

IFT6390

Fondements de l'apprentissage machine

General framework of learning
Evaluation of the generalization performance.
Model selection
Learning curves.

Professor: Ioannis Mitliagkas
Slides: Pascal Vincent

Homework I

- Out this week
- Practical part
- Theoretical part
- You will submit individually (no groups)
- Another glance into the type of questions
 - Useful for students trying to decide
- You can discuss, but write down own solutions

Plagiarism

- Write your own solutions -> success in exams
- It is against university rules to copy your classmates' answers
- Gradescope allows us to identify cases
- We are required to report egregious cases

Start of our lecture

Probabilistic approach to learning

- It is assumed that the data is generated by an unknown process.
- X, Y is seen as a pair of random variables, distributed according to an unknown probability law $P(X, Y)$.
- X (a vector variable) is itself seen as a set of scalar random variables.

$$P(X, Y) = P(X_{[1]}, \dots, X_{[d]}, Y)$$

General framework of learning: the dataset

- Data
 - $D_n = (Z_1, Z_2, \dots, Z_n)$ generated by “nature”
 - IID: “independent and identically distributed”
 - drawn from the same UNKNOWN distribution $p(Z)$
 - independently

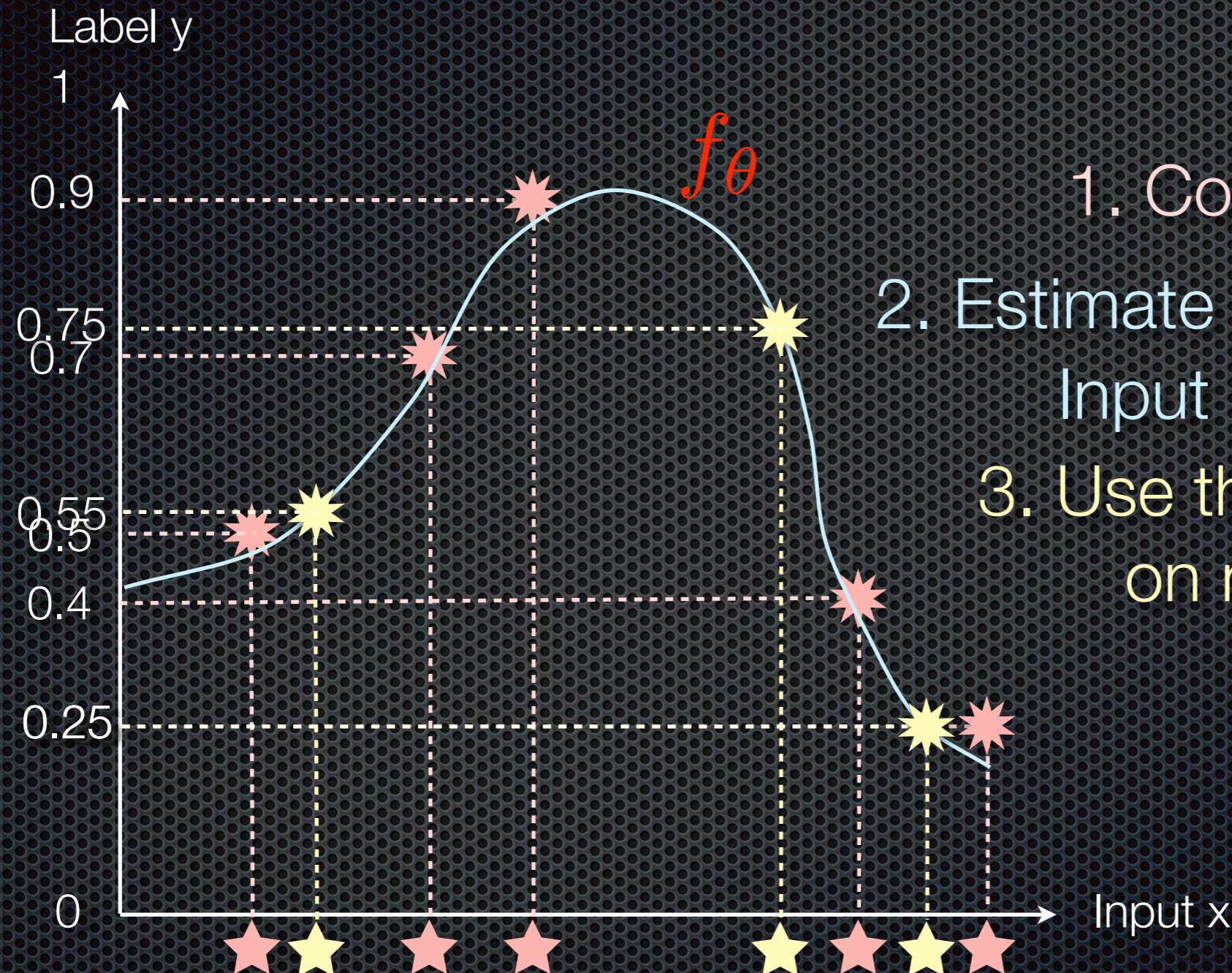
General framework of learning: three classic problems

- The three problems we consider
 - **classification:** $Z = (X, Y) \in \mathbb{R}^d \times \{1, \dots, N\}$ ($\mathbb{R}^d \times \{-1, 1\}$)
 - **regression:** $Z = (X, Y) \in \mathbb{R}^d \times \mathbb{R}$
 - **density estimation:** $Z \in \mathbb{R}^d$
- Collection of functions F (possible solutions), $f \in F$:
 - **classification:** $f : \mathbb{R}^d \rightarrow \{-1, 1\}$
 - **regression:** $f : \mathbb{R}^d \rightarrow \mathbb{R}$
 - **density estimation:** F contains density functions

Phases of learning

- **Training:** we learn a function f_θ optimizing its parameters θ so that it works well on the training set.
- **Prediction / testing :** then we can use the function on new points (test examples) that are not part of the training set.
 - ⇒ It is *not important* to learn perfectly (*memorise*) the training set.
 - ⇒ It is important to be able to *generalise* on new cases (points/examples).

Eg: regression 1D



1. Collect data
2. Estimate the function
Input \longleftrightarrow Label
3. Use that function
on new data

Supervised task

predict y from x

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

target (label) y

n examples

| x_1 | x_2 | x_3 | x_4 | 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

$f = f_\theta$ which minimizes a cost (loss).

loss function

$$L(f_\theta(x), y)$$

output/prediction $f_\theta(x)$

f_θ : parameters

| | | | | |
|-------|------|----|---|------|
| -0.12 | 0.42 | -1 | 1 | 0.22 |
|-------|------|----|---|------|

34

input x

target y

A learning algorithm is often the combination of the following:
(given explicitly or implicitly)

- ✓ the specification of a **family of functions** F
(often a *parametric family*)
- ✓ a **method to evaluate the quality** of a function $f \in F$ (typically using a **cost function (loss)** L eg: count how many points are misclassified by f)
- ✓ a **way to search the «best» function** $f \in F$ (ie, optimization of parameters to minimize the loss).

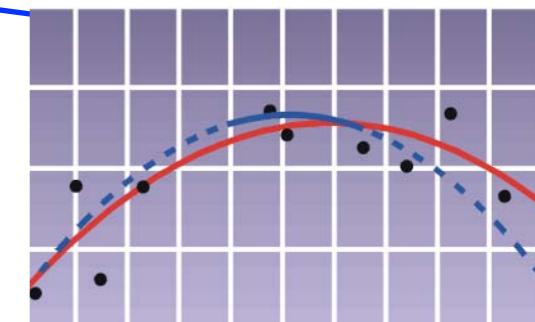
A learning algorithm is often the combination of the following:
(given explicitly or implicitly)

- ✓ the specification of a **family of functions F**
*(often a **parametric family**)*
- ✓ a method to evaluate the quality of a function $f \in F$ (typically using a cost function (loss) L eg: count how many points are misclassified by f)
- ✓ a way to search the «best» function $f \in F$ (ie, optimization of parameters to minimize the loss).

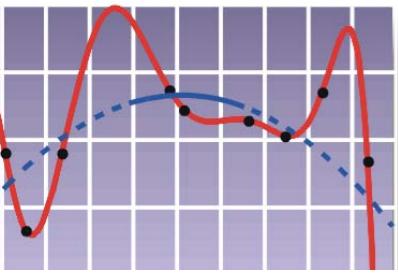
Examples of parametric function family

$F_{\text{polynomial } p}$
polynomial predictor (of degree p):

$$f_{\theta}(x) = b + a_1x + a_2x^2 + a_3x^3 + \dots + a_px^p$$



(in 1 dimension)



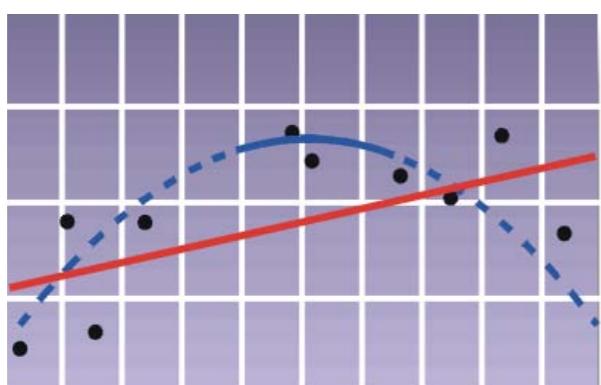
Linear (affine) predictor:
«linear regression»

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

F_{linear}

$$f_{\theta}(x) = wx + b \quad (\text{in 1 dimension})$$

$$f_{\theta}(x) = w^T x + b \quad (\text{in } d \text{ dimensions})$$



The simplest class

F_{const}
constant predictor: $f_{\theta}(x) = b$
or $\theta = \{b\}$

(Always predict the same value or class!)

There exist many different function families

- Constant
- Linear / affine
- polynomial
- Histograms “staircase functions”
- Weighted sum of kernels (eg Parzen or SVM with kernel)
- Decision trees
- Neural networks (composition of non-linear functions)
Each neural network "architecture" corresponds to a different family of functions..

Examples of parametric functions

- Linear and affine maps

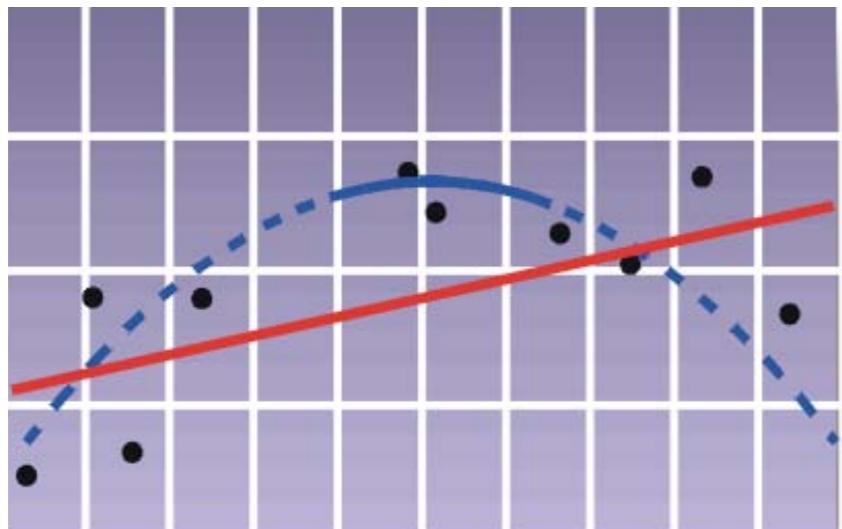
(e.g., for linear regression):

In 1 dimension:

$$f_{\theta}(x) = wx + b$$

Dimension d:

$$f_{\theta}(x) = w^T x + b$$

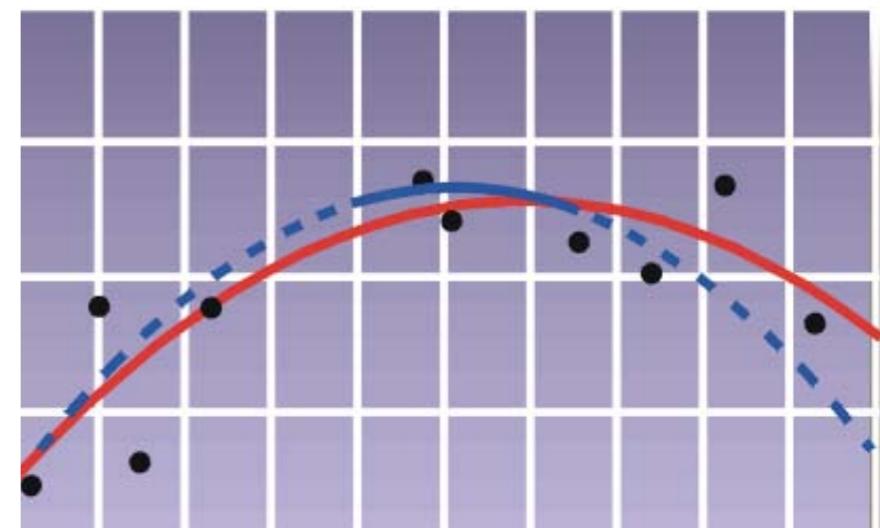


Parameters: $\theta = \{w \in \mathbb{R}^d, b \in \mathbb{R}\}$

- Polynomials of degree p
- (e.g., in polynomial regression):

In 1 dimension:

$$f(x) = b + a_1x + a_2x^2 + a_3x^3 + \dots + a_p x^p$$



A learning algorithm is often the combination of the following:
(given explicitly or implicitly)

- ✓ the specification of a family of functions F
(often a *parametric* family)
- ✓ a **method to evaluate the quality** of a function $f \in F$ (typically using a **cost function (loss)** L eg: count how many points are misclassified by f)
- ✓ a way to search the «best» function $f \in F$ (ie, optimization of parameters to minimize the loss).

Evaluate the quality of a predictor $f(x)$

The performance of one of a prediction function $f(x)$ is often evaluated with **many** different evaluation metrics:

- Evaluation of the **true quantity of interest** (\$ saved/gained, #lives saved, ...) when using the predictor within a **complex complete system**.
- "Standard" performance measures specific to a particular field (e.g., BLEU score: Bilingual Evaluation Understudy for translation)
- Classification error rate for a classifier (or precision and recall, or F-score, ...).
- The **cost** that is actually optimized by the machine learning algorithm.

Simple cost (loss) functions for performance evaluation

(« *cost function* » or « *loss function* »)

- **Classification task**

classification error

$$f : \mathbb{R}^d \rightarrow \{0, \dots, m-1\}$$

$$L(f(x), y) = I_{\{f(x) \neq y\}}$$

- **Regression task:**

quadratic error:

$$f : \mathbb{R}^d \rightarrow \mathbb{R}$$

$$L(f(x), y) = (f(x) - y)^2$$

- **Density estimation task**

negative log likelihood:

$$f : \mathbb{R}^d \rightarrow \mathbb{R}^+$$

a probability density (or
mass) function

$$L(f(x)) = -\log f(x)$$

Performance evaluation is often the average of a cost (loss)
function on a set of test data

Surrogate losses

- For a classification task:
classification error:

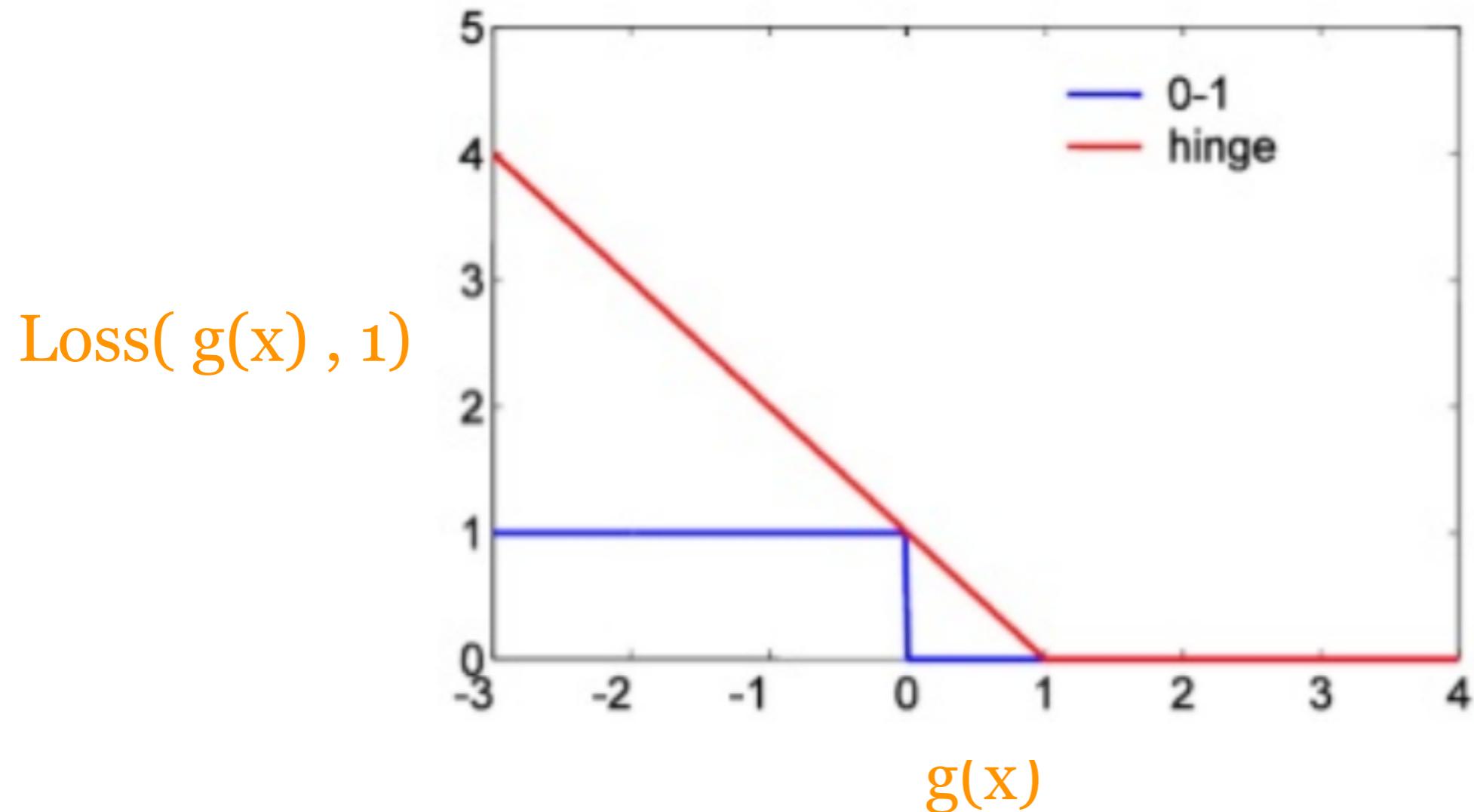
$$f : \mathbb{R}^d \rightarrow \{0, \dots, m-1\}$$

$$L(f(x), y) = I_{\{f(x) \neq y\}}$$

Problem: difficult to directly optimize the classification error rate
(gradient 0 everywhere. NP-hard with a linear classifier) **Use a surrogate loss!**

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

Hinge Loss vs. 0-1 Loss



Expected risk v.s. Empirical risk

“Risk” = “average” loss

Example (\mathbf{x}, \mathbf{y}) drawn i.i.d. from unknown distribution $p(\mathbf{x}, \mathbf{y})$
(from nature or an industrial process)

- Generalization error = Expected risk
(or just «Risk» = expected loss)
«*the loss/error we will get on average over infinitely many future examples from this unknown distribution*»

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

- Empirical risk = the average loss over a finite dataset D (drawn from p)
«*the average error on the examples in this dataset*»

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

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

Estimating the expected risk

- We **cannot calculate it exactly**... because $p(x,y)$ is unknown!
- We **can estimate it approximately**:
If we did not use D at all to choose f the empirical risk is an **unbiased estimate** (though noisy) of the expected risk:

$$\text{Expected Risk} \quad R(f) \approx \hat{R}(f, D) \quad (\text{Empirical Risk})$$

because $\mathbb{E}_{p(\mathbf{x},\mathbf{y})}[L(f(\mathbf{x}), \mathbf{y})] \approx \frac{1}{|D|} \sum_{(\mathbf{x},\mathbf{y}) \in D} L(f(\mathbf{x}), \mathbf{y})$

More precisely unbiased estimator means that:

$$\mathbb{E}_{D \sim p}[\hat{R}(f, D)] = R(f)$$

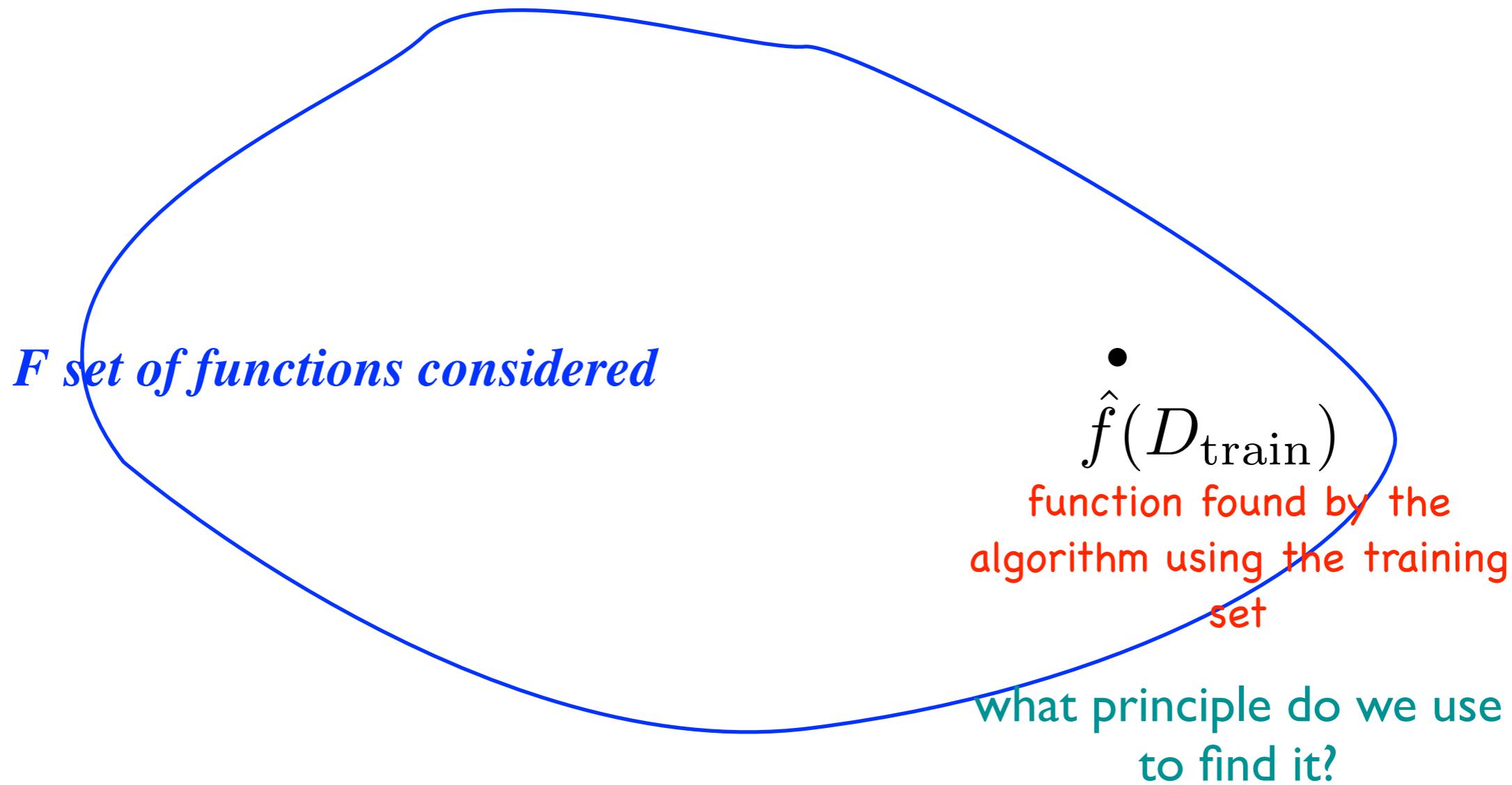
which is also reflected in $\lim_{|D| \rightarrow \infty} \hat{R}(f, D) = R(f)$

A learning algorithm is often the combination of the following:
(given explicitly or implicitly)

- ✓ the specification of a family of functions F
(often a *parametric* family)
- ✓ a method to evaluate the quality of a function $f \in F$ (typically using a cost function (loss) L eg: count how many points are misclassified by f)
- ✓ a **way to search the «best» function $f \in F$ (ie, optimization of parameters to minimize the loss).**

Learning is choosing a function from a set of functions

Set of all possible functions “in the universe”



Empirical Risk Minimization (ERM)

- We **would like** to find a predictor that minimizes the error of generalization (expected risk)
- But **we can not even calculate it!**
- Instead: **Principle of ERM minimization**
«Find the predictor that minimizes the average loss on a training set»

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

This is the training phase

Estimate the generalization error of the predictor $\hat{f}(D_{\text{train}})$

The problem:

- ERM minimization principle: we train a model (we adapt its parameters) so that it makes a minimum of errors on the training set.
$$\hat{f}(D_{\text{train}}) = \operatorname{argmin}_{f \in F} \hat{R}(f, D_{\text{train}})$$
- BUT what really interests us is to generalize on new examples.
- Since the parameters of the model are chosen, specialized, to minimize training error, this may underestimates the generalization error.
- So training error is not a good estimate for generalization error (there is an estimation bias).

Estimate the generalization error of the predictor $\hat{f}(D_{\text{train}})$

- ▶ $\hat{R}(f, D)$ is a good estimator for $R(f)$ under the condition:
 - D was not used to find / choose f otherwise biased \Rightarrow we cannot use the training set!
 - D is sufficiently big (otherwise estimate is noisy); drawn from p

→ We must keep a separate test set $D_{\text{test}} \neq D_{\text{train}}$ to correctly estimate the 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 average error on the test set
error (never used for training)

This is the test phase

Generalization error estimation: Simple validation

All of the labeled data we have:

$D =$

(x_1, y_1)

(x_2, y_2)

⋮

(x_N, y_N)

*(it may be necessary to first mix the rows
of data)*

Training set
(size n)

Test set
(size m)

We split our data
in two

We **train the model** to
minimize the error on the
training set

Empirical Risk

The **generalization performance**
is evaluated by measuring errors on the
test set that was never viewed during
training.
(performance measure "out of sample").

If we do not have enough data: cross-validation (in k blocks)

$D =$

$$(x_1, y_1)$$

$$(x_2, y_2)$$

:

$$(x_N, y_N)$$



Simple idea: the training / test procedure is repeated several times by dividing the data set differently.

| <i>Training on</i> | <i>Test error calculation on</i> |
|------------------------------|--------------------------------------|
| $D \setminus \text{Bloc 1}$ | Bloc 1 |
| $D \setminus \text{Bloc 2}$ | Bloc 2 |
| ... | ... |
| $D \setminus \text{Bloc } k$ | Bloc k |

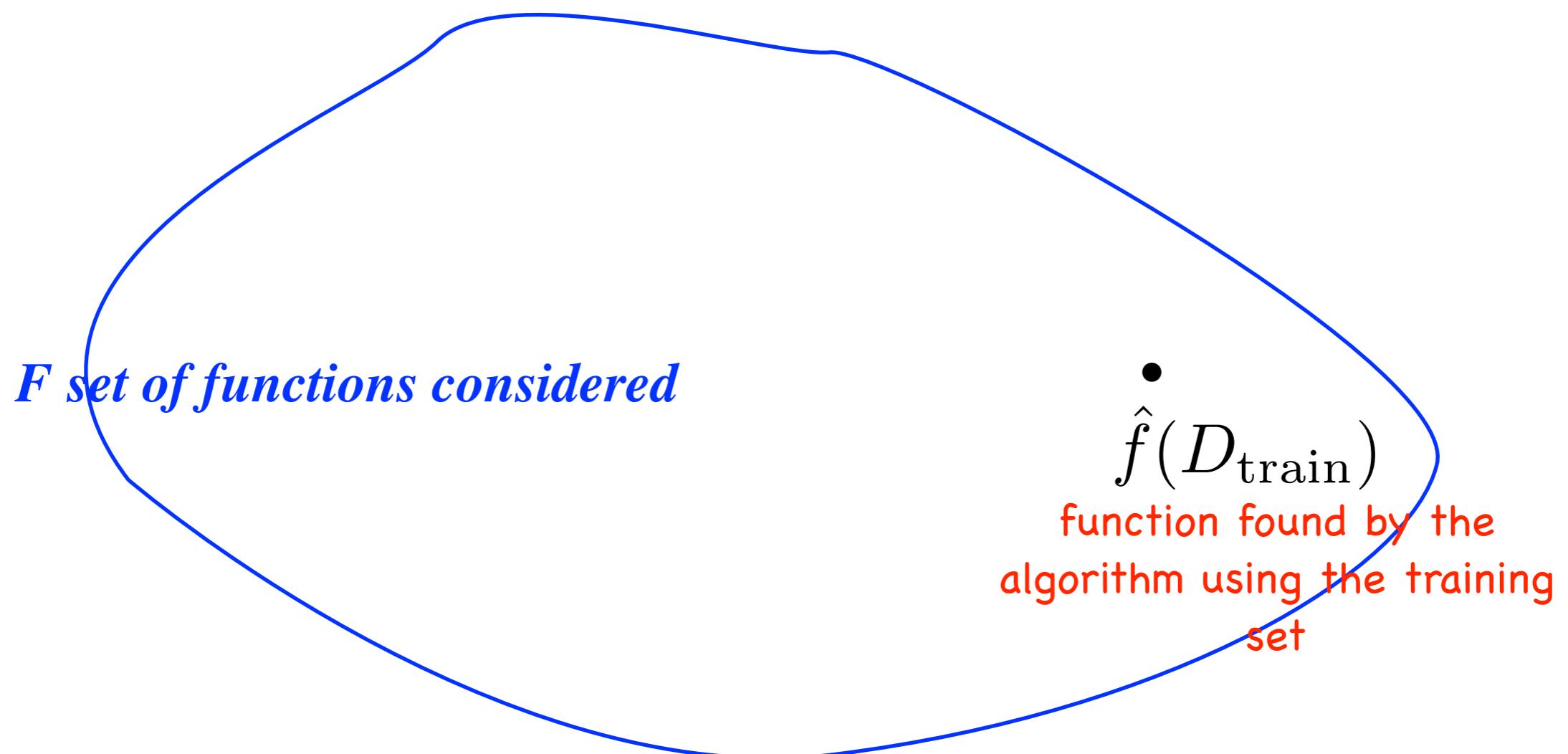
Our estimate of the generalization error of the algorithm on this problem is deduced from the sum of the errors obtained on all the blocks.

We will this method ***k-fold cross validation***.
The case $k=N$ is called ***leave-one-out*** or ***jackknife***.

Notions of capacity
of overfitting
or underfitting

Learning is choosing a function from a set of functions

Set of all possible functions “in the universe”



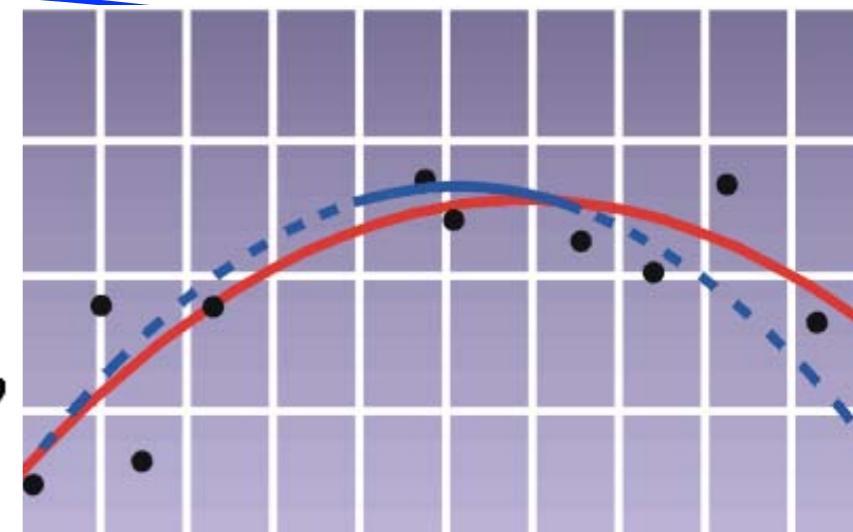
Concept of capacity or complexity of model

- Corresponds to the "size" or "richness" of the function set considered (\pm flexible functions)
- Often related to the **number of free parameters** (to learn from the training set) Usually **more parameters means a bigger capacity.**
- But not always ... it is rather the number of effective degrees of freedom of the function (effective capacity).
- There are formal measures for some restricted frameworks. Ex:VC-dimension (Vapnik-Chervonenkis).

Examples of parametric function family

$F_{\text{polynomial } p}$
polynomial predictor (of degree p):

$$f_{\theta}(x) = b + a_1x + a_2x^2 + a_3x^3 + \dots + a_px^p$$



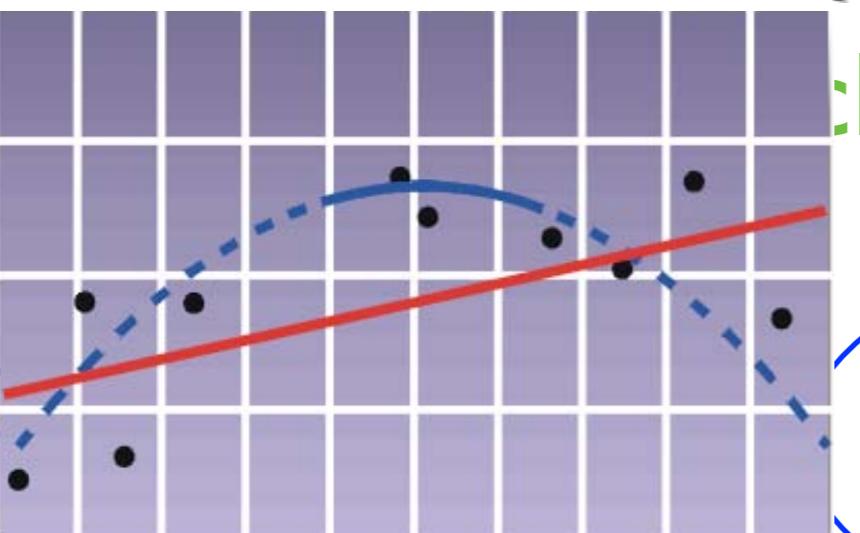
F_{linear}

Linear (affine) predictor:
«linear regression»

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

$$f_{\theta}(x) = wx + b \quad (\text{in 1 dimension})$$

$$f_{\theta}(x) = w^T x + b \quad (\text{in } d \text{ dimensions})$$



: class

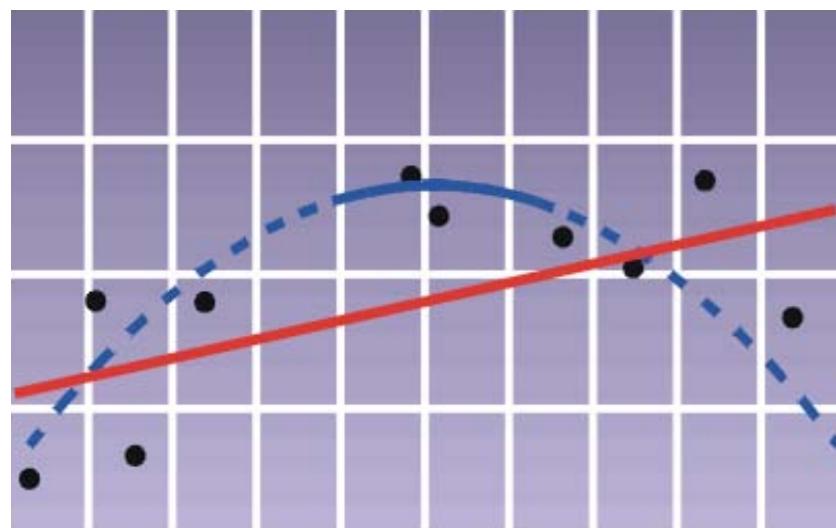
F_{const}
constant predictor: $f_{\theta}(x) = b$
or $\theta = \{b\}$

(Always predict the same value or class!)

Capacity control

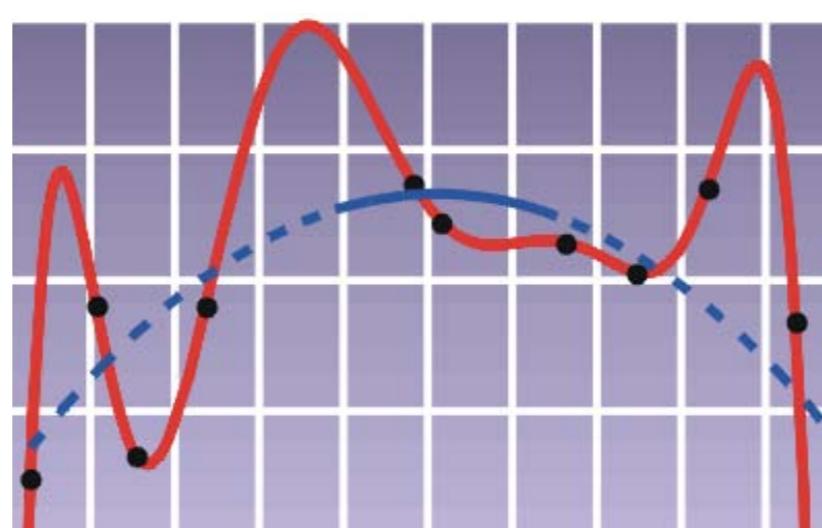
- The choice of the function set and the hyper-parameters of the algorithms make it possible to control the capacity of the model.
- It is essential to have a good generalization.

Ex: Regression 1D



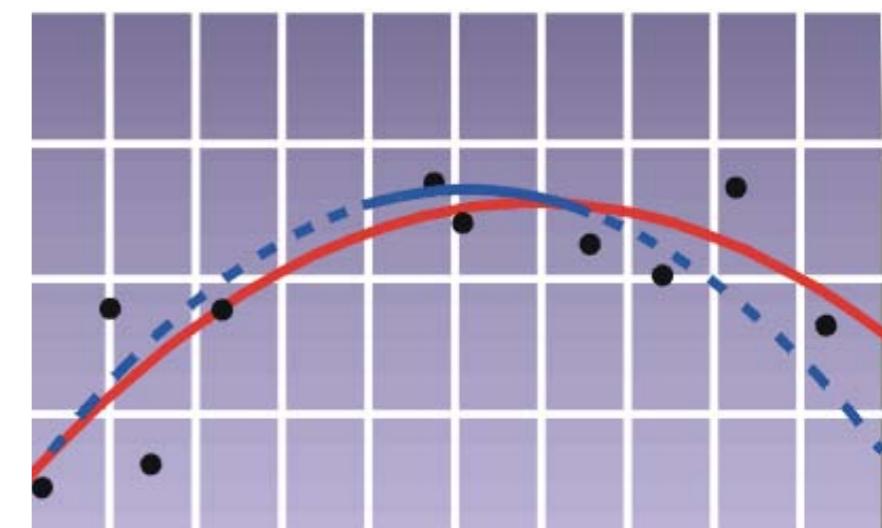
low capacity
→ underfitting

BIAS



too high capacity
→ overfitting

VARIANCE



optimal capacity
→ good generalization

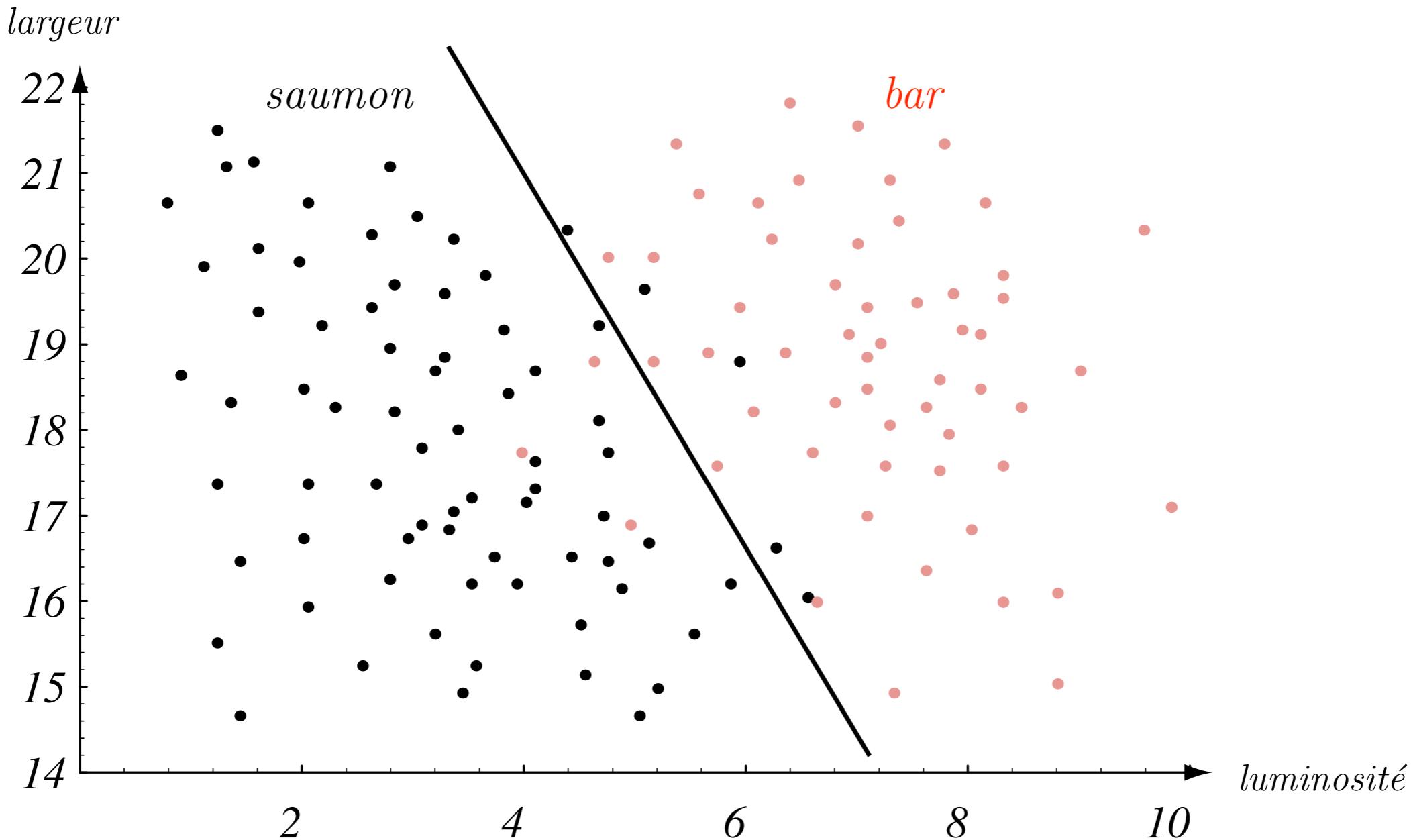
VARIANCE

the performance on the training set is not a good estimate of the generalization

Ex: classification 2D

*Linear classifier
(low capacity)*

- Function family poor (too little expressivity)
- => Capacity too small for this problem (relative to the amount of data)
- => Underfitting



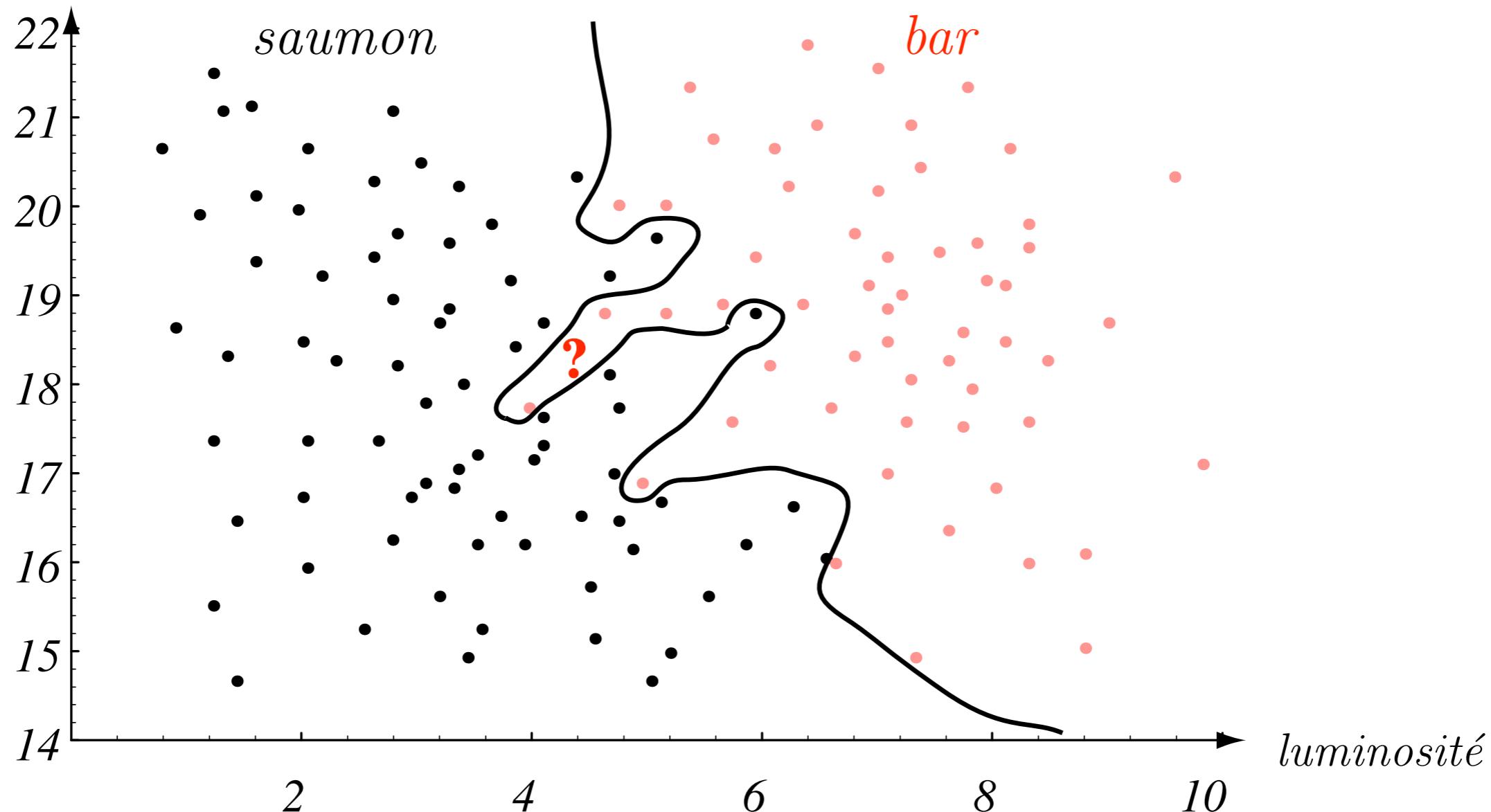
Number of training errors: 3+5 = 8

Ex: classification 2D

High capacity classifier

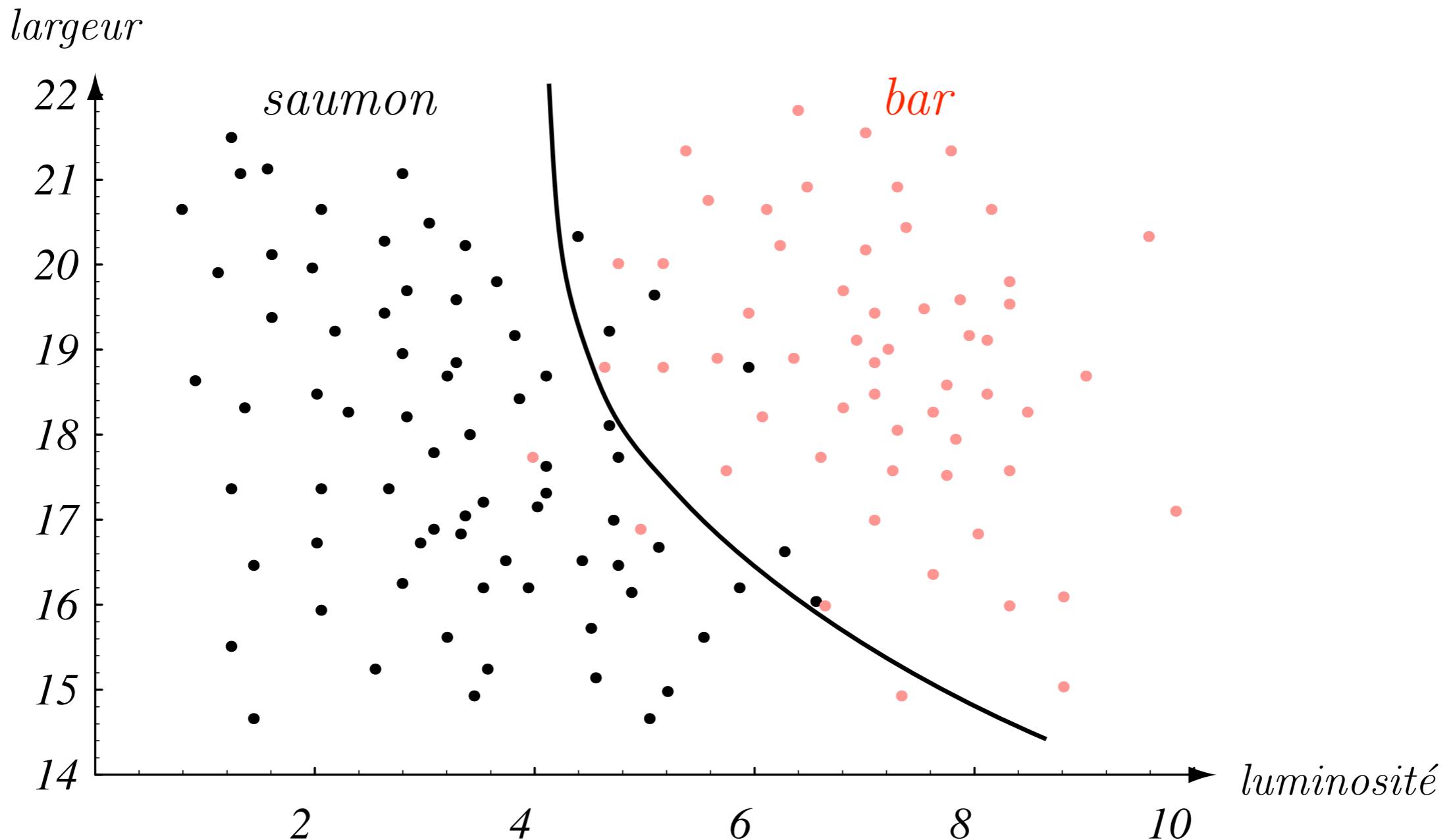
- Function set too rich (too expressive)
- = Capacity too high for this problem (relative to the amount of data)
- => Overfitting

largeur



Number of training errors: 0

- Optimum capacity for this problem (relative to the amount of data)
- => Better generalization (on future test points)



Occam's razor

Occam's Razor



- through the ages...

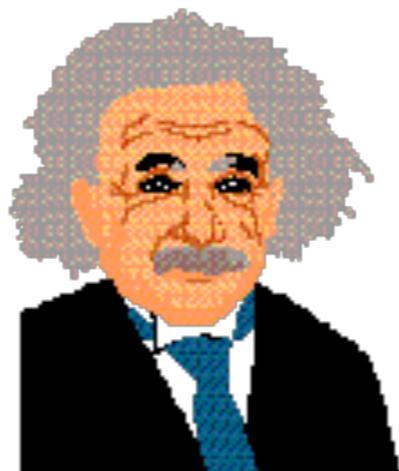
*Pluralitas non
est ponenda sine
necessitate.*

(*Plurality should not be
posited without necessity.*)

- William of Ockham

Everything should be
made as simple as
possible, but not
simpler.

- Albert Einstein



K
eep
I
t
S
imple,
S
tupid !



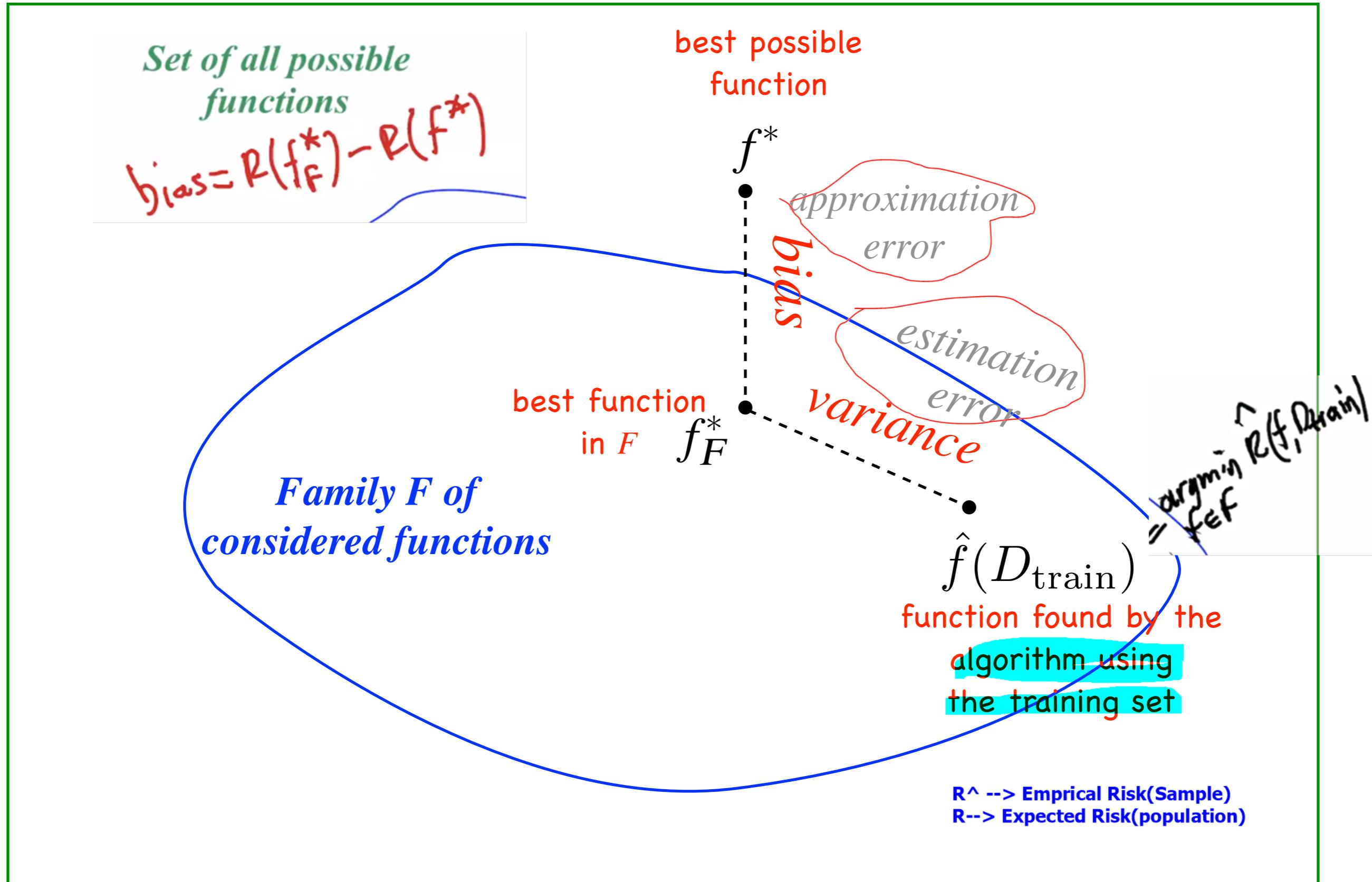
William of Ockham

In learning:

Choose the simplest possible
model that learns the data well.

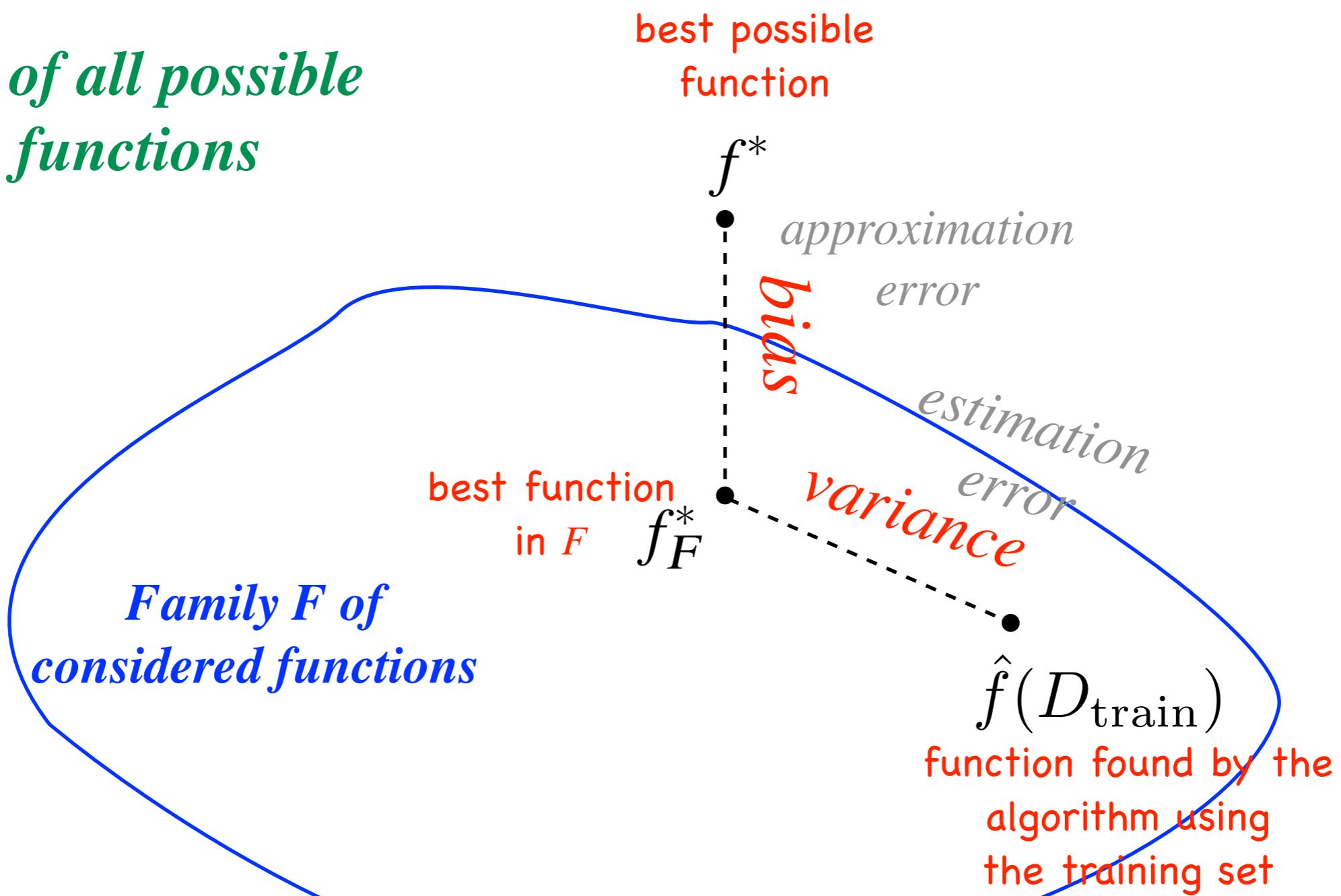
Issue: the more the function
family is rich (complex) the
more the training data are well
"learned" ...

Decomposition of the generalization error + variance



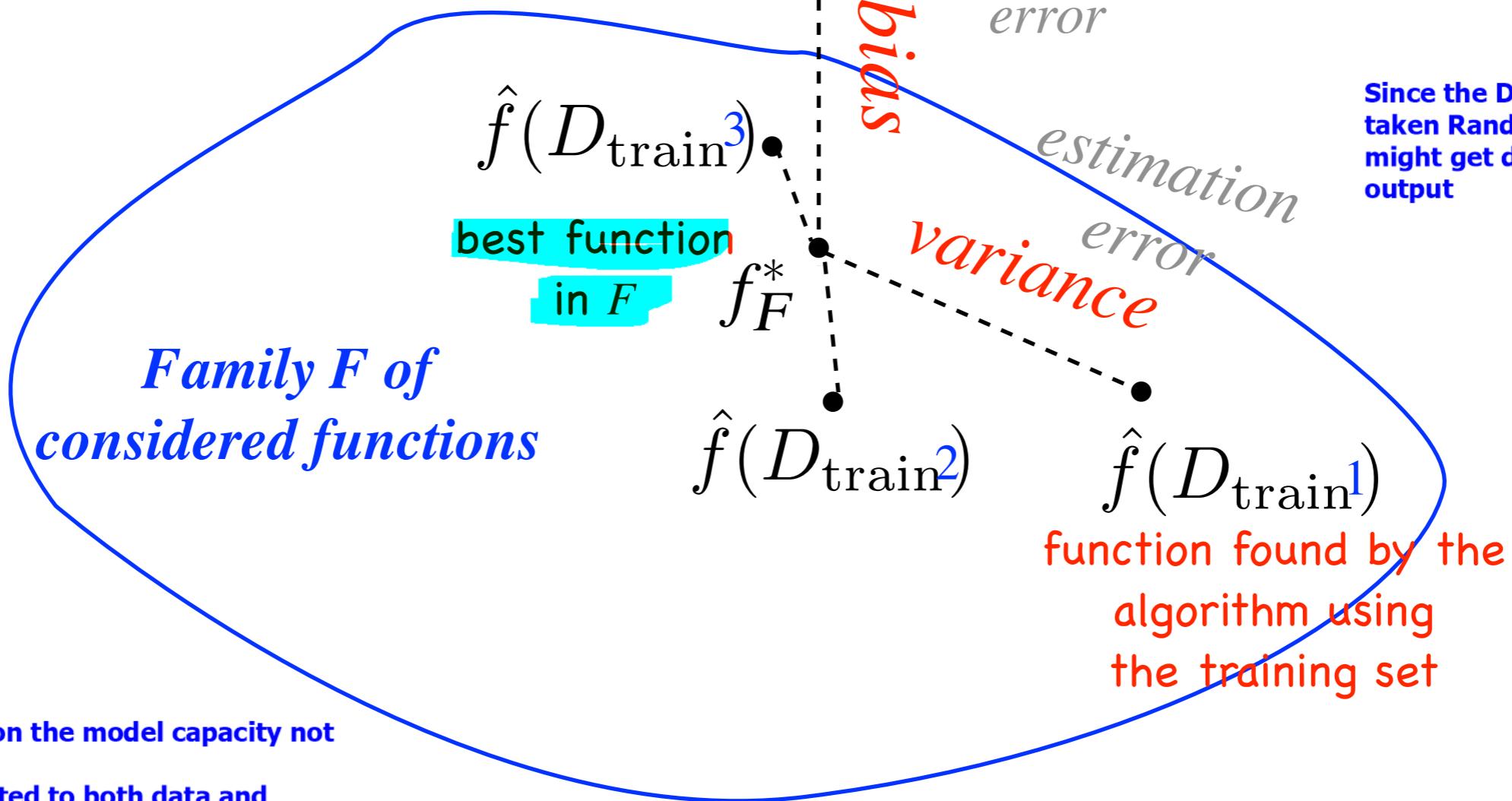
What is the source of variance?

Set of all possible functions



What is the source of variance?

Set of all possible functions



What is the right capacity? The **bias-variance tradeoff**

- If we choose a rich set F : capacity ↑
 ⇒ bias ↓ *but* variance ↑.
richer $F \rightarrow$ higher degree of polynomial
- If we choose a more limited set F : capacity ↓
 ⇒ variance ↓ *but* bias ↑.
- The optimal compromise... will depend on the problem and the number of training examples n
- Bigger n ⇒ variance ↓
Variance only, depends on the dataset. If the dataset grows the variance will decrease.
We can then afford to increase the capacity (to decrease the bias)
⇒ we can afford more complex models.
- The best regularizer is more data :)

Model selection
Selection of hyper-
parameters

in practice

Model selection

We will consider:

- **Several** learning algos.
- For each algo, **several** choices of values of its hyper-parameters (eg nb of neighbors k in k -NN, nb of neurons in a neural network, ...)

For each case, we will train a model (optimize its parameters on a training set)
then evaluate its out-of-sample generalization performance (on a validation set)

Model selection (and

$D =$

$$(x_1, y_1)$$

$$(x_2, y_2)$$

— — —

⋮

— — —

$$(x_N, y_N)$$

Training set
 D_{train}

Validation set
 D_{valid}

Test set
 D_{test}

Make sure the examples in D are in random order. Divide D in 3: D_{train} D_{valid} D_{test}

Model selection meta-algorithm

For each model considered model (algo) A :

For each configuration of hyper-parameters λ :

- train A with hyperparams λ on D_{train}

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

- evaluate the predictor on D_{valid}

(with an appropriate evaluation metric)

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

Identify A^*, λ^* that give the best e_{A_λ}
And return $f^* = f_{A_{\lambda^*}}$

Or re-train and return

$$f^* = A_{\lambda^*}^*(D_{\text{train}} \cup D_{\text{valid}})$$

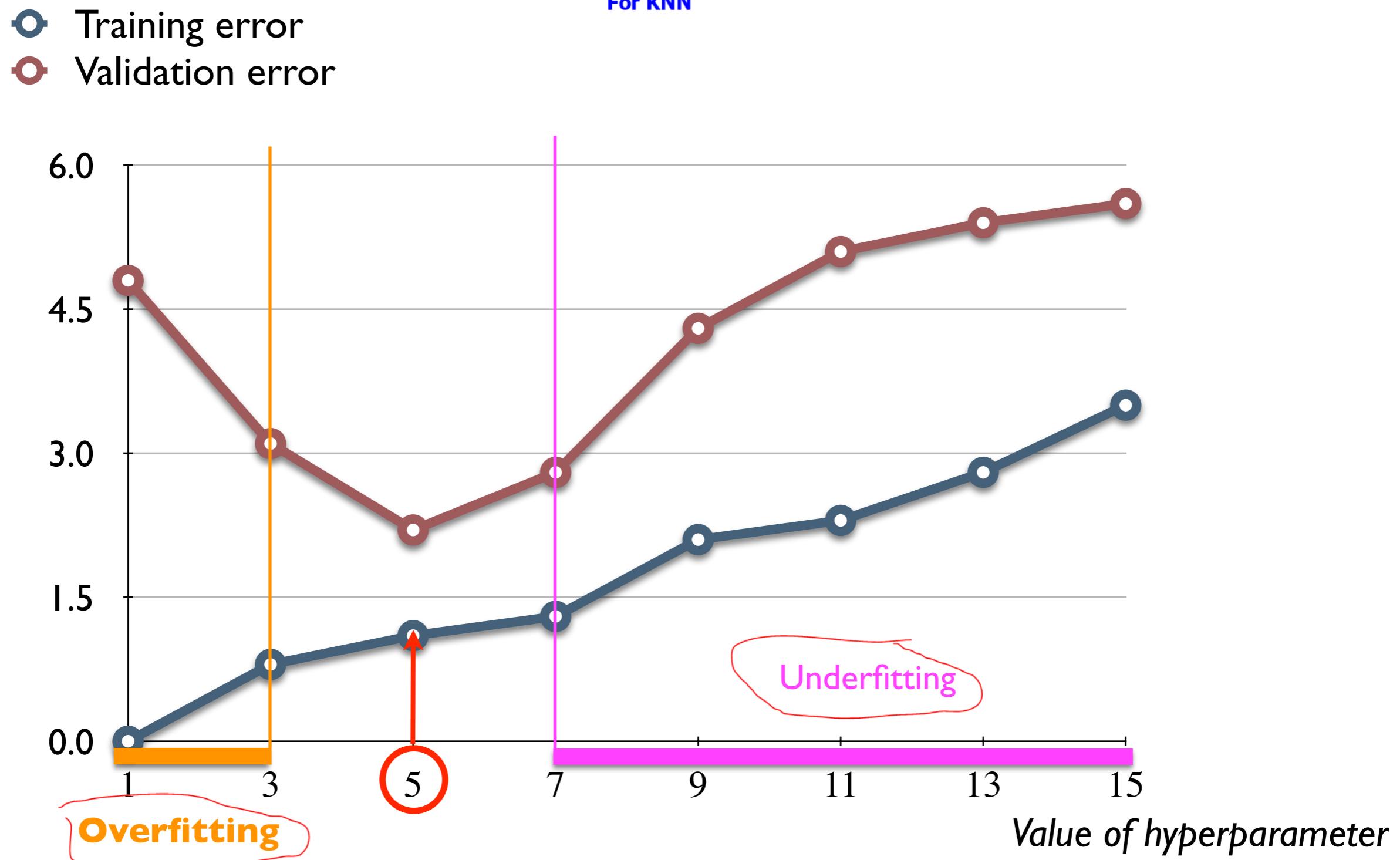
based on validation set,
to make it unbiased

Finally, calculate an unbiased estimate of the generalization performance of f^* using D_{test}

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

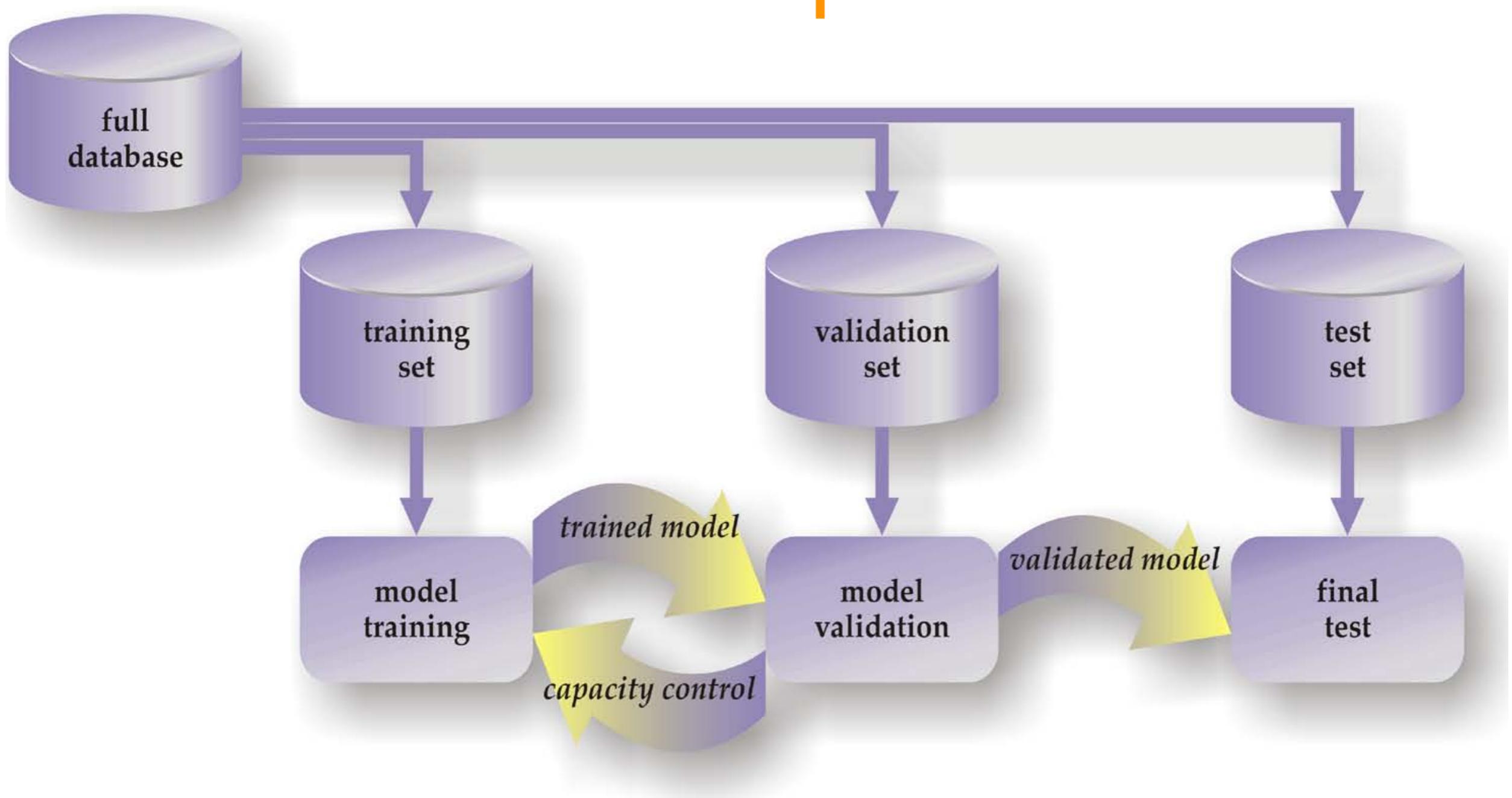
D_{test} must never have been used during training or model selection to select, learn, or adjust anything

Learning curves



The value of the hyper-parameter giving the minimum error on the validation set is 5 (while it is 1 for the training set)

Model Selection: How to proceed



The final performance evaluation must be based on data that has not been used for training or selection of models or capacity.

Parameters v.s. hyperparameters?

- Parameters are optimized on the training set (learned by the algorithm).
- Hyper-parameters are fixed by the user generally before training.
We optimize them on a validation set (external loop) Generally they control capacity
- What would happen if we chose both hyper-parameters and optimal parameters on the training set?
 - ➡ Would tend to choose the highest possible capacity (to make the least mistake on the training set)
 - ➡ Overfitting
 - ➡ Bad generalization

Parameters? Hyper-parameters?

Eg: for histogram methods

- Which are the hyper-parameters?
- Which are the parameters?

Parameters? Hyper-parameters?

Eg: for histogram methods

- Which are the hyper-parameters?
 - ▶ Number of bins (subdivisions/cases)
- Which are the parameters?
 - ▶ The count (how many examples) in each bin

Parameters?

Hyper-parameters?

Eg: For k-NN

- Which are the hyper-parameters?
- Which are the parameters?

Parameters?

Hyper-parameters?

Eg: For k-NN

- Which are the hyper-parameters?
 - ▶ Neighborhood size: k
- Which are the parameters?
 - ▶ We memorize the training data set

Parameters?

Hyper-parameters?

Eg: for Parzen windows

- Which are the hyper-parameters?
- Which are the parameters?

Parameters?

Hyper-parameters?

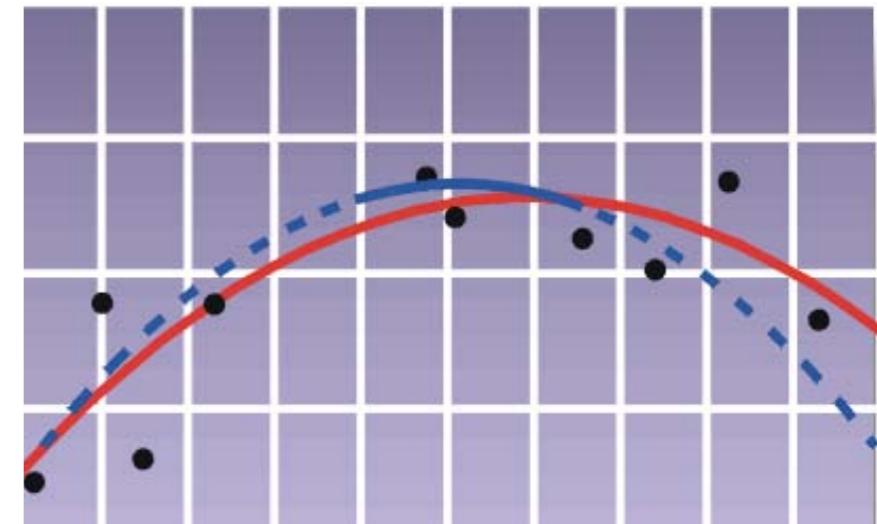
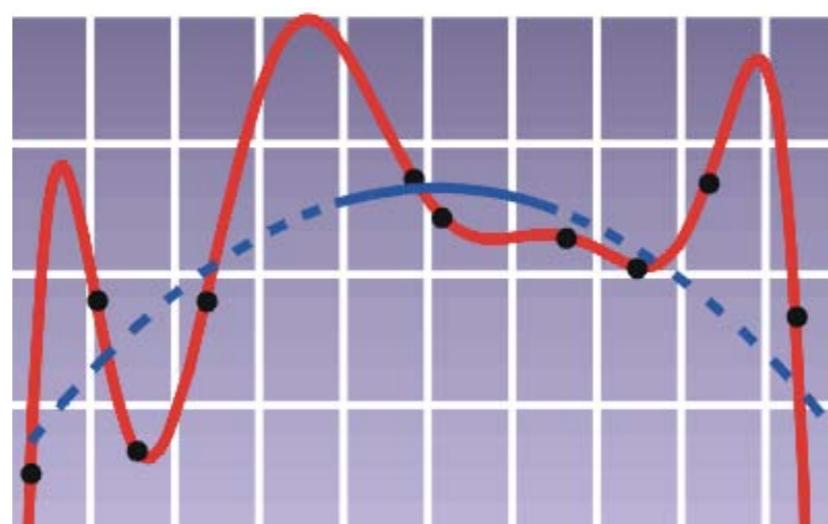
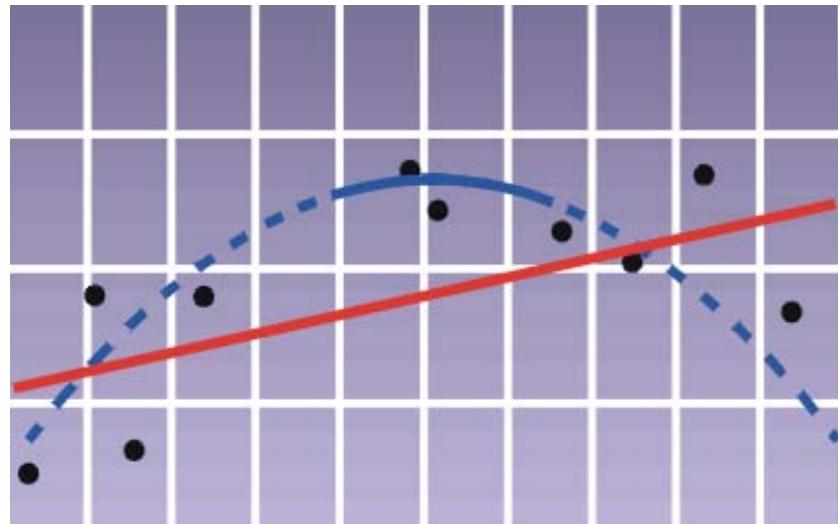
Eg: for Parzen windows

- Which are the hyper-parameters?
 - ▶ Neighborhood Size: Kernel width
- Which are the parameters?
 - ▶ We memorize the training data set

Parameters? Hyper-parameters?

Eg: For polynomial regression

- Which are the hyper-parameters?
 - ▶ degree of the polynomial
- Which are the parameters?
 - ▶ learned polynomial coefficients



Hyper-parameters?

There may be several hyper-parameters

- Controlling the "size" of the function family
- Inducing a more-or-less restrictive preference among the functions of a family
- Choosing one of several families
- Controlling certain aspects of parameter optimization (which will be performed on the training set).

Regularization

- To more finely control overfitting, we can induce a **soft preference among the functions in F**

$$\Omega : F \rightarrow \mathbb{R}$$

a small $\Omega(f)$, indicates preference for function f .

- Rather than minimize the empirical risk:

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

- We will minimize regularized empirical risk

$$\hat{f}(D_{\text{train}}, \lambda) = \operatorname{argmin}_{f \in F} \hat{R}(f, D_{\text{train}}) + \boxed{\lambda \Omega(f)}$$

If the degree is high, this term will be high and that will add to the cost fun. If it is linear fn then reg val is small. ie putting penalty based on the deg. of polynomial

λ is a scalar that controls the power of regularization:

the tradeoff between empirical risk and the “a priori” preference between functions in F .

- Eg: for $F=\text{polynomials}$ we can have $\Omega(\text{polynomial}_p)=p$