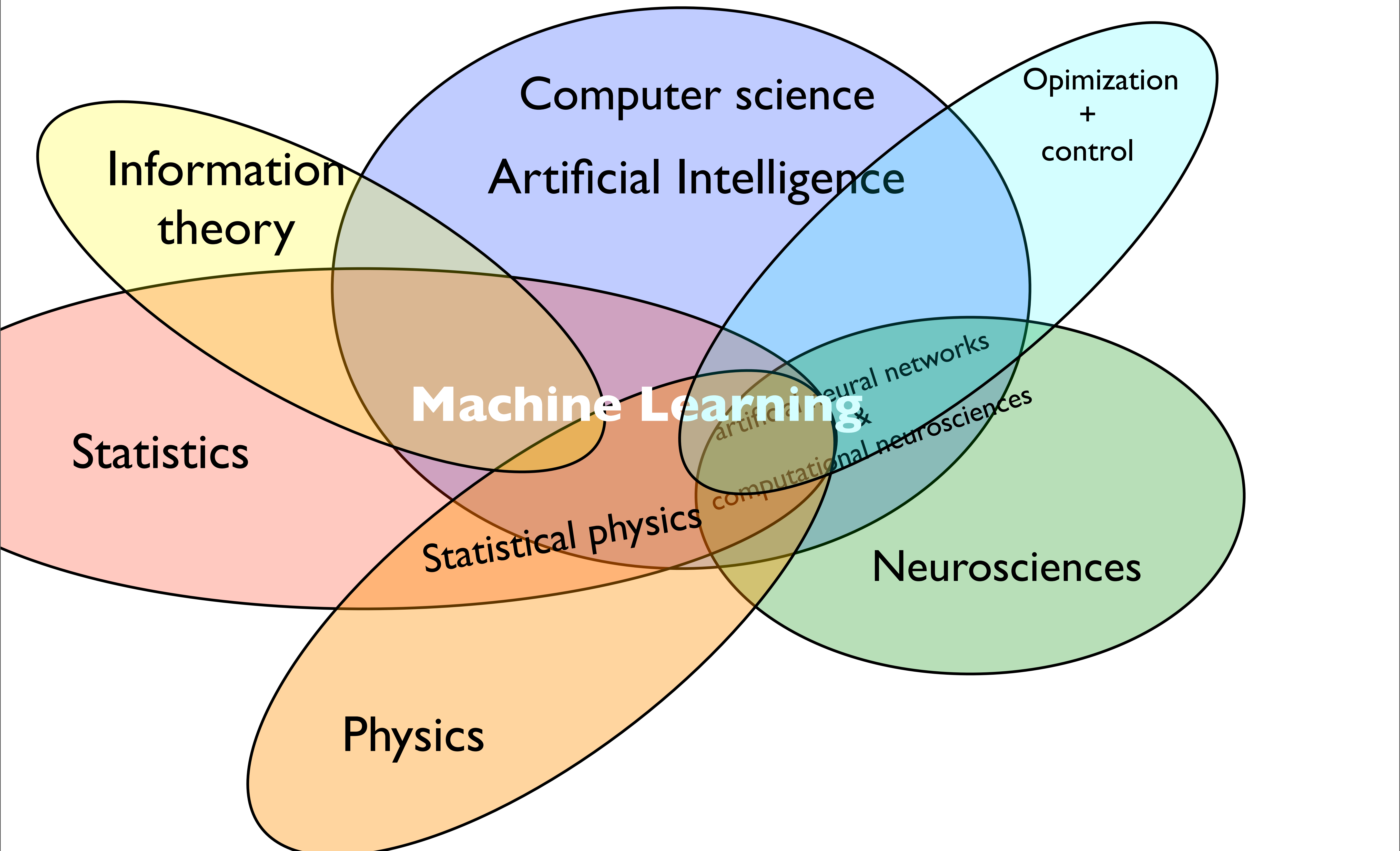
Classical symbolic AI:
Primacy of logical reasoning capabilities
➪ No learning (humans coding rules)
➪ poor handling of uncertainty
Got eventually fixed (Bayes Nets...)

Learning and probabilistic models largely won ➪ machine learning

Artificial Intelligence in the 60s



Current view of ML founding disciplines



What is machine-learning?

A (hypnotized) user's perspective

A scientific (witchcraft) field that

- researches fundamental principles (potions)
- and **develops** magical **algorithms** (spells to invoke)
- capable of leveraging collected data to (automagically) ***produce accurate*** ***predictive*** ***functions*** applicable to similar data (in the future!)

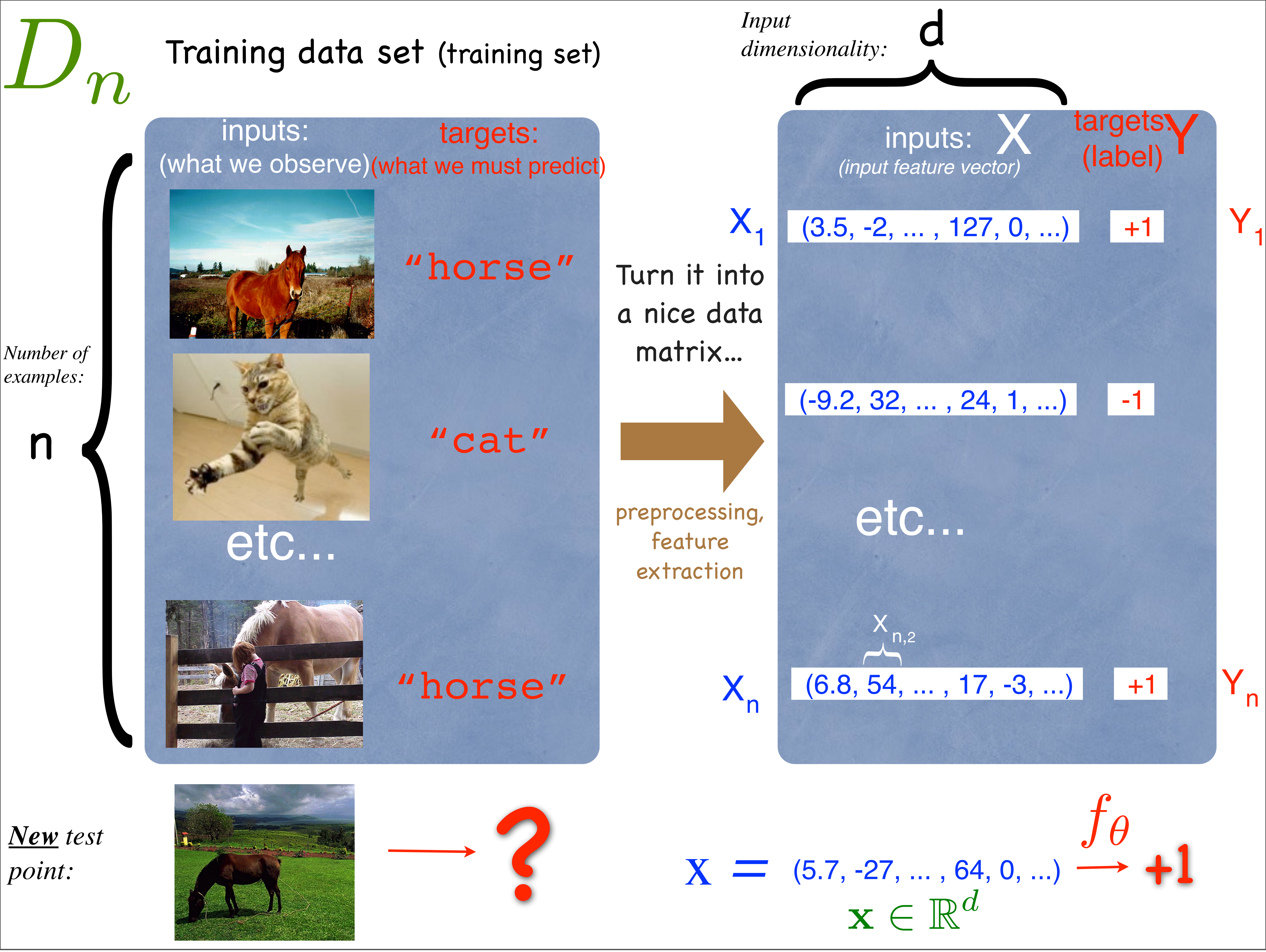
(may also yield informative *descriptive* functions of data)

The key ingredient of machine learning is...

- Collected from nature... or industrial processes.
- Comes stored in many forms (and formats...), strucutred, unstructured, occasionally clean, usually messy, ...
- In ML we like to view data as a **list of examples** (or we'll turn it into one)

➡ ideally ***many** examples* of the *same nature*.

➡ preferably with each example a *vector of numbers* (or we'll first turn it into one!)


D_n
Training data set (training set)
Input dimensionality: d
inputs: (what we observe)
targets: (what we must predict)
"horse"
"cat"
etc...
"horse"
Number of examples:
n
Turn it into a nice data matrix...
preprocessing, feature extraction
inputs: X
(input feature vector)
targets: Y (label)
X_1
(3.5, -2, ... , 127, 0, ...)
+1
Y_1
(-9.2, 32, ... , 24, 1, ...)
-1
etc...
$X_{n,2}$
X_n
(6.8, 54, ... , 17, -3, ...)
+1
Y_n
New test point:
?
$x = (5.7, -27, ... , 64, 0, ...) \xrightarrow{f_\theta} +1$
$x \in \mathbb{R}^d$

Importance of the

Problem dimensions

⇨ Détermines which learning algorithms will be practically applicable (based on their algorithmic complexity and memory requirements).

- Number of examples: **n**
  (sometimes several millions)
- Input dimensionality: **d**
  number of input features characterizing each example
  (often 100 to 1000, sometimes 10000 or much more)
- Target dimensionality ex. number of classes **m**
  (often small, sometimes huge)

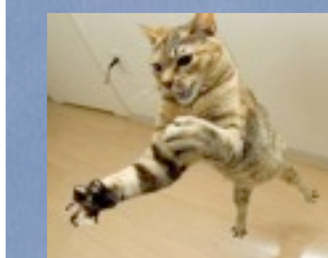
→ Data suitable for ML will often be organized as a matrix: n x (d+1) ou n x (d+m)

Turning messy data into a nice list of examples

data-plumbing

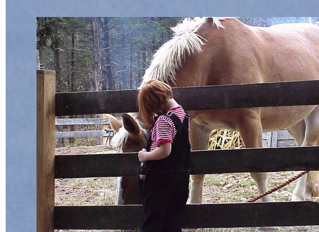
inputs: targets:

"horse



"cat

etc..

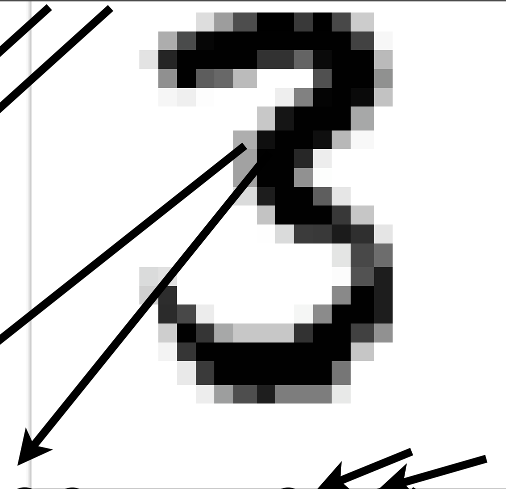


"horse

Key questions to decide what «examples» should be:

- **input:** What is **all** the (potentially relevant) **information** I will have **at my disposal** about a case **<u>when</u>** I will have to make a prediciton about it?(at test time)
- **target:** what I want to predict: Can I get my hands on <u>many</u> such examples that are actually labeled with prediciton targets?

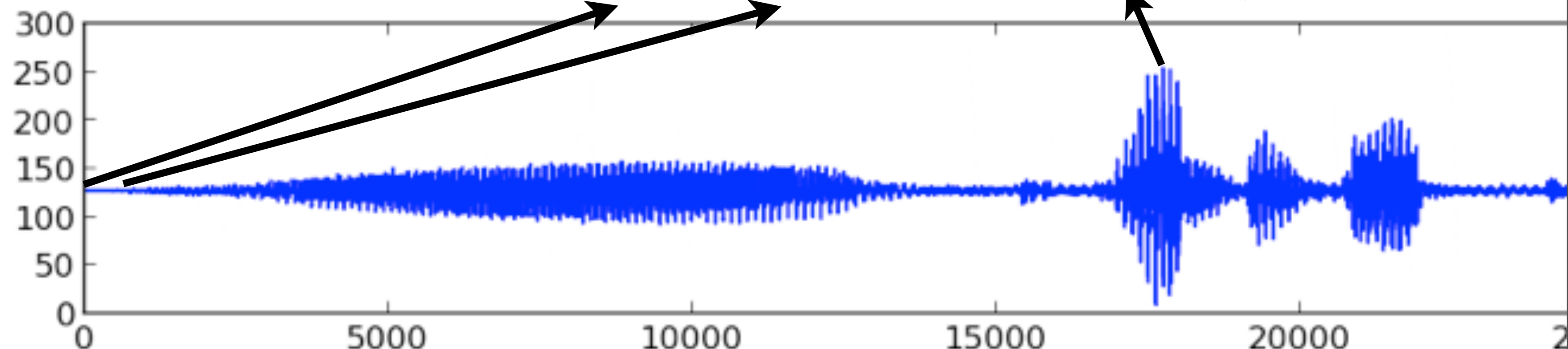
Turning an example into an input vector $\mathbf{x} \in \mathbb{R}^d$



Raw input representation:

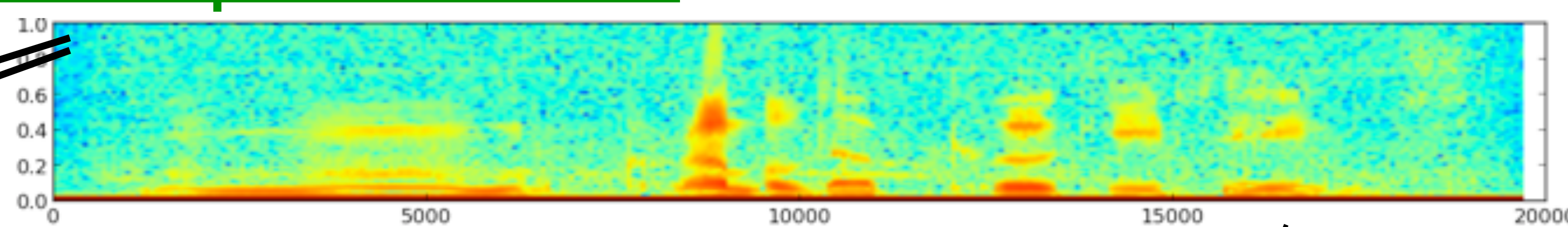
$x = (0, 0, \ldots, 54, 120, \ldots, 0, 0)$

$x = (125, 125, \ldots, 250, \ldots)$



OR some preprocessed representation:

$x = (\ ,\ \ ,\ \ ,\ \ ,\ \ldots)$



we the jumped jumping run elephant dog cat horse

Bag of words for «The cat jumped»: $x = (\ldots 0 \ldots, 0, 1, \ldots 0 \ldots, 1, 0, 0, \ldots, 0, 0, 1, 0, \ldots 0 \ldots$

OR vector of hand-engineered features:

$x = (\text{feature } 1, \ldots, \text{feature } d)$

ex: Histograms of Oriented Gradients

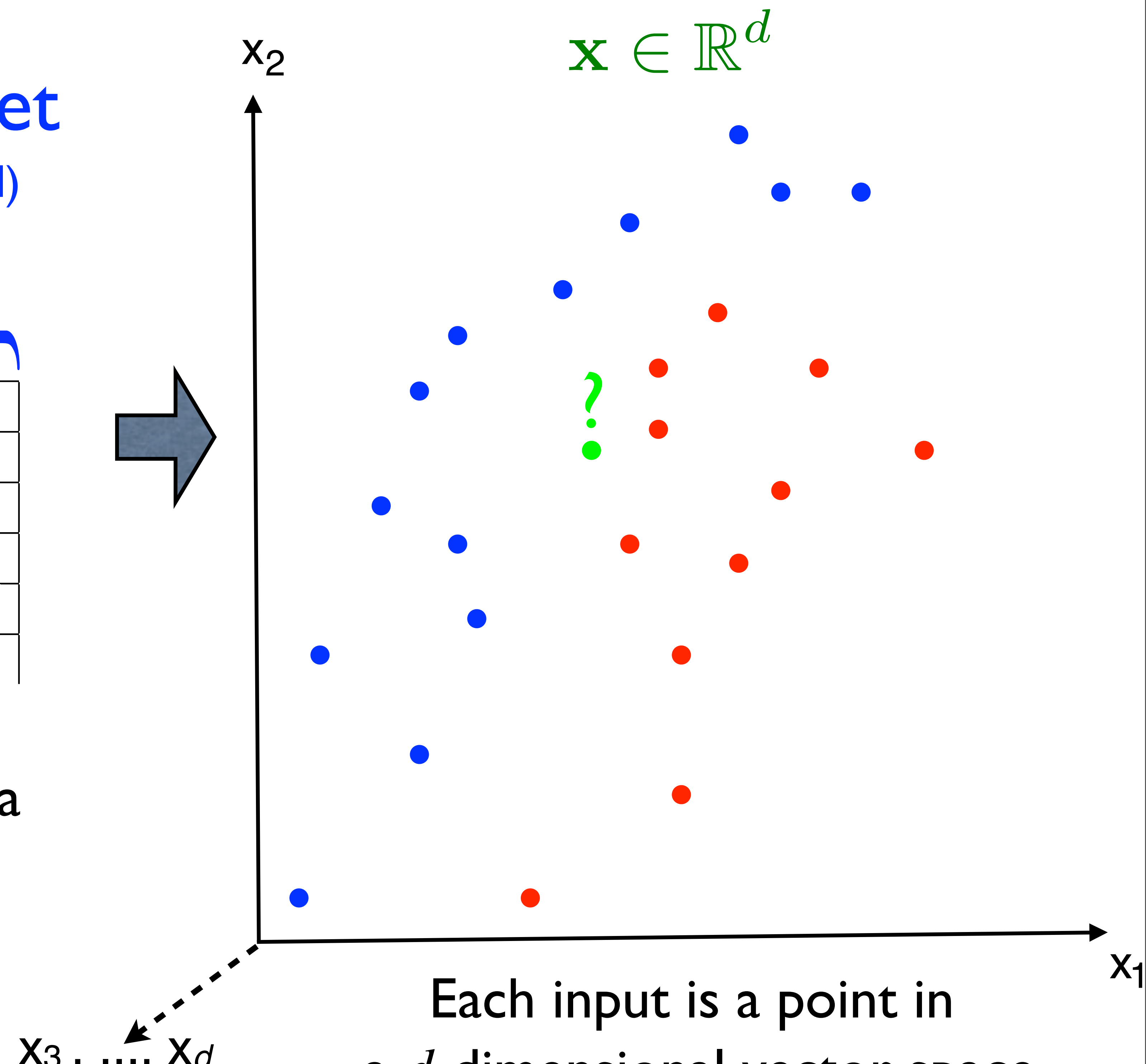
Dataset imagined as a point cloud in a *high-dimensional* vector space

input $\mathbf{x} \in \mathbb{R}^d$ | target (label) y

n examples

$\mathbf{x}_1$	$\mathbf{x}_2$	$\mathbf{x}_3$	$\mathbf{x}_4$	$\mathbf{x}_5$	
0.32	-0.27	+1	0	0.82	1
-0.12	0.42	-1	1	0.22	0
0.06	0.35	-1	1	-0.37	1
0.91	-0.72	+1	0	-0.63	1
...	...	...	...	...	...

Each example (row) is now a d+1-dimensional vector



Each input is a point in a d-dimensional vector space

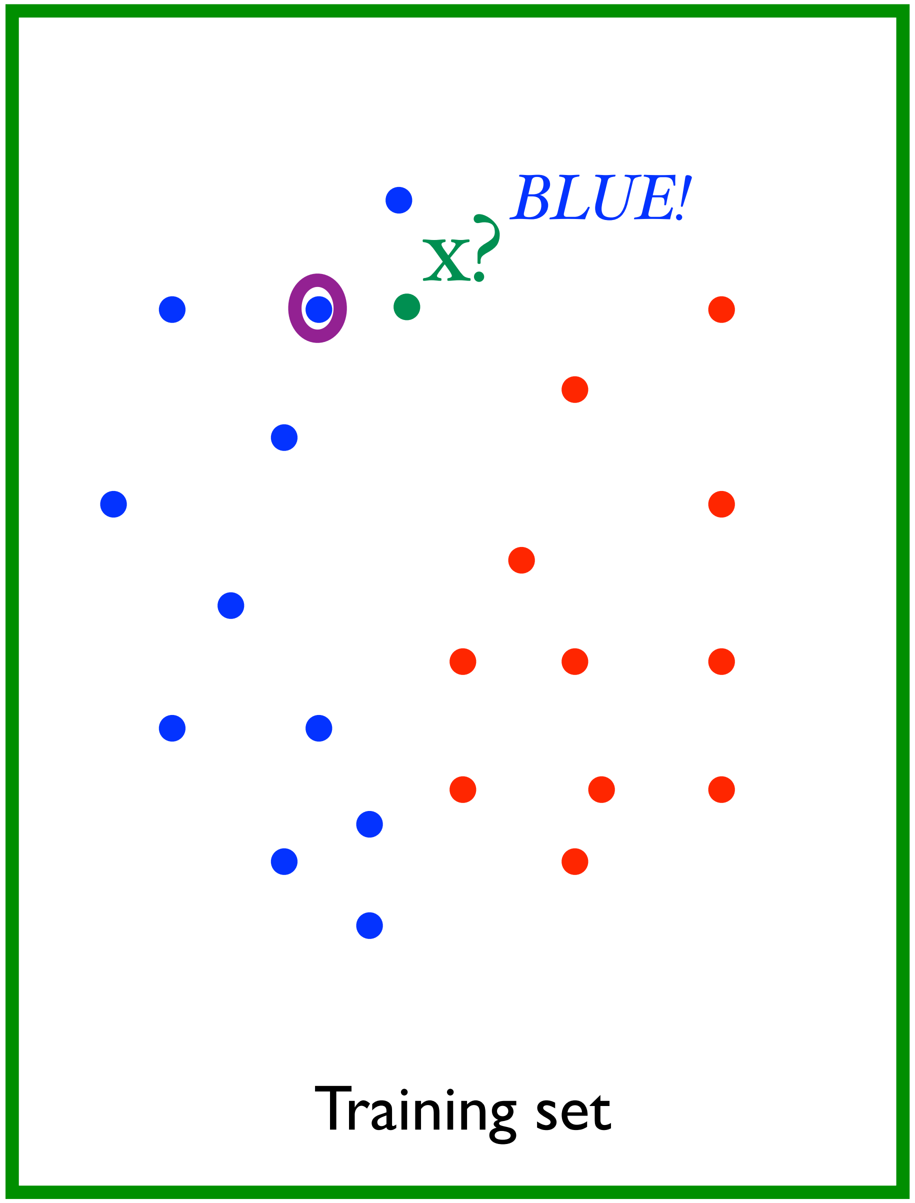
Ex: nearest-neighbor classifier

Algorithm:

For test point x:

- Find **nearest neighbor** of x among the training set according to some distance measure (eg: Euclidean distance).

- Predict that x has the same class as this nearest neighbor.



Machine learning tasks (problem types)

Supervised learning = predict a target y from input x

(and *semi*-supervised learning)

- y represents a category or "class"
  - ➠ classification — binary : $\mathbf{y} \in \{-1, +1\}$ or $\mathbf{y} \in \{0, 1\}$; multiclass : $\mathbf{y} \in \{1, m\}$ or $\mathbf{y} \in \{0, m-1\}$
- y is a real-value number
  - ➠ regression — $\mathbf{y} \in \mathbb{R}$ or $\mathbf{y} \in \mathbb{R}^m$

} Predictive models

Unsupervised learning: no explicit prediciton target y

- model the probability distribution of x
  - ➠ density estimation
- discover underlying structure in data
  - ➠ clustering
  - ➠ dimensionality reduction
  - ➠ (unsupervised) representation learning

} Descriptive modeling

Reinforcement learning: taking good sequential decisions to maximize a reward in an environment influenced by your decisions.

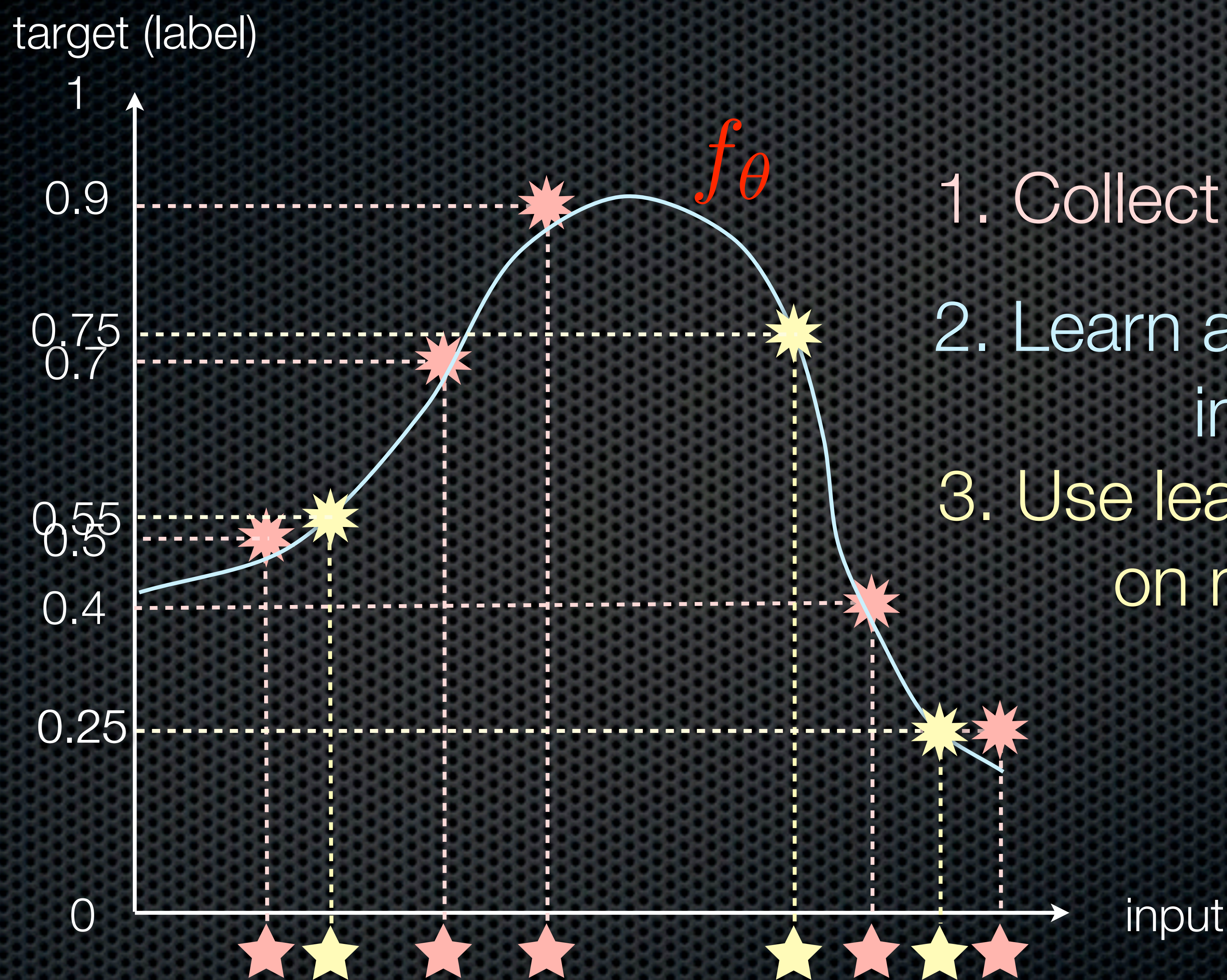
Learning phases

- **Training**: we learn a predictive function f_θ by optimizing it so that it predicts well on the training set.
- **Use** for prediction: we can then use f_θ on new (test) inputs that were not part of the training set.

⇨ The GOAL of learning is *NOT* to learn perfectly *(memorize)* the training set.

⇨ What's important is the ability for the predictor to ***generalize*** well on new (future) cases.

Ex: 1D regression



1. Collect training data
2. Learn a function (predictor)
   input → target
3. Use learned function
   on new inputs

Original slide by Olivier Delalleau

Supervised task:

predict y from x

input $\mathbf{x} \in \mathbb{R}^d$

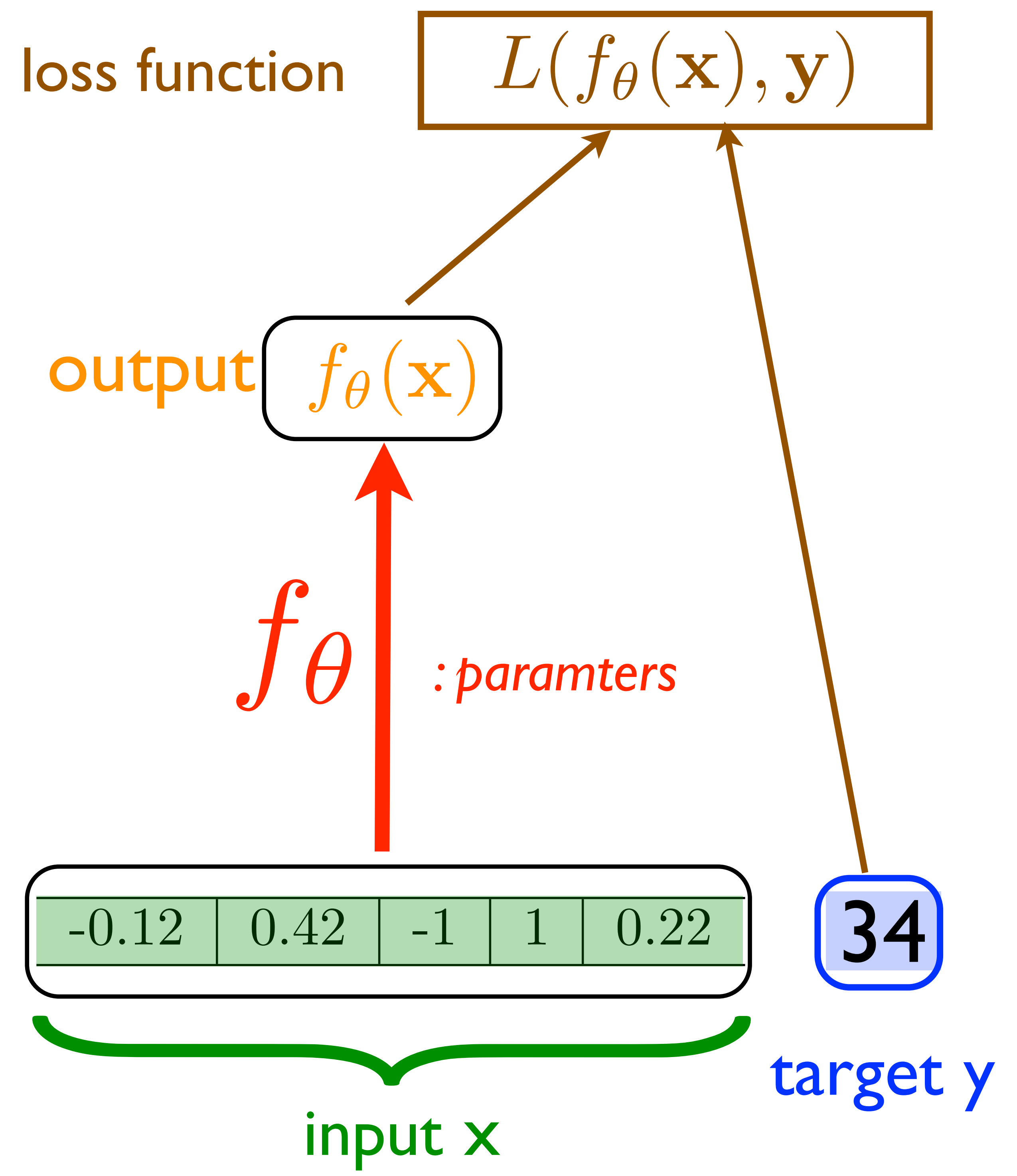
target (label) y

n examples

$\mathbf{x}_1$	$\mathbf{x}_2$	$\mathbf{x}_3$	$\mathbf{x}_4$	$\mathbf{x}_5$	t
0.32	-0.27	+1	0	0.82	113
-0.12	0.42	-1	1	0.22	34
0.06	0.35	-1	1	-0.37	56
0.91	-0.72	+1	0	-0.63	77
...	...	...	...	...	...

Training set D_n

Learn a function f_θ that will minimize prediciton errors as measured by cost (loss) L.



A machine learning algorithm usually corresponds to a combination of the following 3 elements:

(either explicitly specified or implicit)

- ✓ the choice of a specific function family: F
  (often a parameterized family)
- ✓ a way to evaluate the quality of a function $f \in F$
  (typically using a cost (or loss) function L
  mesuring how wrongly f prédicts)
- ✓ a way to search for the «best» function $f \in F$
  (typically an optimization of function parameters to minimize the overall loss over the training set).

Evaluating the quality of a function $f \in F$

and

Searching for the «best» function $f \in F$

Evaluating a predictor $f(x)$

The performance of a predictor is often evaluated using **several** different evaluation metrics:

- Evaluations of true quantities of interest ($ saved, #lifes saved, ...) when using predictor inside a more complicated system.
- «Standard» evaluation metrics in a specific field (e.g. BLEU (Bilingual Evaluation Understudy) scores in translation)
- Misclassification error rate for a classifier (or precision and recall, or F-score, ...).
- The loss actually being optimized by the ML algorithm (often different from all the above...)

Standard loss-functions

- For a density estimation task: $f : \mathbb{R}^d \to \mathbb{R}^+$ a proper probability mass or density function

  negative log likelihood loss: $L(f(x)) = -\log f(x)$

- For a regression task: $f : \mathbb{R}^d \to \mathbb{R}$

  squared error loss: $L(f(x), y) = (f(x) - y)^2$

- For a classification task: $f : \mathbb{R}^d \to \{0, \ldots, m-1\}$

  misclassification error loss: $L(f(x), y) = I_{\{f(x) \neq y\}}$

Surrogate loss-functions

- For a classification task: $f : \mathbb{R}^d \rightarrow \{0, \ldots, m-1\}$

  misclassification error loss: $L(f(x), y) = I_{\{f(x) \neq y\}}$

Problem: it is hard to *optimize* the misclassification loss directly

(gradient is 0 everywhere. NP-hard with a linear classifier) Must use a surrogate loss:

	Binary classifier	Multiclass classifier
Probabilistic classifier	Outputs probability of class 1 $g(x) \approx P(y=1 \mid x)$ Probability for class 0 is $1-g(x)$ Binary cross-entropy loss: $L(g(x),y) = -(y \log(g(x)) + (1-y) \log(1-g(x))$ Decision function: $f(x) = I_{g(x)>0.5}$	Outputs a vector of probabilities: $g(x) \approx (P(y=0 \mid x), \ldots, P(y=m-1 \mid x))$ Negated conditional log likelihood loss $L(g(x),y) = -\log g(x)_y$ Decision function: $f(x) = \operatorname{argmax}(g(x))$
Non-probabilistic classifier	Outputs a «score» $g(x)$ for class 1. score for the other class is $-g(x)$ Hinge loss: $L(g(x),t) = \max(0, 1-tg(x))$ where $t=2y-1$ Decision function: $f(x) = I_{g(x)>0}$	Outputs a vector $g(x)$ of real-valued scores for the m classes. Multiclass margin loss $L(g(x),y) = \max(0,1+\max_{k \neq y}(g(x)_k)-g(x)_y)$ Decision function: $f(x) = \operatorname{argmax}(g(x))$

Expected risk v.s. Empirical risk

Examples (x,y) are supposed drawn i.i.d. from an unknown true distribution $p(x,y)$ (from nature or industrial process)

- Generalization error = Expected risk (or just «Risk»)
  «how poorly we will do on average on the infinity of future examples from that unknown distribution»

$$R(f) = \mathbb{E}_{p(\mathbf{x},\mathbf{y})}[L(f(\mathbf{x}),\mathbf{y})]$$

- Empirical risk = average loss on a finite dataset
  «how poorly we're doing on average on this finite dataset»

$$\hat{R}(f,D) = \frac{1}{|D|} \sum_{(\mathbf{x},\mathbf{y})\in D} L(f(\mathbf{x}),\mathbf{y})$$

where $|D|$ is the number of examples in D

Empirical risk minimization

Examples (x,y) are supposed drawn i.i.d. from an unknown true distribution p(x,y) (nature or industrial process)

- We'd love to find a predictor that minimizes the generalization error (the expected risk)
- But can't even compute it! (expectation over unknown distribution)
- Instead: **Empirical risk minimization principle**
  «Find predictor that minimizes average loss over a trainset»

$$\hat{f}(D_{\text{train}}) = \underset{f \in F}{\operatorname{argmin}}\, \hat{R}(f, D_{\text{train}})$$

This is the training phase in ML

Evaluating the generalization error

- We can't compute expected risk $R(f)$
- But $\hat{R}(f, D)$ is a good estimate of $R(f)$ provided:
  - D was not used to find/choose f
    otherwise estimate is biased ⇨ can't be the training set!
  - D is large enough (otherwise estimate is too noisy); drawn from p

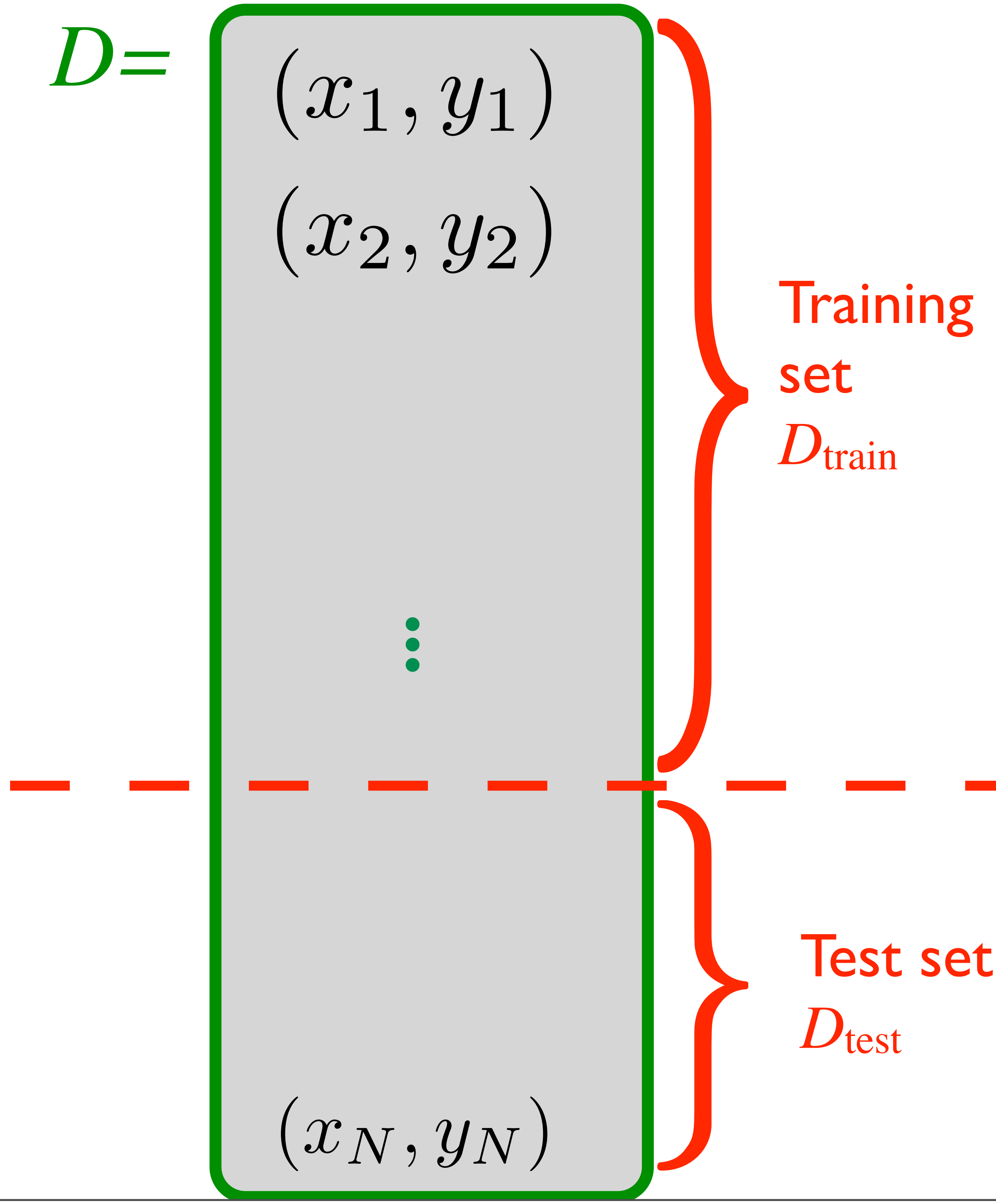
➡ Must keep a separate test-set $D_{\text{test}} \neq D_{\text{train}}$ to properly estimate generalization error of $\hat{f}(D_{\text{train}})$:

$$R(\hat{f}(D_{\text{train}})) \approx \hat{R}(\hat{f}(D_{\text{train}}), D_{\text{test}})$$

generalization error | average error on **test**-set (never used for training)

This is the <u>test</u> phase in ML

Simple train/test procedure



- Provided large enough dataset D drawn from $p(\mathrm{x,y})$
- Make sure examples are in random order.
- Split dataset in **two:** D_{train} and D_{test}
- Use D_{train} to choose/ optimize/find best predictor $f = \hat{f}(D_{\text{train}})$
- Use D_{test} to evaluate generalization performance of predictor f.

Model selection

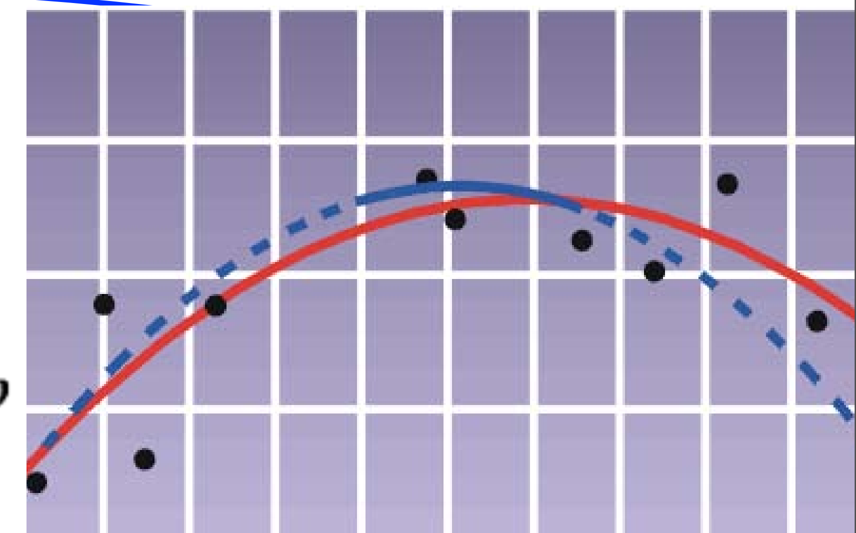
Choosing a specific function family F

Ex. of parameterized function families

$F_{polynomial\ p}$

Polynomial predictor (of degree p):

$$f(x) = b + a_1 x + a_2 x^2 + a_3 x^3 + \ldots + a_p x^p$$



F_{linear}

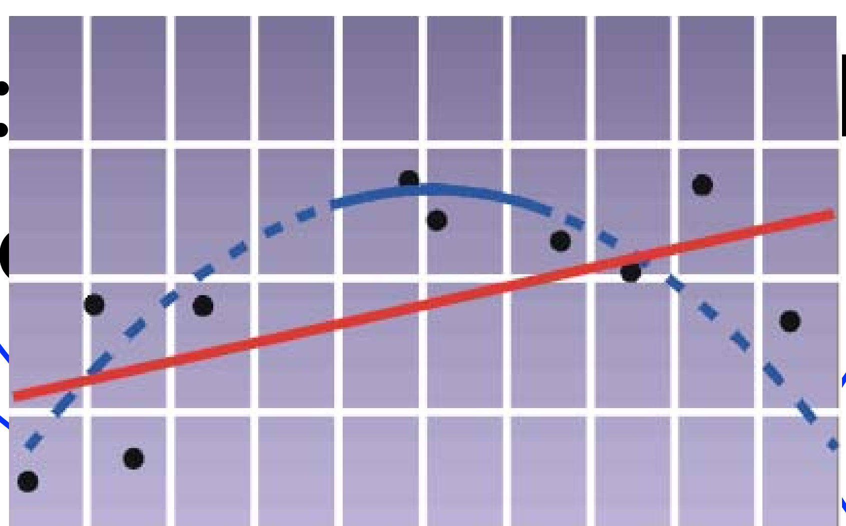
Linear (affine) predictor: $f_\theta(x) = wx + b$ (in 1 dimension)

(«linear regression») $f_\theta(x) = w^T x + b$ (in d dimensions)

$$\theta = \{w \in \mathbb{R}^d, b \in \mathbb{R}\}$$

Q: ... lest

pr...



F_{const}

Constant predictor: $f_\theta(x)=b$

where $\theta=\{b\}$

(always predict the same value or class!)

Capacity of a learning algorithm

- Choosing a specific Machine Learning algorithm means choosing a specific function family F.
- How «big, rich, flexible, expressive, complex» that family is, defines what is informally called the «capacity» of the ML algorithm.

  Ex: capacity($F_{polynomial\ 3}$) > capacity(F_{linear})
- One can come up with <u>several</u> formal measures of «capacity» for a function family / learning algorithm (e.g. VC-dimension Vapnik–Chervonenkis)
- One rule-of-thumb estimate, is the number of adaptable parameters: i.e. how many *scalar* values are contained in θ.

  Notable exception: chaining many linear mappings is still a linear mapping!

Effective capacity, and capacity-control hyper-parameters

The «effective» capacity of a ML algo is controlled by:

- Choice of ML algo, which determines big family F
- Hyper-parameters that further specify F
  e.g.: degree p of a polynomial predictor; Kernel choice in SVMs;
  #of layers and neurons in a neural network
- Hyper-parameters of «regularization» schemes
  e.g. constraint on the norm of the weights w
  (⇨ ridge-regression; L_2 weight decay in neural nets);
  Bayesian prior on parameters; noise injection (dropout); ...
- Hyper-parameters that control early-stopping of the iterative search/optimization procedure.
  (⇨ won't explore as far from the initial starting point)

Popular classifiers

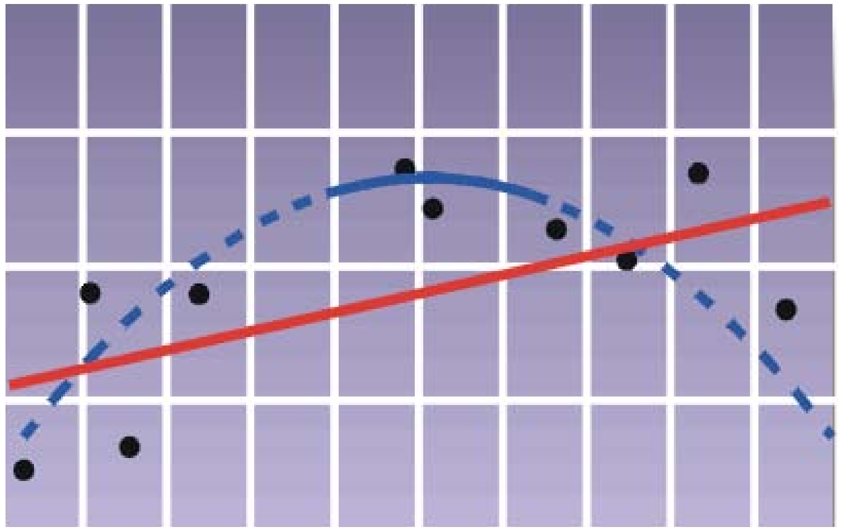
their parameters and hyper-parameters

Algo	**Capacity-control hyperparameters**	**Learned parameters**
logistic regression (L2 regularized)	strength of L2 regularizer	w,b
linear SVM	C	w,b
kernel SVM	C; kernel choice & params (σ for RBF; degree for polynomal)	support vector weights: α
neural network	layer sizes; early stop; ...	layer weight matrices
decision tree	depth	the tree (with index and threshold of variables)
k-nearest neighbors	k; choice of metric	memorizes trainset

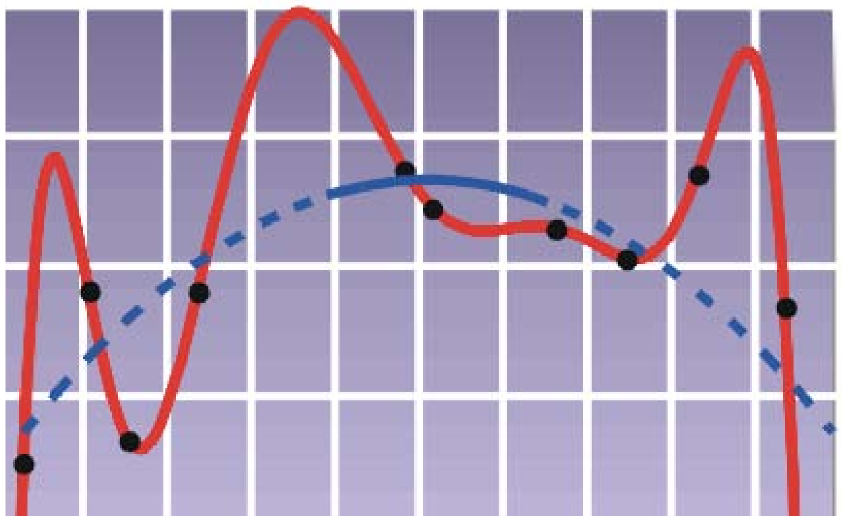
Tuning the capacity

- Capacity must be optimally tuned to ensure good generalization
- by choosing Algorithm and hyperparameters
- to avoid under-fitting and over-fitting.

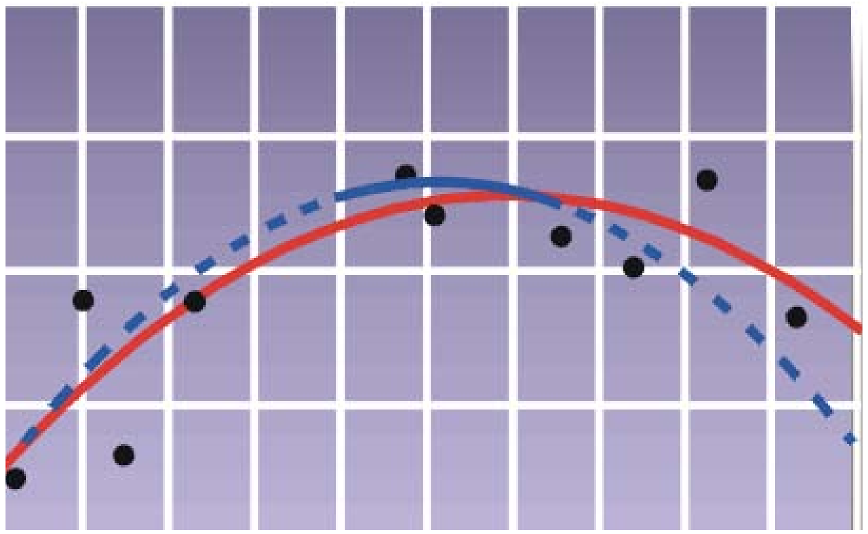
Ex: 1D regression with polynomial predictor



capacity too low
⇨under-fitting



capacity too high
⇨over-fitting



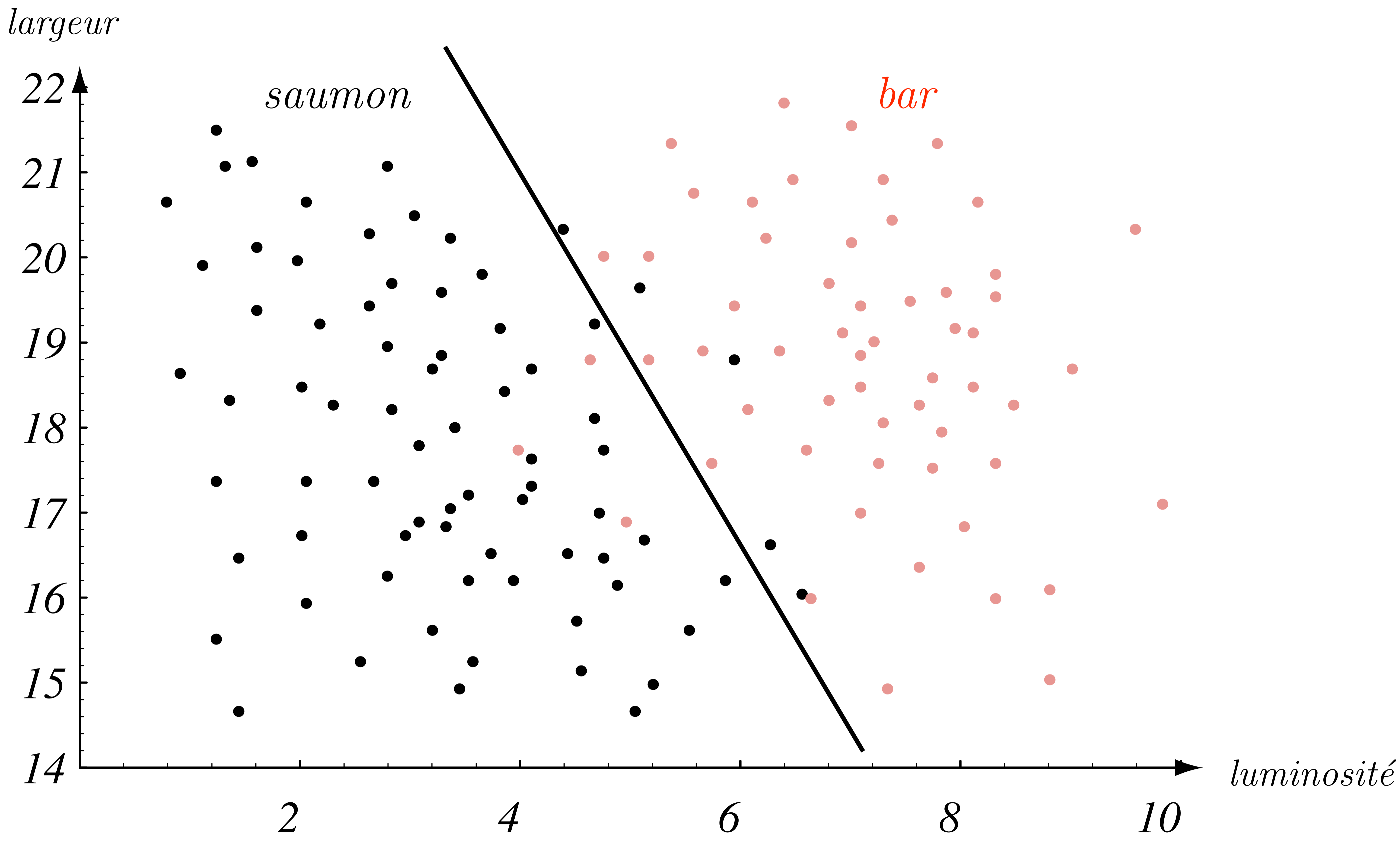
optimal capacity
⇨good generalisation

performance on training set is not a good estimate of generalization

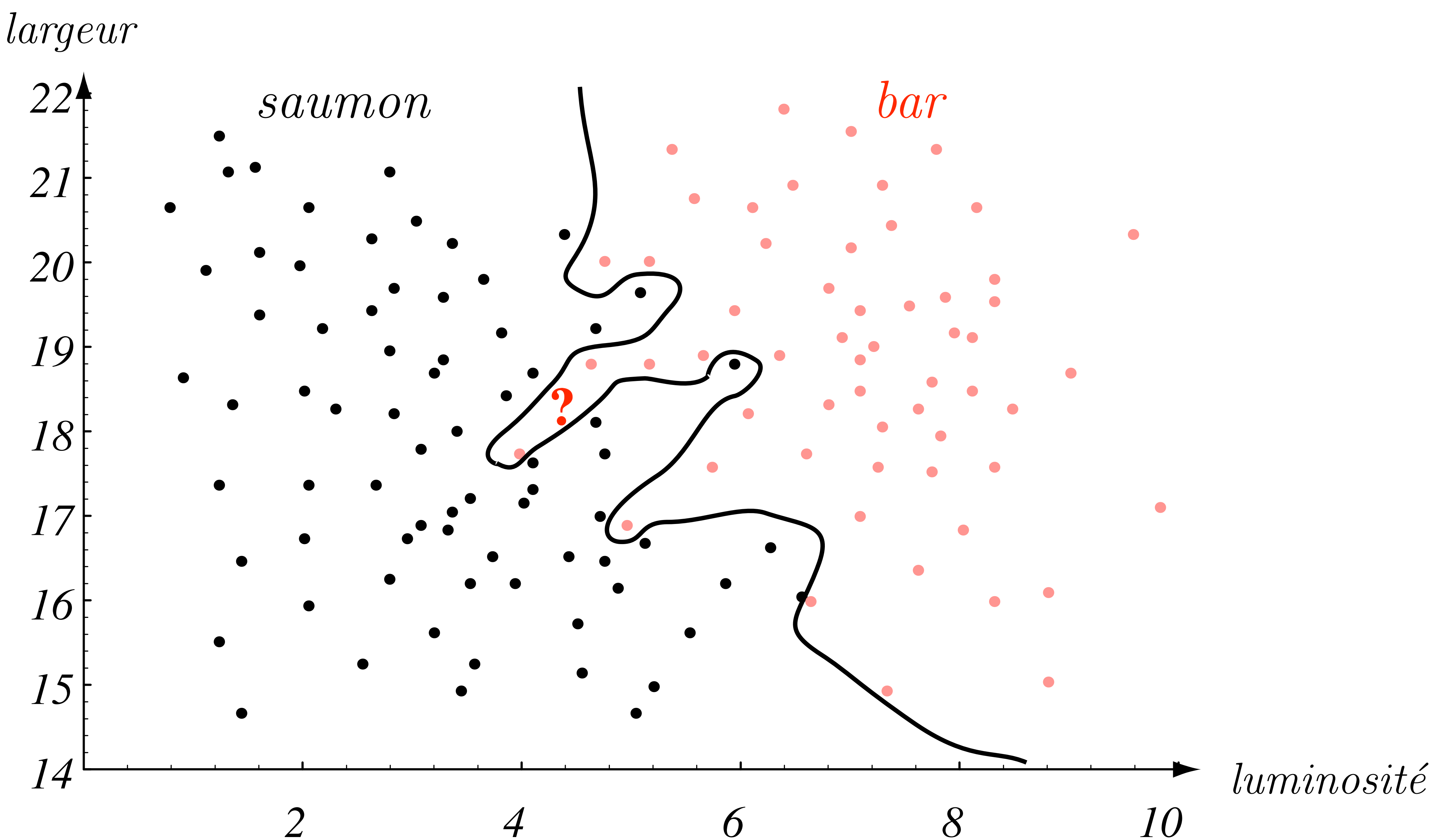
Ex: 2D classification

Linear classifier

- Function family too poor
  (too inflexible)
- = **Capacity too low** for this problem
  (relative to number of examples)
- => Under-fitting

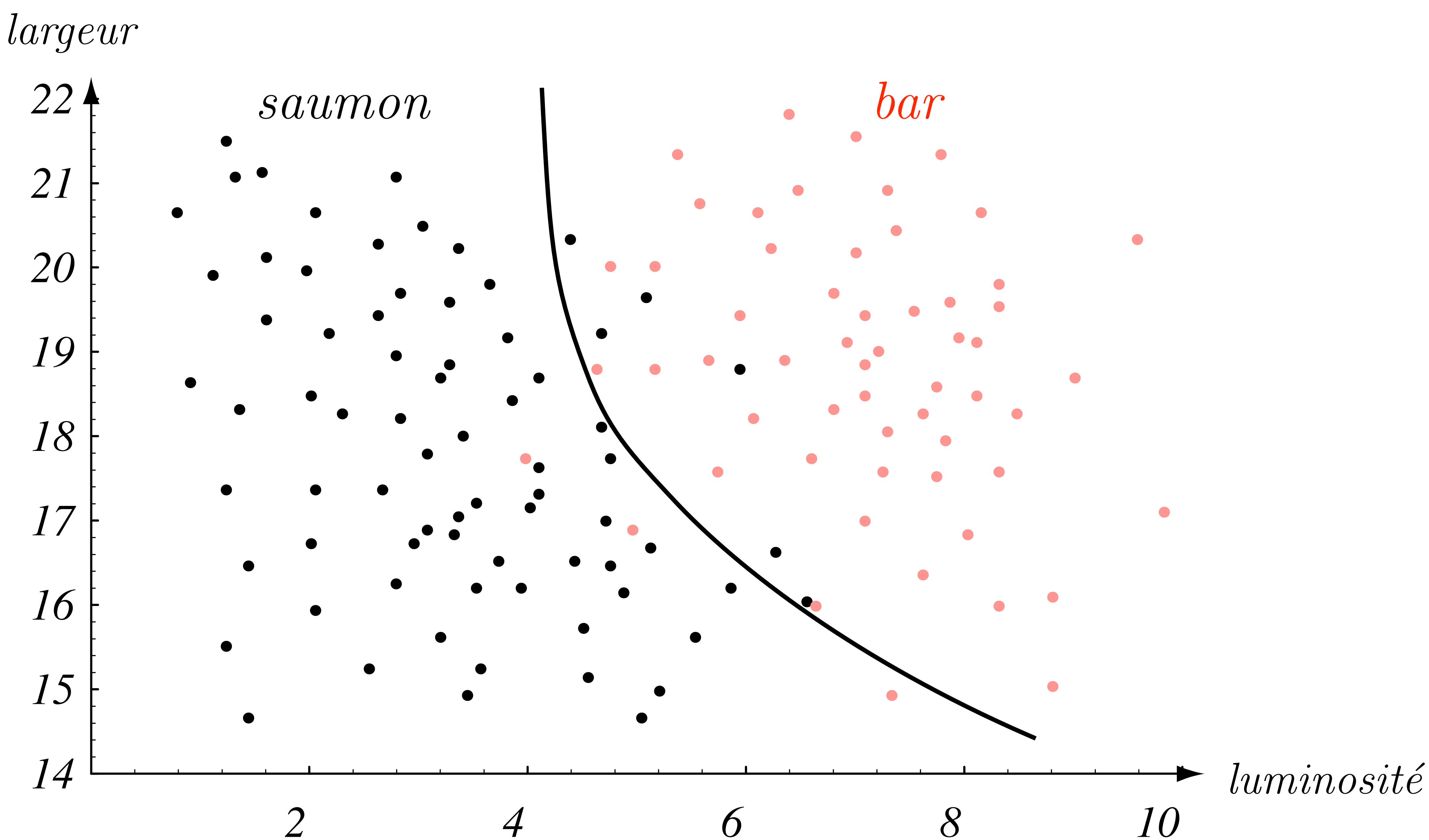


- Function family too rich
  (too flexible)
- = **Capacity too high** for this problem
  (relative to the number of examples)
- => Over-fitting

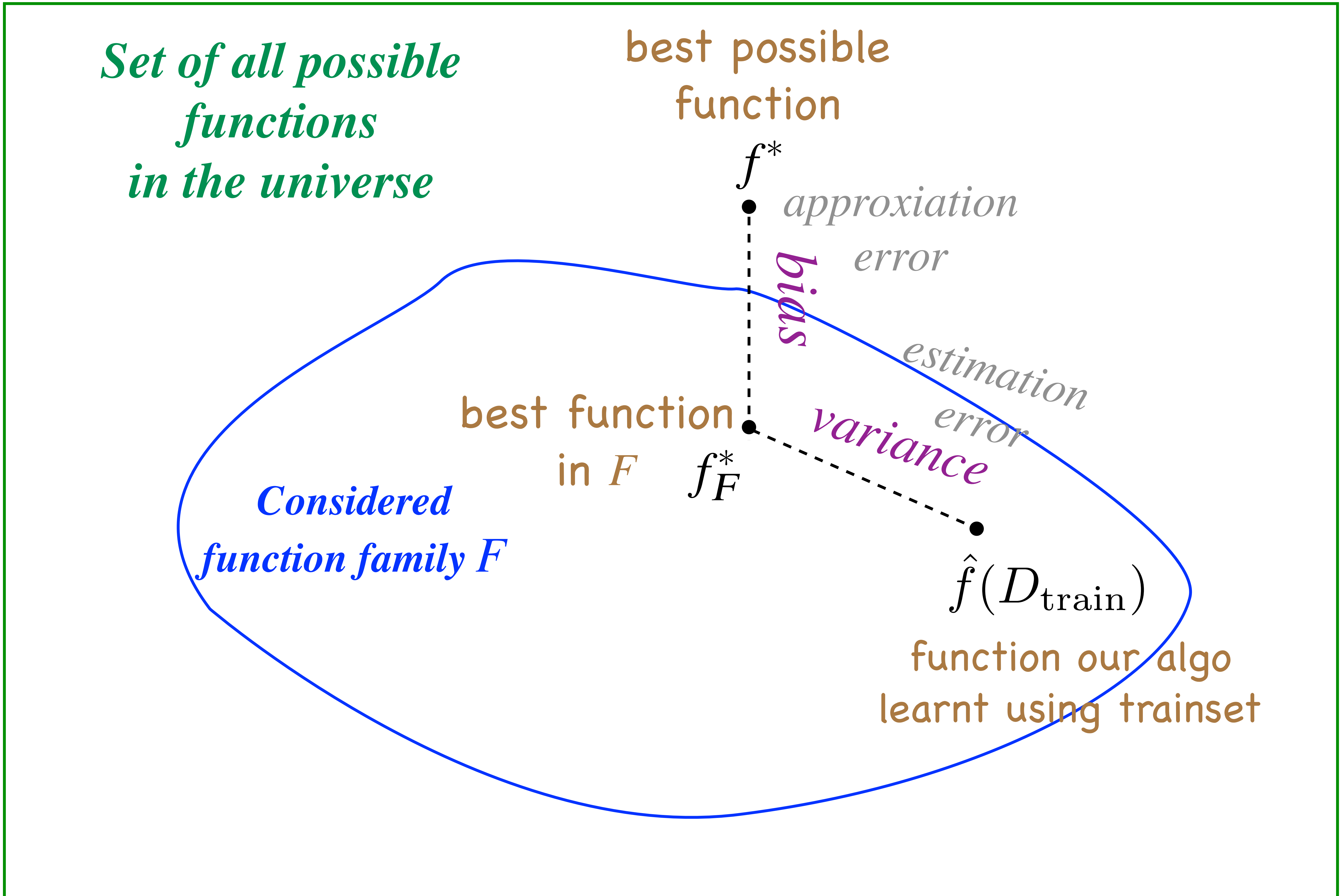


Nombre d'erreurs d'entrainement: 0

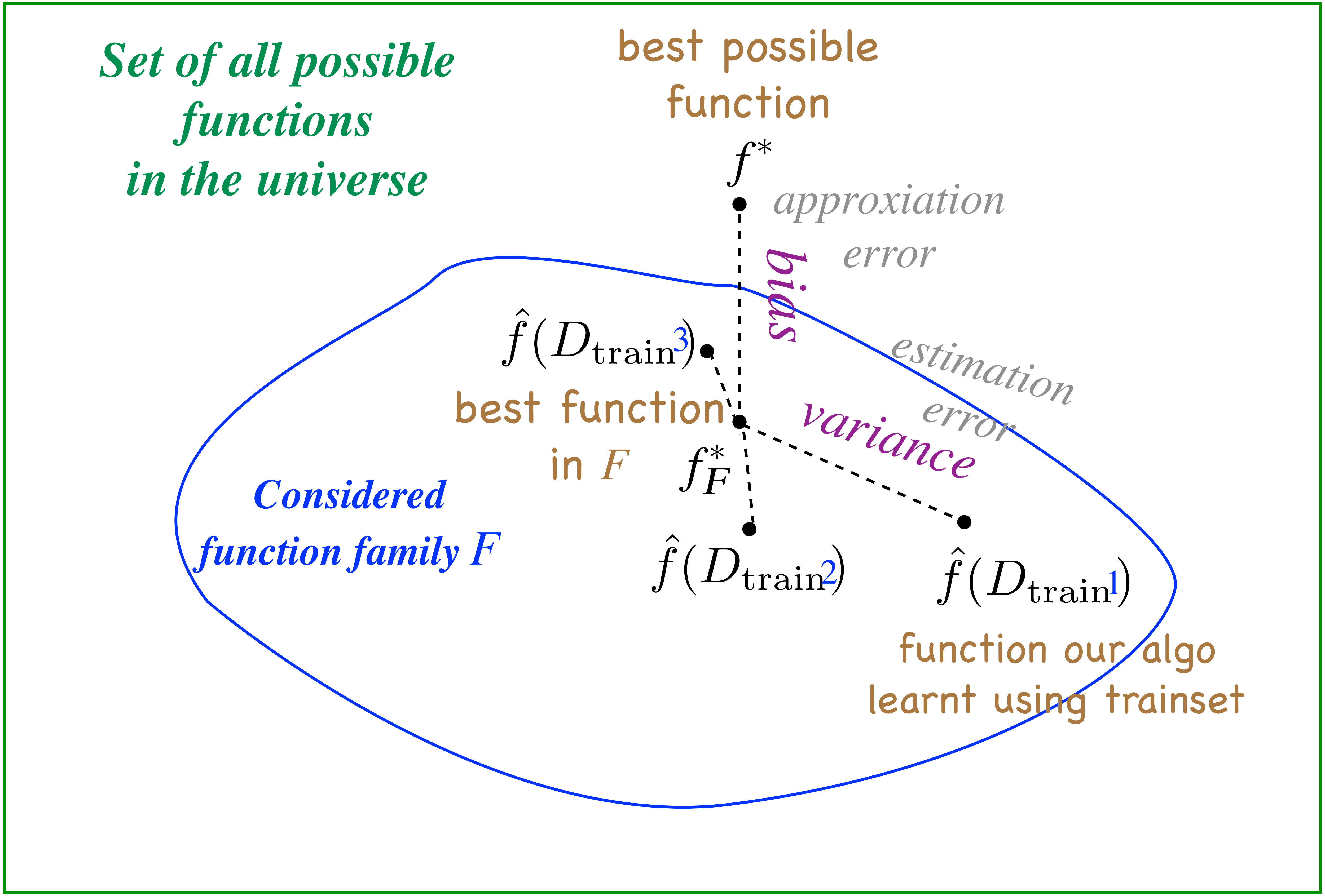
- **Optimal capacity** for this problem
  (par rapport à la quantité de données)
- => Best generalization
  (on future test points)



Decomposing the generalization error



What is responsibe for the variance?



Optimal capacity
& the biais-variance dilemma

- Choosing richer F: capacity ↑
  ➪ bias ↓ *but* variance ↑.
- Choosing smaller F : capacity ↓
  ➪ variance ↓ *but* bias ↑.
- Optimal compromise... will depend on number of examples ***n***
- Bigger ***n*** ➪ variance ↓
  So we can afford to increase capacity (to lower the bias)
  ➪ can use more expressive models
- The best regularizer is more data!

Model selection how to

Make sure examples are in random order

Split data D in 3: D_{train} D_{valid} D_{test}

$D=$

(x_1, y_1)

(x_2, y_2)

⋮

(x_N, y_N)

Training set D_{train}

Validation set D_{valid}

Test set D_{test}

Model selection meta-algorithm:

For each considered model (ML algo) $\mathbf{A}$:

For each considered hyper-parameter config λ:

- train model $\mathbf{A}$ with hyperparams λ on D_{train}

$$\hat{f}_{\mathbf{A}_\lambda} = \mathbf{A}_\lambda(D_{\text{train}})$$

- evaluate resulting predictor on D_{valid} (with preferred evaluation metric)

$$e_{\mathbf{A}_\lambda} = \hat{R}(\hat{f}_{\mathbf{A}_\lambda}, D_{\text{valid}})$$

Locate $\mathbf{A}^*, \lambda^*$ that yielded best $e_{\mathbf{A}_\lambda}$

Either return $f^* = f_{\mathbf{A}^*_{\lambda^*}}$

Or retrain and return

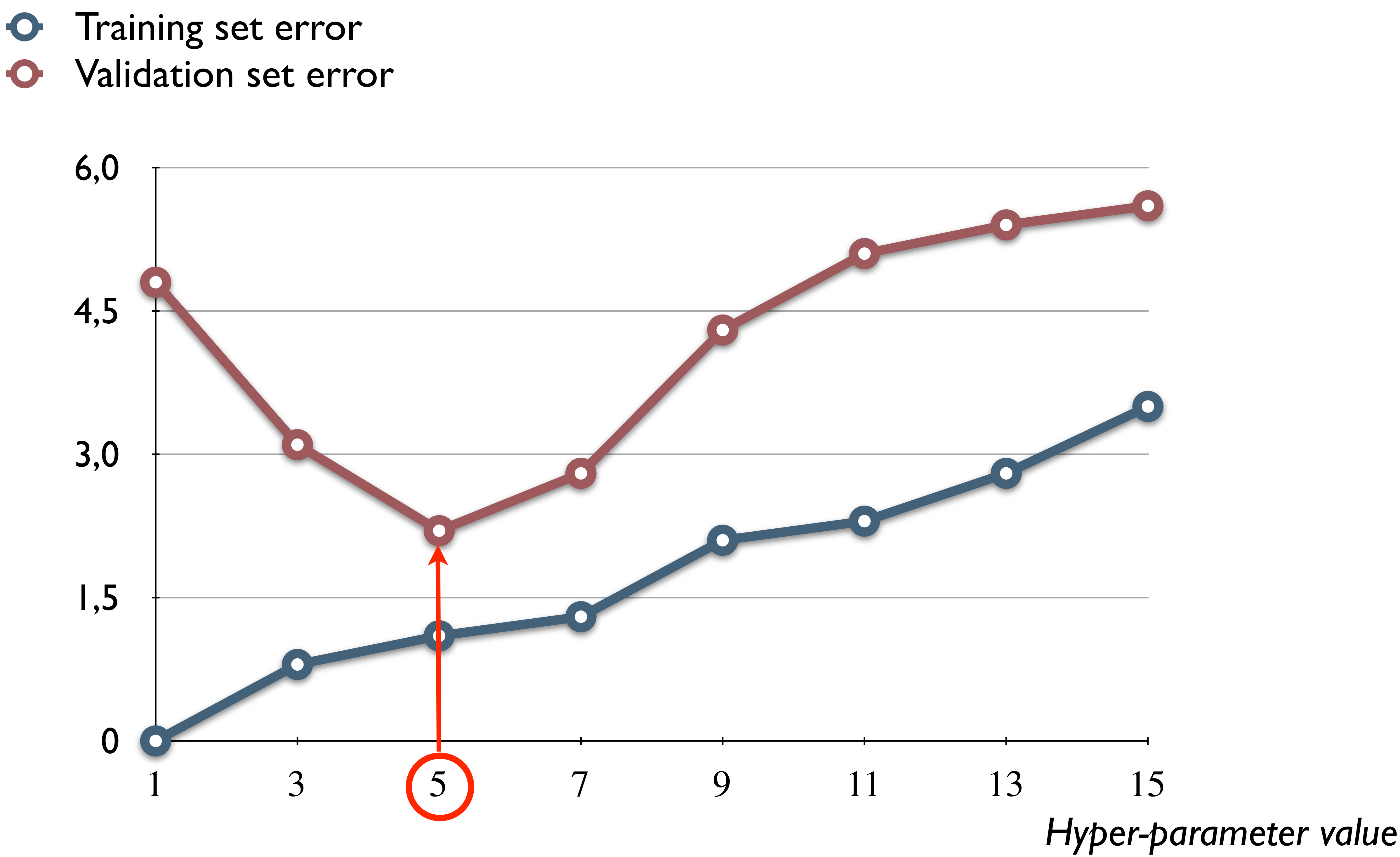
$$f^* = \mathbf{A}^*_{\lambda^*}(D_{\mathbf{train}} \cup D_{\mathbf{valid}})$$

Finally: compute unbiased estimate of generalization performance of f^* using D_{test}

$$\hat{R}(f^*, D_{\mathbf{test}})$$

D_{test} must never have been used during training or model selection to select, learn, or tune anything.

Ex of model *hyper-parameter* selection



Hyper-parameter value which yields smallest error on validaiton set is 5
(it was 1 for the training set)

Question

What if we selected capacity-control hyper-parameters that yield best performance on the training set?

What would we tend to select?

Is it a good idea? Why?

Model selection procedure summary:

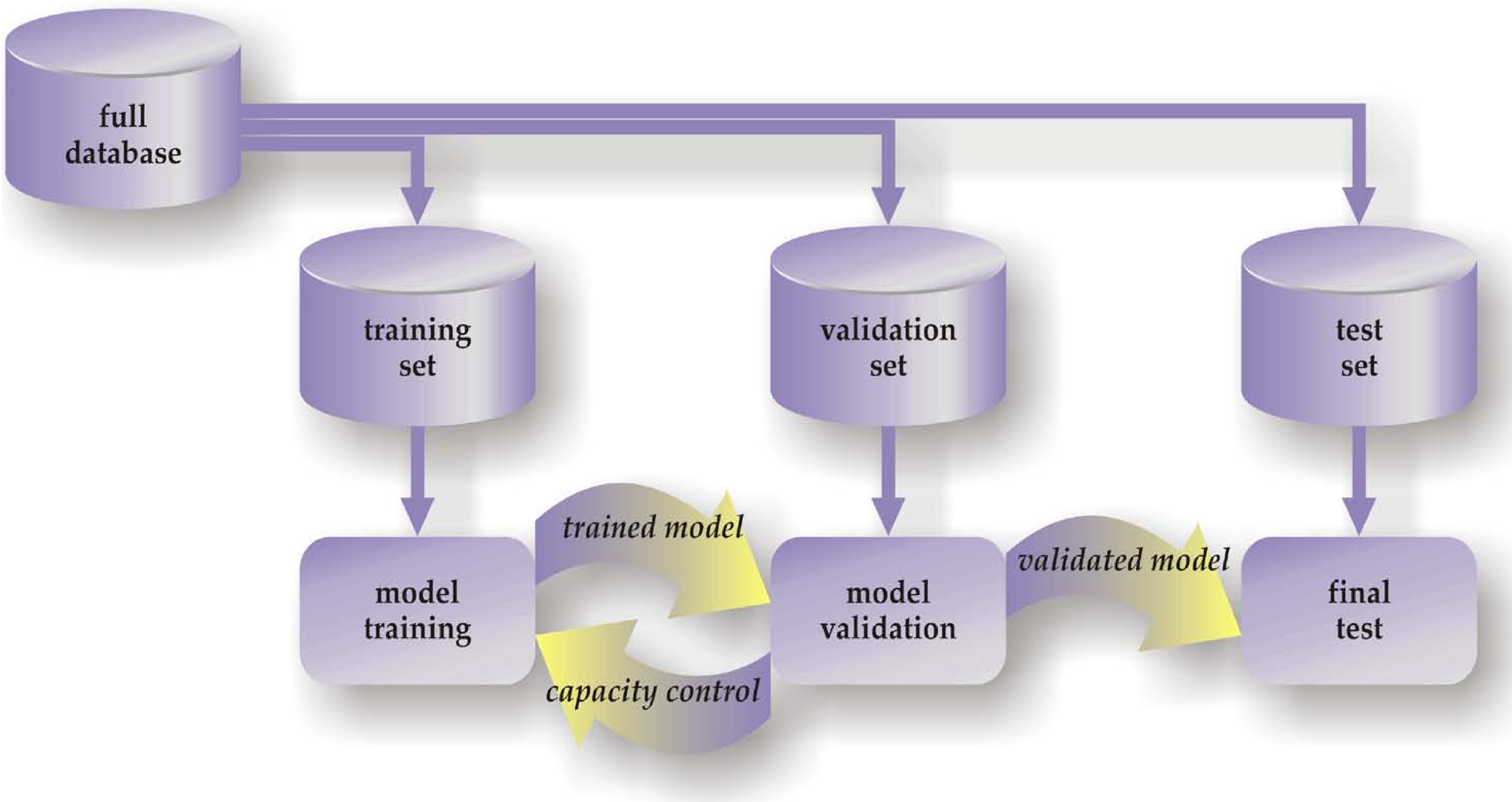
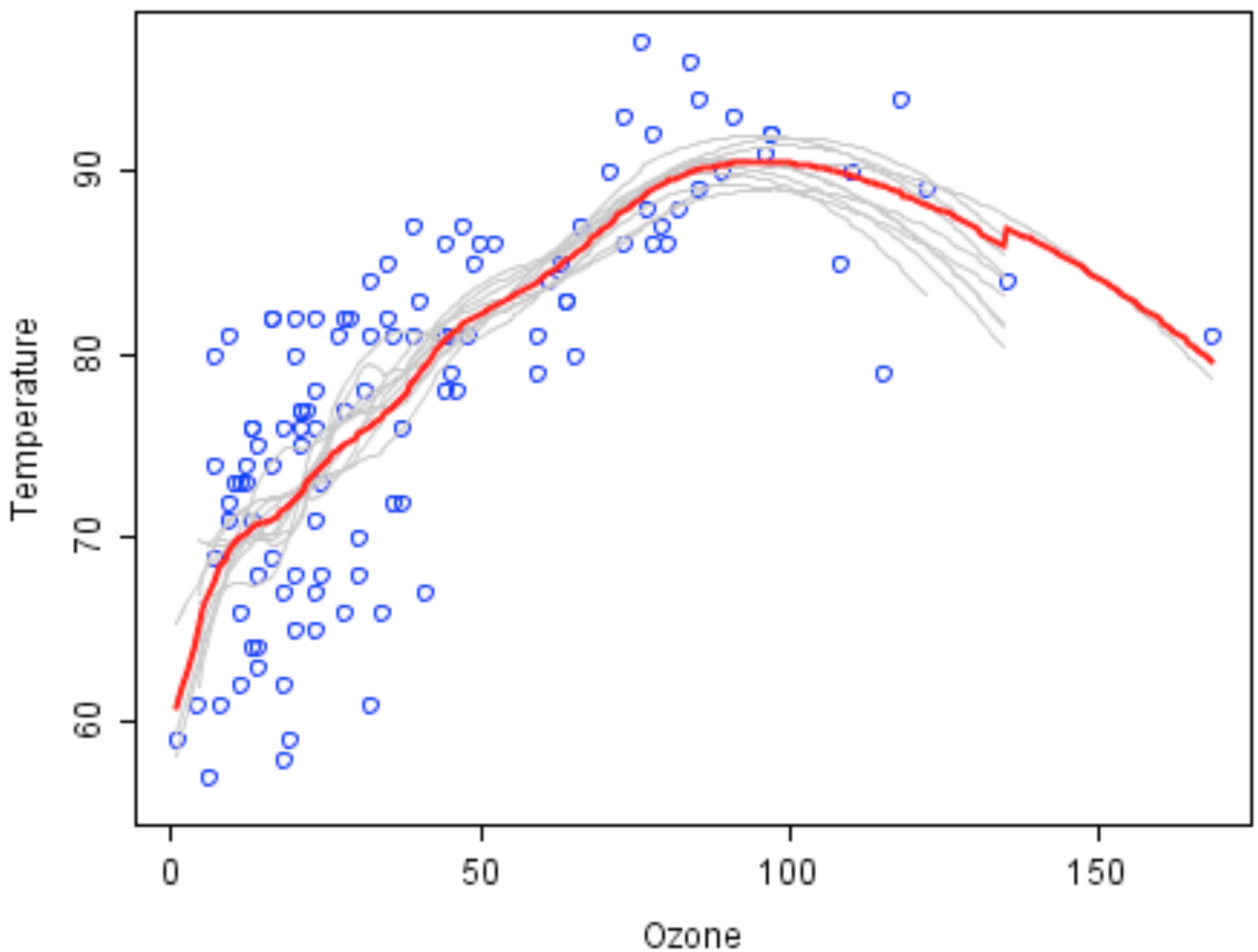


Figure by Nicolas Chapados

Ensemble methods

- Principle: train and combine multiple predictors to good effect
- Bagging: average many high-variance predictors
  ➪ variance ↓
  (e.g.: average deep trees ➪ Random decision forests)
- Boosting: build weighted combination of low-capacity classifiers
  ➪ bias ↓ and capacity ↑
  (e.g. boosting shallow trees; or linear classifiers)

Bagging
for reducing variance on a regression problem



How to obtain non-linear predictor with a linear predictor

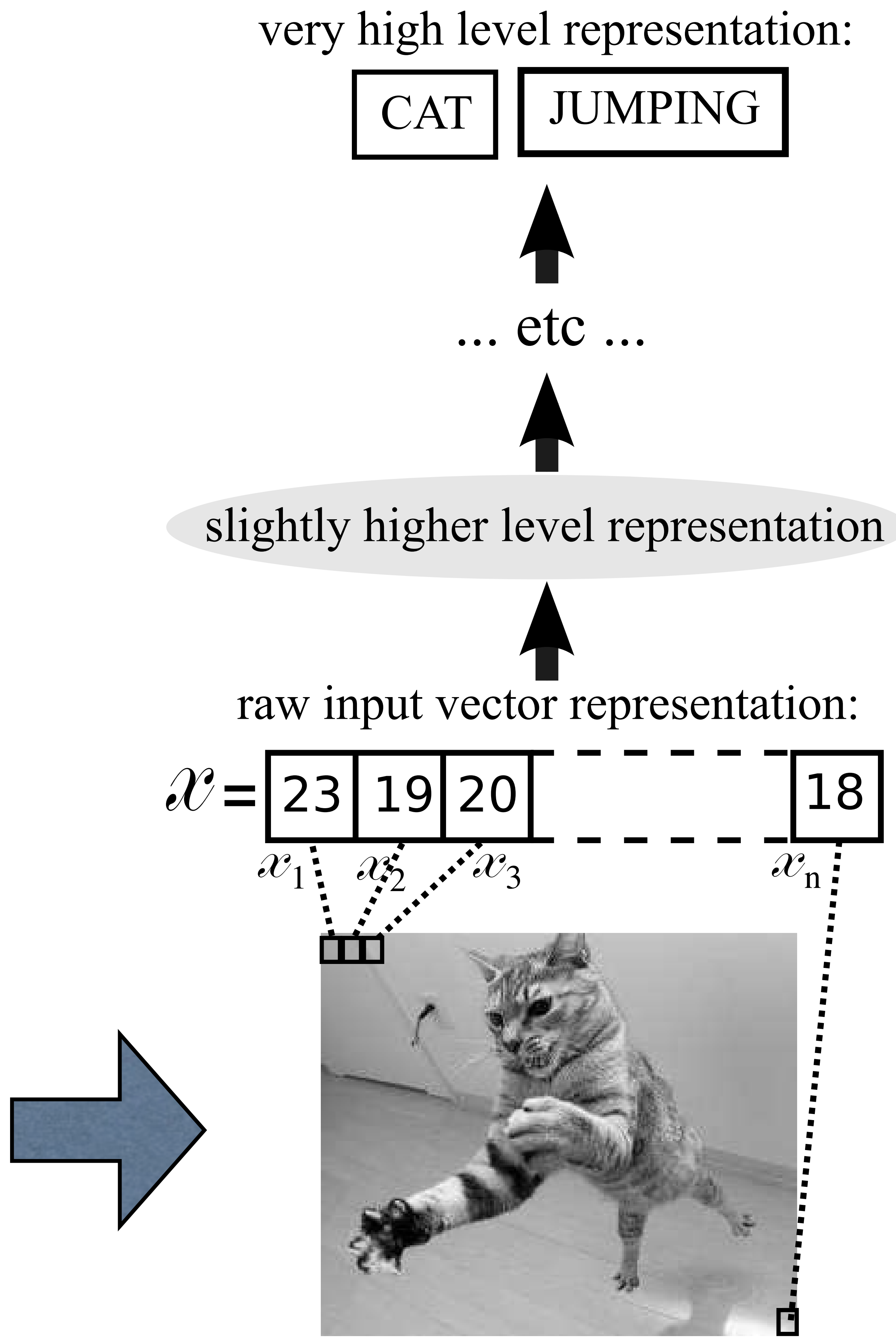
Three ways to map x to a feature representation $\tilde{\mathbf{x}} = \phi(\mathbf{x})$

- Use an explicit fixed mapping (ex: hand-crafted features)

- Use an implicit fixed mapping
  ➠Kernel Methods (SVMs, Kernel Logistic Regression ...)

- Learn a parameterized mapping
  (i.e. let the ML algo learn the new representation)
  ➠ Multilayer feed-forward Neural Networks
  such as Multilayer Perceptrons (MLP)

Levels of representation



Questions ?