Université de Montréal
Département d'informatique et de recherche opérationnelle

IFT3395/6390
Fondements de l'apprentissage machine

Cadre général de l'apprentissage
Évaluation de la performance de généralisation.
Sélection de modèle
Courbes d'apprentissage.

Professeur: Pascal Vincent

LISA Laboratoire d'Informatique des Systèmes d'Apprentissage

Approche probabiliste de l'apprentissage

- On suppose que les données sont générées par un processus inconnu.
- X,Y est vu comme une paire de variables aléatoires, distribuées selon une loi de probabilité inconnue P(X,Y).
- X (une variable vectorielle) est elle-même vue comme un ensemble de variables aléatoires scalaires.

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

Cadre général de l'apprentissage
l'ensemble de données

- Données
  - $D_n = (Z_1, Z_2, \ldots, Z_n)$ générées par la "nature"
- IID: "independent and identically distributed"
  - tirés de la même distribution INCONNUE $p(Z)$
  - de manière indépendante

Cadre général de l'apprentissage
les 3 problèmes classiques

- Les trois problèmes considérés
  - classification: $Z = (X, Y) \in \mathbb{R}^d \times \{1, \ldots, N\}$ ($\mathbb{R}^d \times \{-1, 1\}$)
  - régression: $Z = (X, Y) \in \mathbb{R}^d \times \mathbb{R}$
  - estimation de densité: $Z \in \mathbb{R}^d$

- Ensemble de fonctions F (solutions possibles), $f \in F$:
  - classification: $f : \mathbb{R}^d \to \{-1, 1\}$
  - régression: $f : \mathbb{R}^d \to \mathbb{R}$
  - estimation de densité: F contient des fonctions de densité

Phases de l'apprentissage

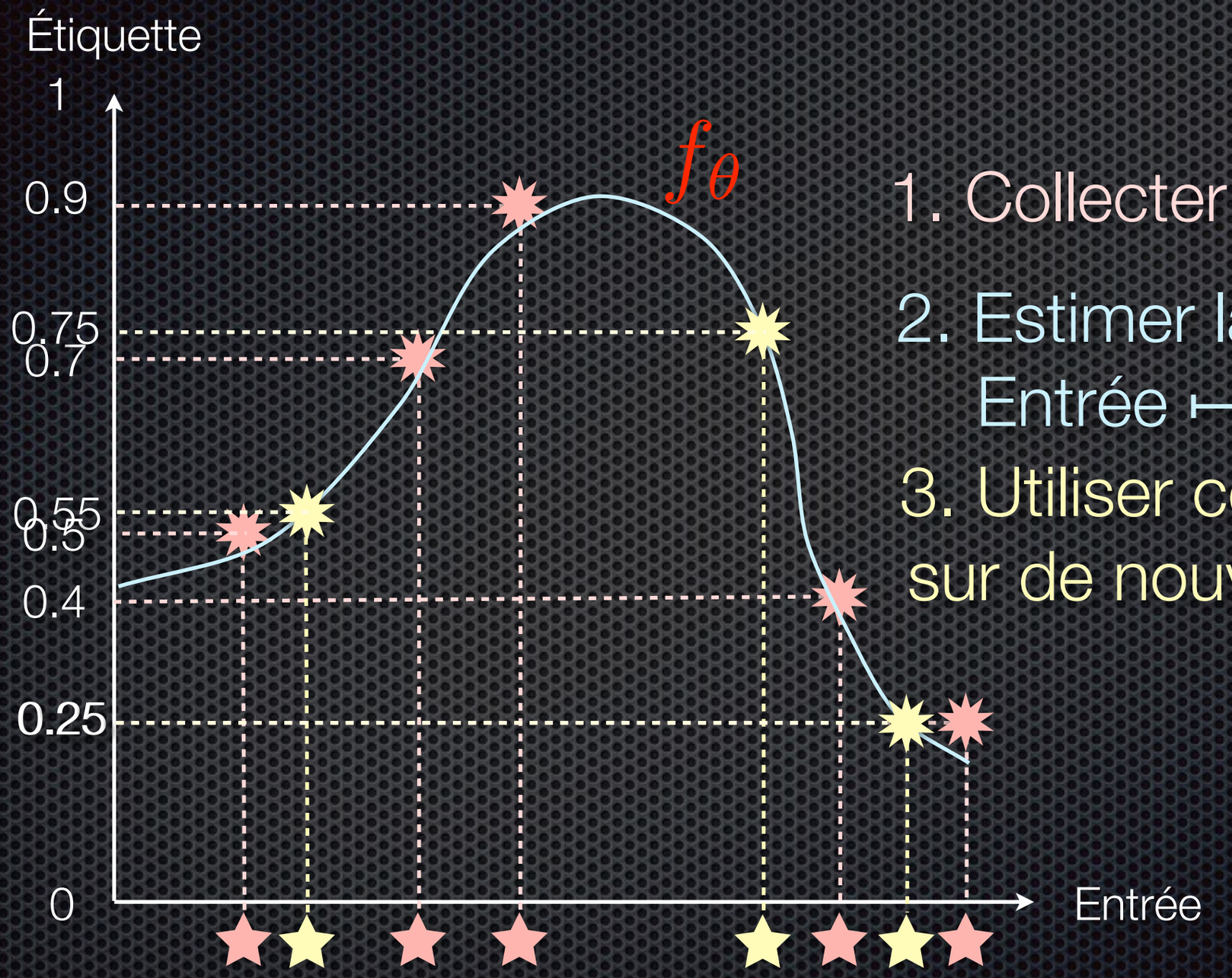
- Entraînement: on apprend une fonction f_θ optimisant ses paramètres θ pour qu'elle fonctionne bien sur l'ensemble d'entraînement.

- Prédiction / utilisation / test : ensuite on peut l'utiliser sur de nouveaux points (de test) qui ne faisaient pas partie de l'ensemble d'entraînement.

⇨ L'important n'est *pas* d'apprendre parfaitement *(mémoriser)* l'ensemble d'entraînement.

⇨ L'important c'est d'être capable de *généraliser* à de nouveaux cas.

Ex: régression 1D



1. Collecter des données
2. Estimer la fonction
   Entrée $\longmapsto$ Étiquette
3. Utiliser cette fonction sur de nouvelles données

Tâche supervisée

prédire y à partir de x

entrée $\mathbf{x} \in \mathbb{R}^d$

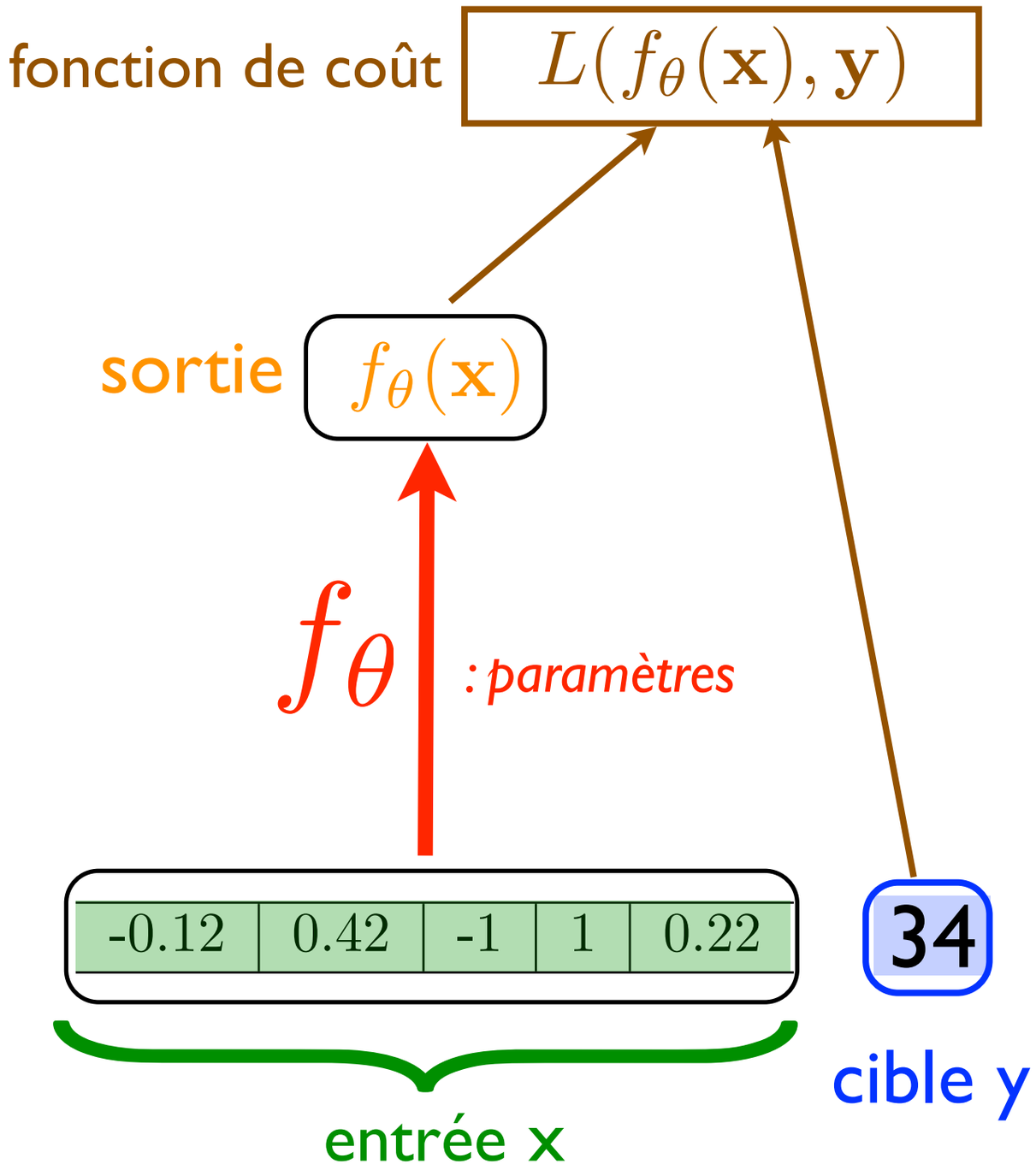
cible (étiquette) y

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
...	...	...	...	...	...

Ensemble d'entraînement D_n

Apprendre une fonction $f = f_\theta$ qui minimize un coût (perte).



Un algorithme d'apprentissage correspond souvent à la combinaison des éléments suivants:

(donnés explicitement ou implicitement)

✓ la spécification d'une famille de fonctions F
(souvent une famille paramétrée)

✓ une manière d'évaluer la qualité d'une fonction $f \in F$ (typiquement utilisant une fonction de coût (perte) L ex: mesure à quel point f prédit mal la classe)

✓ une manière de chercher la «meilleure» fonction $f \in F$ (ex: optimisation des paramètres pour minimiser la perte).

Un algorithme d'apprentissage correspond souvent à la combinaison des éléments suivants:

(donnés explicitement ou implicitement)

✓ la spécification d'une famille de fonctions F
(souvent une famille paramétrée)

✓ une manière d'évaluer la qualité d'une fonction $f \in F$ (typiquement utilisant une fonction de coût (perte) L ex: mesure à quel point f prédit mal la classe)

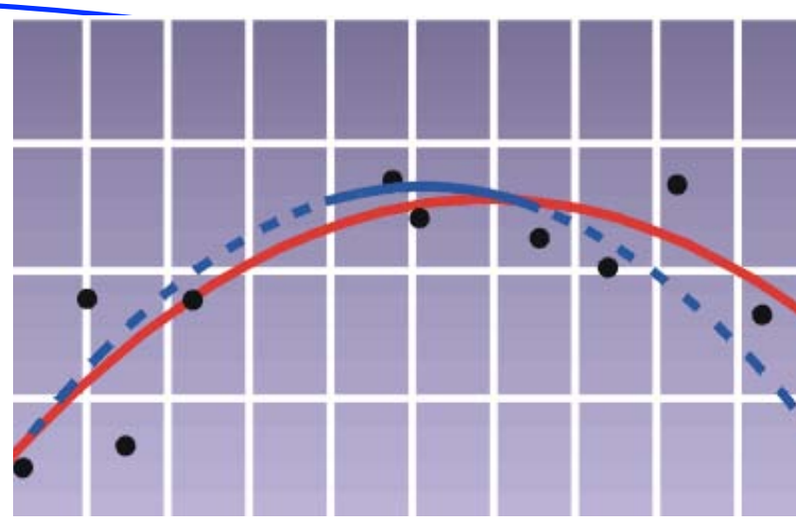
✓ une manière de chercher la «meilleure» fonction $f \in F$ (ex: optimisation des paramètres pour minimiser la perte).

Ex de familles de fonctions paramétrées

$F_{polynomial\ p}$

Prédicteur polynomial (de degré p):

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

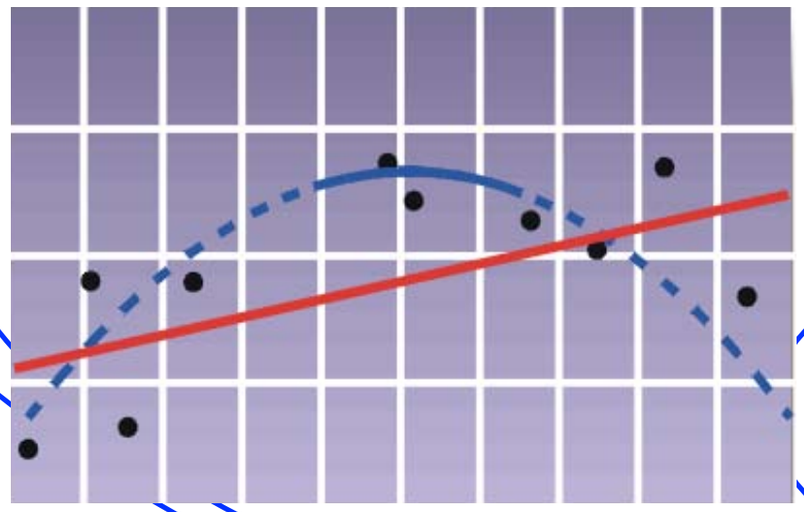


F_{linear}

Prédicteur linéaire (affine): $f_\theta(x) = wx + b$ (en 1 dimension)

(«régression linéaire») $f_\theta(x) = w^T x + b$ (en d dimensions)

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



F_{const}

Prédicteur constant: $f_\theta(x)=b$

où $\theta=\{b\}$

(Prédit toujours la même valeur ou classe!)

Il existe de nombreuses familles de fonctions…

- Constante
- Linéaire /affine
- Polynômiale
- Fonctions en escalier par blocs (histogrammes)
- Somme pondérée de noyaux (ex: Parzen ou SVM à noyau)
- Arbres de décision
- Réseaux de neurones (composition de fonctions non linéaires)
  Chaque « architecture » de réseau de neurones correspond à une famille de fonctions différente.

Un algorithme d'apprentissage correspond souvent à la combinaison des éléments suivants:

(donnés explicitement ou implicitement)

✓ la spécification d'une famille de fonctions F (souvent une famille paramétrée)

✓ une manière d'évaluer la qualité d'une fonction $f \in F$ (typiquement utilisant une fonction de coût (perte) L ex: mesure à quel point f prédit mal la classe)

✓ une manière de chercher la «meilleure» fonction $f \in F$ (ex: optimisation des paramètres pour minimiser la perte).

Evalualuer la qualité d'un prédicteur $f(\mathrm{x})$

La performance d'un d'une fonction de prédiction $f(\mathrm{x})$ est souvent évaluée avec **plusieurs** mesures d'évaluation différentes:

- Évaluation de la vraie quantité d'intérêt ($ gagnés, #vies sauvées, ...) quand on utilise le prédicteur au sein d'un système complet complexe.
- Mesures de performance «standard» spécifiques à un domaine particulier (e.g. score BLEU *Bilingual Evaluation Understudy* pour la traduction)
- Taux d'erreur de classification pour un classifieur (ou *précision* et *rappel*, ou F-score, ...).
- Le coût qui est en réalité optimisé par l'algorithme d'apprentissage machine.

Fonctions de coût (perte) usuelles simples pour évaluation de performance

(« cost function » ou « loss function »)

- Tâche de classification: $f : \mathbb{R}^d \to \{0, \ldots, m-1\}$

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

- Tâche de régression: $f : \mathbb{R}^d \to \mathbb{R}$

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

- Tâche d'estimation de densité $f : \mathbb{R}^d \to \mathbb{R}^+$ une fonction de densité (ou masse) de probabilité

  log vraisemblance négative: $L(f(x)) = -\log f(x)$

L'évaluation de performance est souvent la moyenne d'une fonction de coût (perte) sur un ensemble de donnée de test

Fonctions de coût de substitution

« surrogate loss »

- Pour une tache de classification: $f : \mathbb{R}^d \to \{0, \ldots, m-1\}$
  erreur de classification: $L(f(x), y) = I_{\{f(x) \neq y\}}$

Problème: difficile d'*optimiser* directement le taux d'erreur de classification
(gradient 0 partout. NP-difficiel avec un classifieur linéaire) Utiliser un coût de substitution!

	Classifieur binaire	Classifier multiclasse
Classifieur probabilistic	Donne la probabilité que la classe soit 1 $g(x) \approx P(y=1 \mid x)$ Probabilité de classe 0 est $1-g(x)$ Entropie-croisée binaire: $L(g(x),y) = -(y \log(g(x)) + (1-y) \log(1-g(x))$ Fonction de décision: $f(x) = I_{g(x)>0.5}$	Outputs a vector of probabilities: $g(x) \approx (P(y=0\|x), \ldots, P(y=m-1\|x))$ Moins log vraisemblance conditionnelle $L(g(x),y) = -\log g(x)_y$ Fonction de décision: $f(x) = \text{argmax}(g(x))$
Non-probabilistic classifier	Donne un «score» $g(x)$ pour la classe 1. score pour l'autre classe est $-g(x)$ Hinge loss: $L(g(x),t) = \max(0, 1-tg(x))$ où $t=2y-1$ Fonction de décision: $f(x) = I_{g(x)>0}$	Outputs a vector $g(x)$ of real-valued scores for the m classes. Multiclass margin loss $L(g(x),y) = \max(0, 1+\max_{k \neq y}(g(x)_k) - g(x)_y)$ Fonction de décision: $f(x) = \text{argmax}(g(x))$

Risque espéré v.s. Risque empirique

On nomme « risque » la perte « moyenne »

Exemple (x,y) supposés tirés i.i.d. d'une distribution $p(\mathrm{x},\mathrm{y})$ inconnue (provenant de la nature ou un procédé industriel)

- Erreur de généralisation = Risque espéré
  (ou juste «Risque» = l'espérance de la perte)
  «l'erreur qu'on va faire en moyenne sur l'infinité d'exemples futurs de cette distribution inconnue»

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

- Risque empirique = la perte moyenne sur un ensemble de données fini D (tiré de p)
  «l'erreur qu'on fait en moyenne sur les exemples de cet ensemble de données»

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

où $|D|$ est le nombre d'exemples dans D

Estimer le risque espéré

- On ne peut pas le calculer exactement… parce que $p(\mathrm{x},\mathrm{y})$ est inconnue!
- On peut l'estimer de manière approximative:
  Si on ne s'est aucunement servi de D pour choisir f le risque empirique est un estimé non-biaisé (mais bruité) du risque espéré:

$$R(f) \approx \hat{R}(f, D)$$

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

Plus précisément estimateur non-biaisé signifie que:

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

ce qui se traduit aussi par $$\lim_{|D| \to \infty} \hat{R}(f, D) = R(f)$$

Un algorithme d'apprentissage correspond souvent à la combinaison des éléments suivants:

(donnés explicitement ou implicitement)

✓ la spécification d'une famille de fonctions F (souvent une famille paramétrée)

✓ une manière d'évaluer la qualité d'une fonction $f \in F$ (typiquement utilisant une fonction de coût (perte) L ex: mesure à quel point f prédit mal la classe)

✓ une manière de chercher la «meilleure» fonction $f \in F$ (ex: optimisation des paramètres pour minimiser la perte).

Apprendre c'est choisir une fonction parmi un ensemble de fonctions

Ensemble de toutes les fonctions possibles de l'univers

Ensemble F de fonctions considéres

$\hat{f}(D_{\text{train}})$

fonction trouvée par l'algorithme en utilisant l'ensemble d'entrainement

quel principe pour la trouver?

Minimisation du risque empirique

- On aimerait trouver un prédicteur qui minimise l'erreur de généralisation (risque espéré)
- Mais on ne peut même pas la calculer!
- À la place: **Principe de minimisation du risque empirique**
  «Trouver le prédicteur qui minimise la perte moyenne sur un ensemble d'entraiment»

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

C'est la phase d'entrainement

Estimer l'erreur de généralisation du prédicteur $\hat{f}(D_{\text{train}})$

Le problème:

- Principe de minimisation du risque empirique: on entraîne un modèle (on adapte ses paramètres) de manière à ce qu'il fasse un minimum d'erreurs sur l'ensemble d'entraînement. $\hat{f}(D_{\text{train}})$
- MAIS ce qui nous intéresse vraiment, c'est de bien généraliser sur de nouveaux exemples.
- Puisque les paramètres du modèle sont choisis, spécialisés, pour minimiser l'erreur sur les exemples d'entraînement, celle-ci sous-estime l'erreur de généralisation.
- Ainsi l'erreur d'entraînement n'est pas un bon estimé de l'erreur de généralisation (c'est un estimé *biaisé*).

Estimer l'erreur de généralisation du prédicteur $\hat{f}(D_{\text{train}})$

- $\hat{R}(f, D)$ est un bon estimateur de $R(f)$ à condition que:
  - D n'aie pas été utilisé pour trouver/choisir f
    sinon biaisé ⇨ on ne peut donc pas utiliser l'ensemble d'entrainement!
  - D est assez grand (sinon estimateur trop bruité); tiré de p

➡ On doit garder un ensemble de test séparé $D_{\text{test}} \neq D_{\text{train}}$ pour correctement estimer l'erreur de généralisation de $\hat{f}(D_{\text{train}})$

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

erreur de generalization

erreur moyenne sur l'ensemble de **test** (jamais utilisé pour entrainement)

Ceci est la phase de <u>test</u>

Estimation d'erreur de généralisation:
Validation simple

Ensemble de toutes les données étiquetées dont on dispose:

$D=$

(x_1, y_1)

(x_2, y_2)

$\vdots$

(x_N, y_N)

On sépare nos données en **deux**

(attention il peut s'avérer nécessaire de d'abord mélanger les rangées de données)

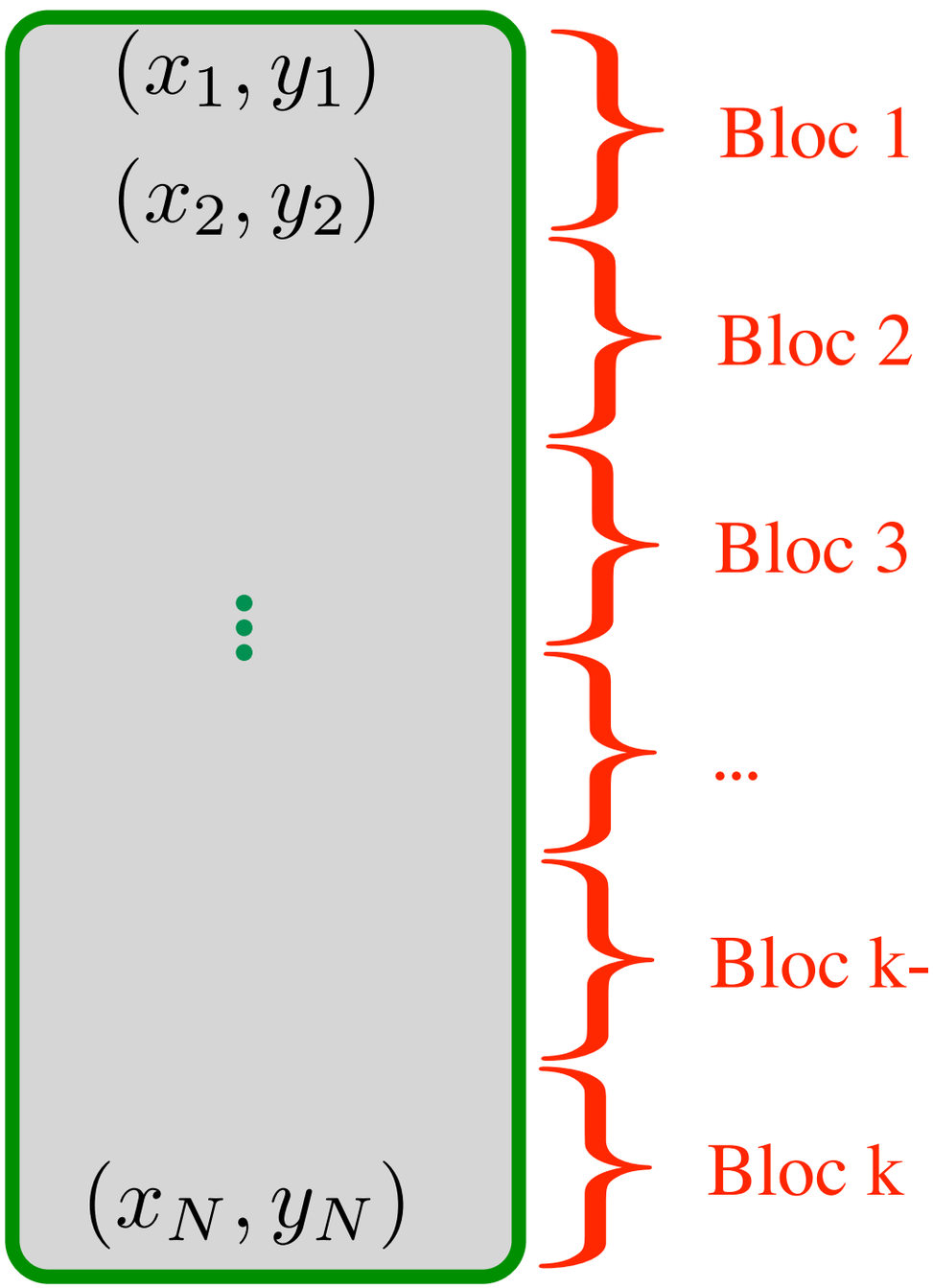
Ensemble d'entraînement (taille n)

On entraîne le modèle pour minimiser l'erreur sur l'ensemble d'entraînement

Ensemble de test (taille m)

On évalue la performance de généralisation en mesurant les erreurs sur l'ensemble de test qu'on n'a jamais regardé pendant l'entraînement.
(mesure de performance "hors échantillon").

Si on n'a pas assez de données: validation croisée *(en k blocs)*



Idée simple: on répète plusieurs fois la procédure entraînement/test en divisant différemment l'ensemble de données.

Entraînement sur	***Calcul de l'erreur (test) sur***
$D \setminus$ Bloc 1	Bloc 1
$D \setminus$ Bloc 2	Bloc 2
...	...
$D \setminus$ Bloc k	Bloc k

Notre estimé de l'erreur de généralisation de l'algorithme sur ce problème est déduit de la somme des erreurs obtenues sur tous les blocs.

En Anglais on appelle cela *k-fold cross validation.*
Le cas où k=N est appelé *leave-one-out* ou *jackknife*.

Notions de capacité de sur-apprentissage de sous-apprentissage

Apprendre c'est choisir une fonction parmi un ensemble de fonctions

Ensemble de toutes les fonctions possibles de l'univers

Ensemble F de fonctions considéres

$\hat{f}(D_{\text{train}})$

fonction trouvée par l'algorithme en utilisant l'ensemble d'entrainement

Notion de capacité
ou complexité de modèle

- Correspond à la «taille» ou «richesse» de l'ensemble de fonction considéré (fonctions ± flexible)

- *Souvent* relié au **nombre** de **paramètres** libres (à apprendre à partir de l'ensemble d'entraînement) Habituellement plus il y en a, plus grande est la capacité.

- Mais pas toujours... il s'agit plutôt du nombre de degrés de liberté effective de la fonction (capacité effective).

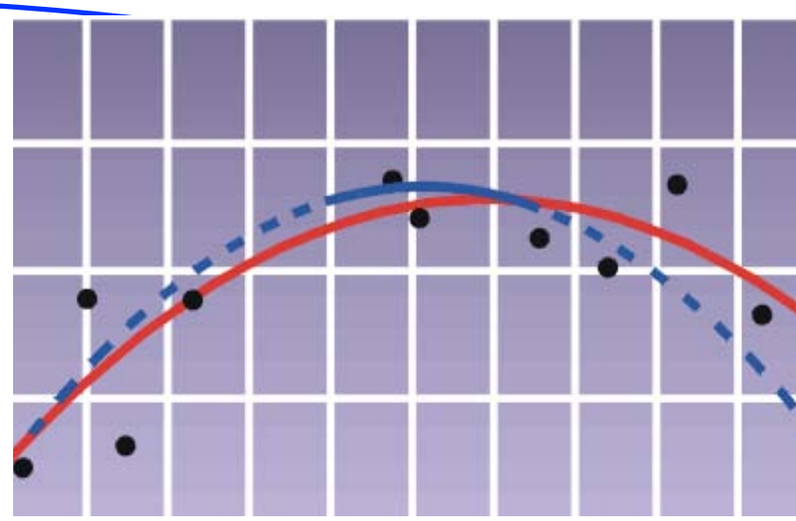
- Il existe des mesures formelles pour certains cadres restreints. Ex: VC-dimension (Vapnik-Chervonenkis).

Ex de familles de fonctions paramétrées

$F_{polynomial\ p}$

Prédicteur polynomial (de degré p):

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

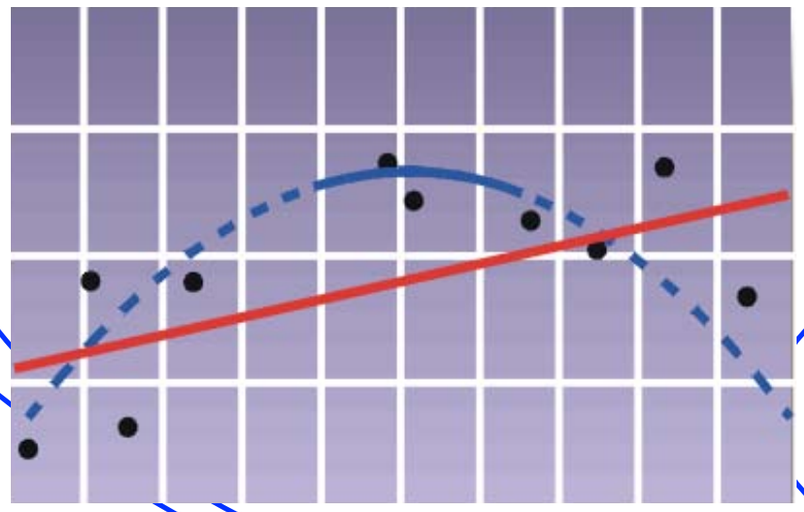


F_{linear}

Prédicteur linéaire (affine): $f_\theta(x) = wx + b$ (en 1 dimension)

(«régression linéaire») $f_\theta(x) = w^T x + b$ (en d dimensions)

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



F_{const}

Prédicteur constant: $f_\theta(x)=b$

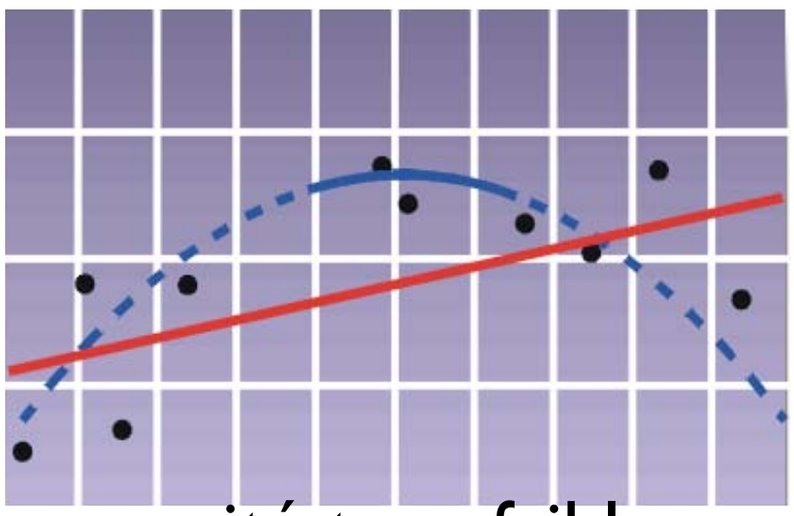
où $\theta=\{b\}$

(Prédit toujours la même valeur ou classe!)

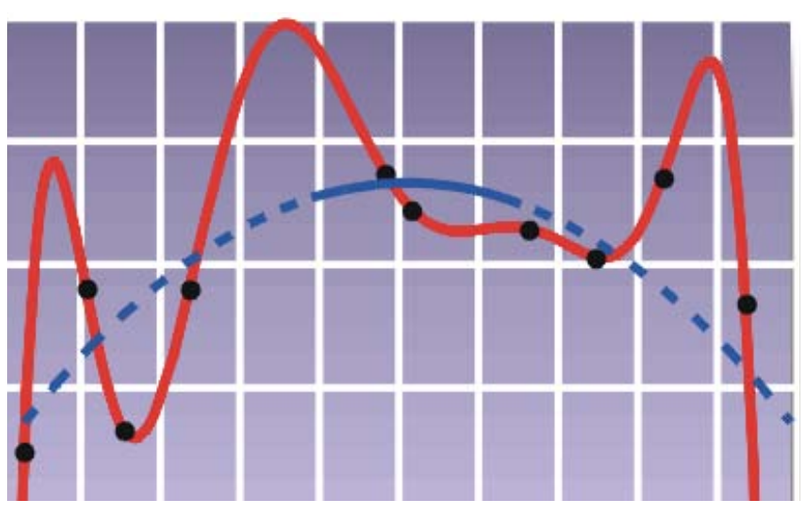
Contrôle de la capacité

- Le choix de l'ensemble de fonction et des hyper-paramètres des algorithmes permettent de contrôler la capacité du modèle.
- C'est indispensable pour avoir une bonne généralisation.

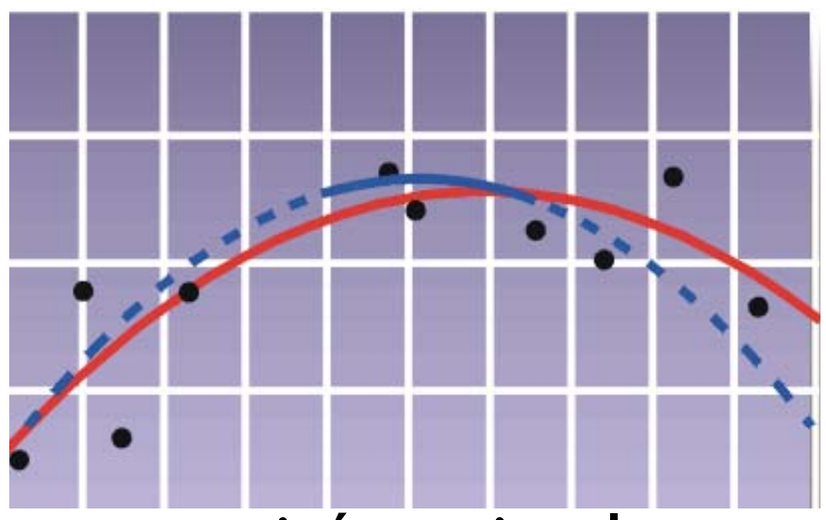
Ex: régresion 1D



capacité trop faible
⇨sous-apprentissage



capacité trop élevée
⇨sur-apprentissage



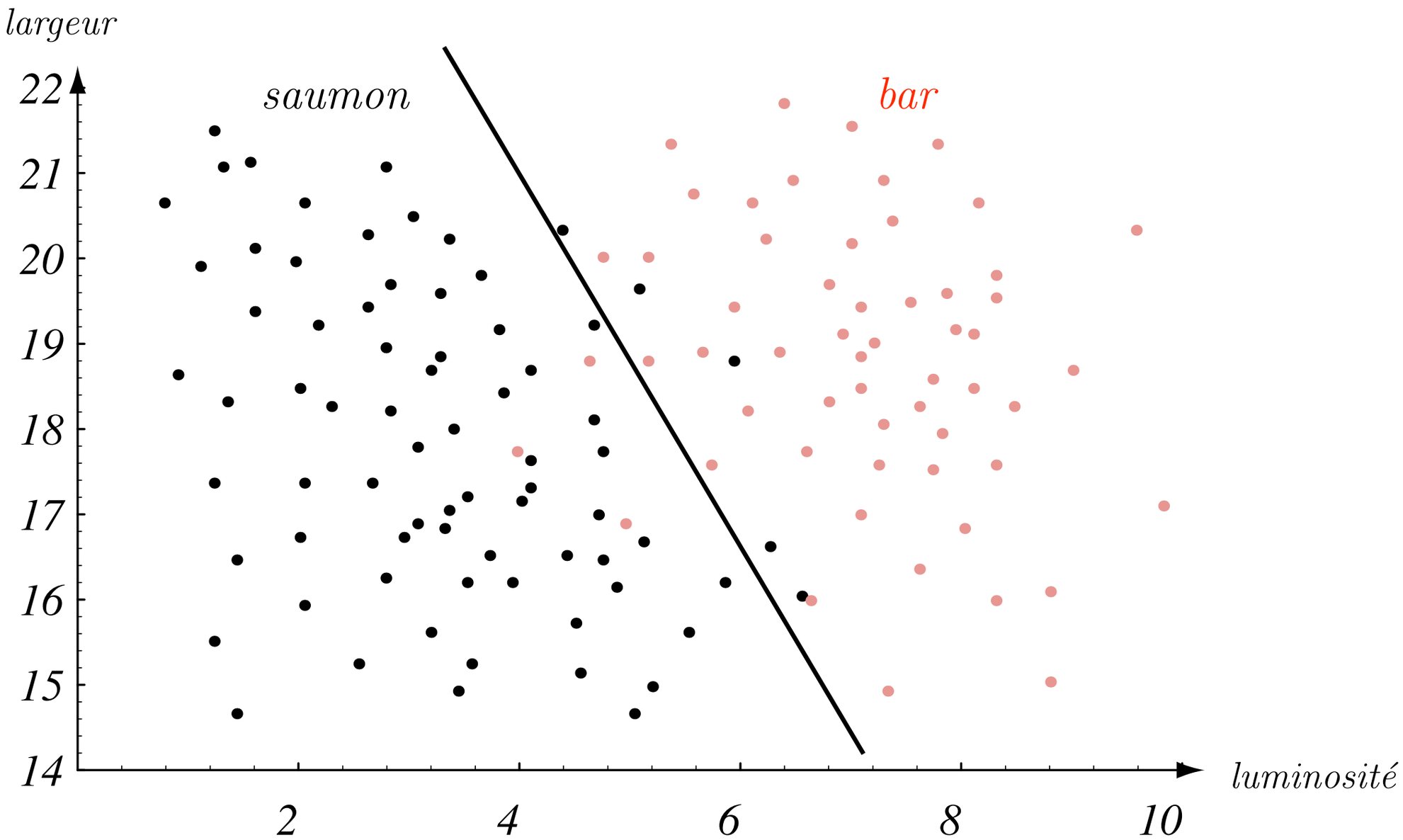
capacité optimale
⇨bonne généralisation

la performance sur l'ensemble d'entraînement n'est pas un bon estimé de la généralisation

Ex: classification 2D

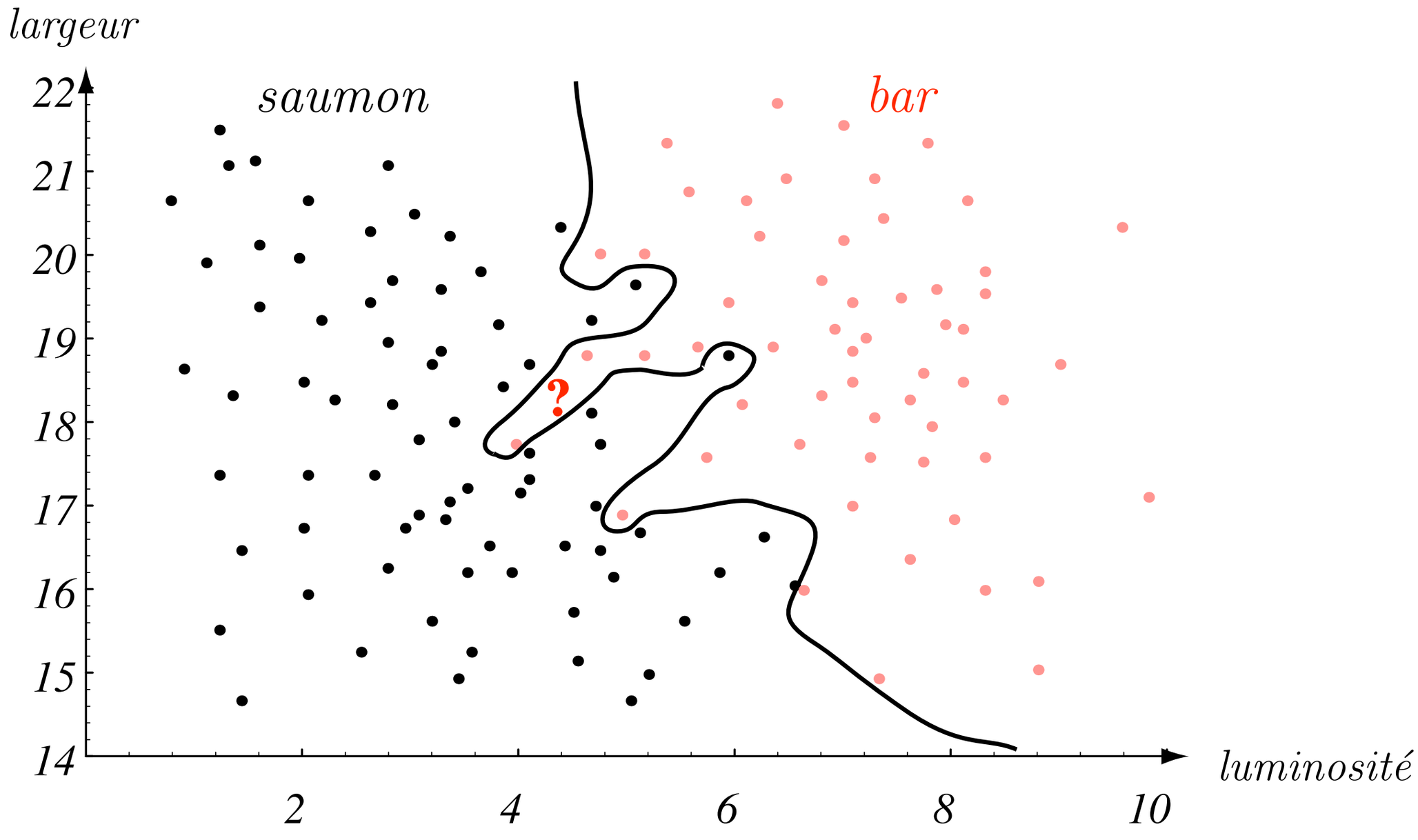
Classifieur linéaire

- Ensemble de fonction trop pauvre
  (trop peu flexible)
- = **Capacité trop faible** pour ce problème
  (par rapport à la quantité de données)
- => Sous-apprentissage



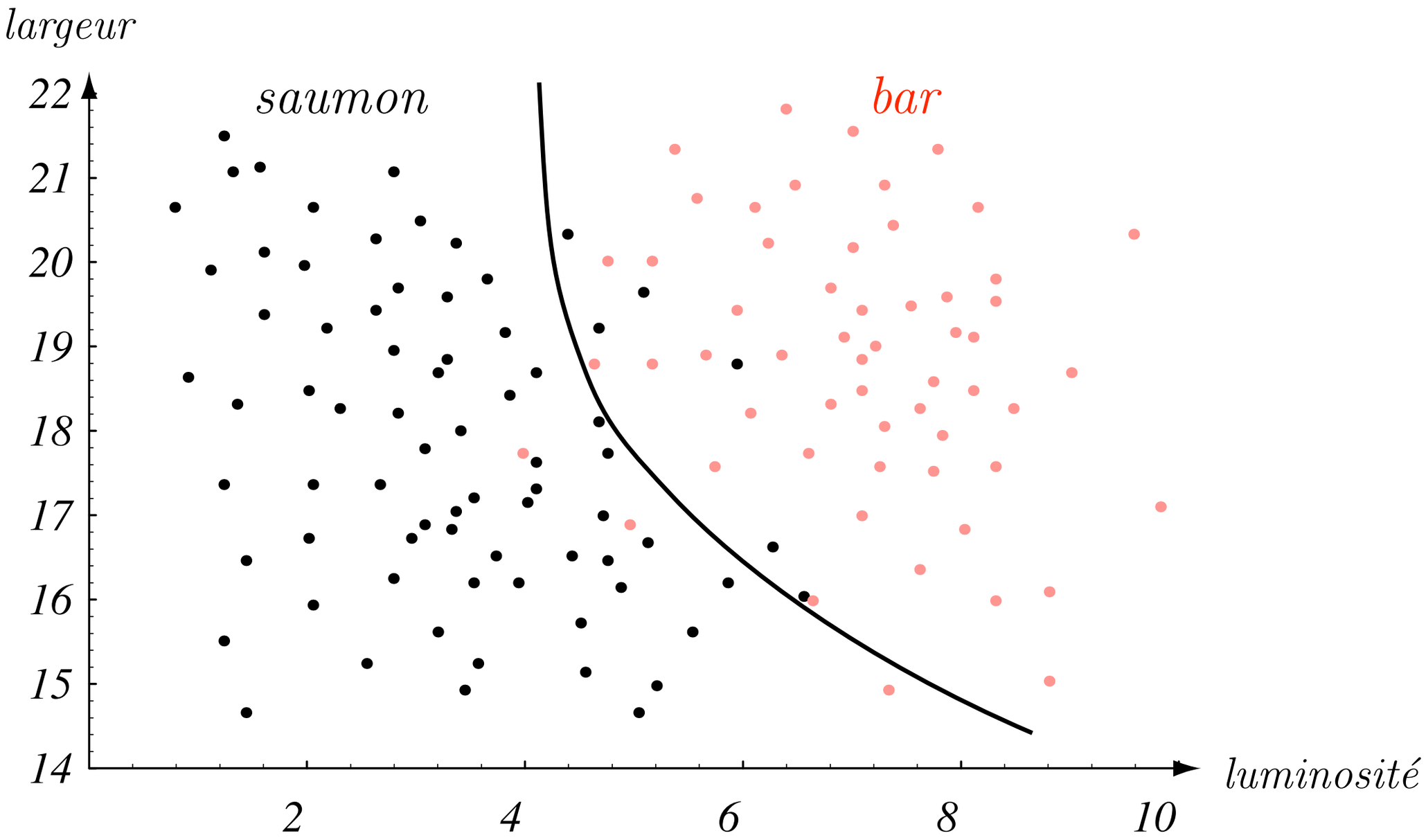
Nombre d'erreurs d'entraînement: 3+5 = 8

- Ensemble de fonction trop riche
  (trop flexible)
- = **Capacité trop élevée** pour ce problème
  (par rapport à la quantité de données)
- => Sur-apprentissage



Nombre d'erreurs d'entrainement: 0

- **Capacité optimale** pour ce problème
  (par rapport à la quantité de données)
- => Meilleure généralisation
  (sur les futurs points de test)



Nombre d'erreurs d'entrainement: 3+6 = 9

Principe du rasoir d'Ockham

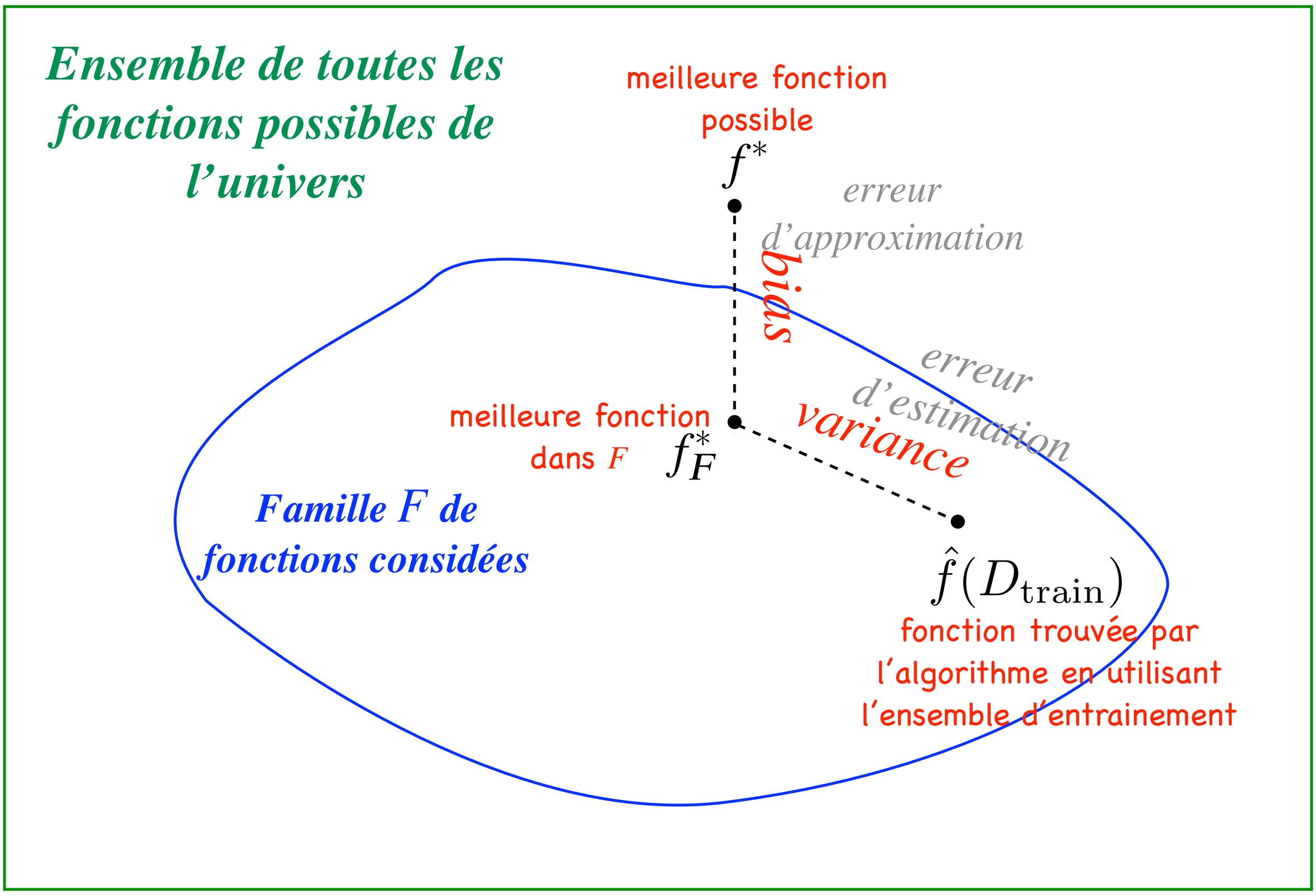
William of Ockham

En apprentissage:
Choisir le modèle le plus **simple** possible qui apprend bien les données.

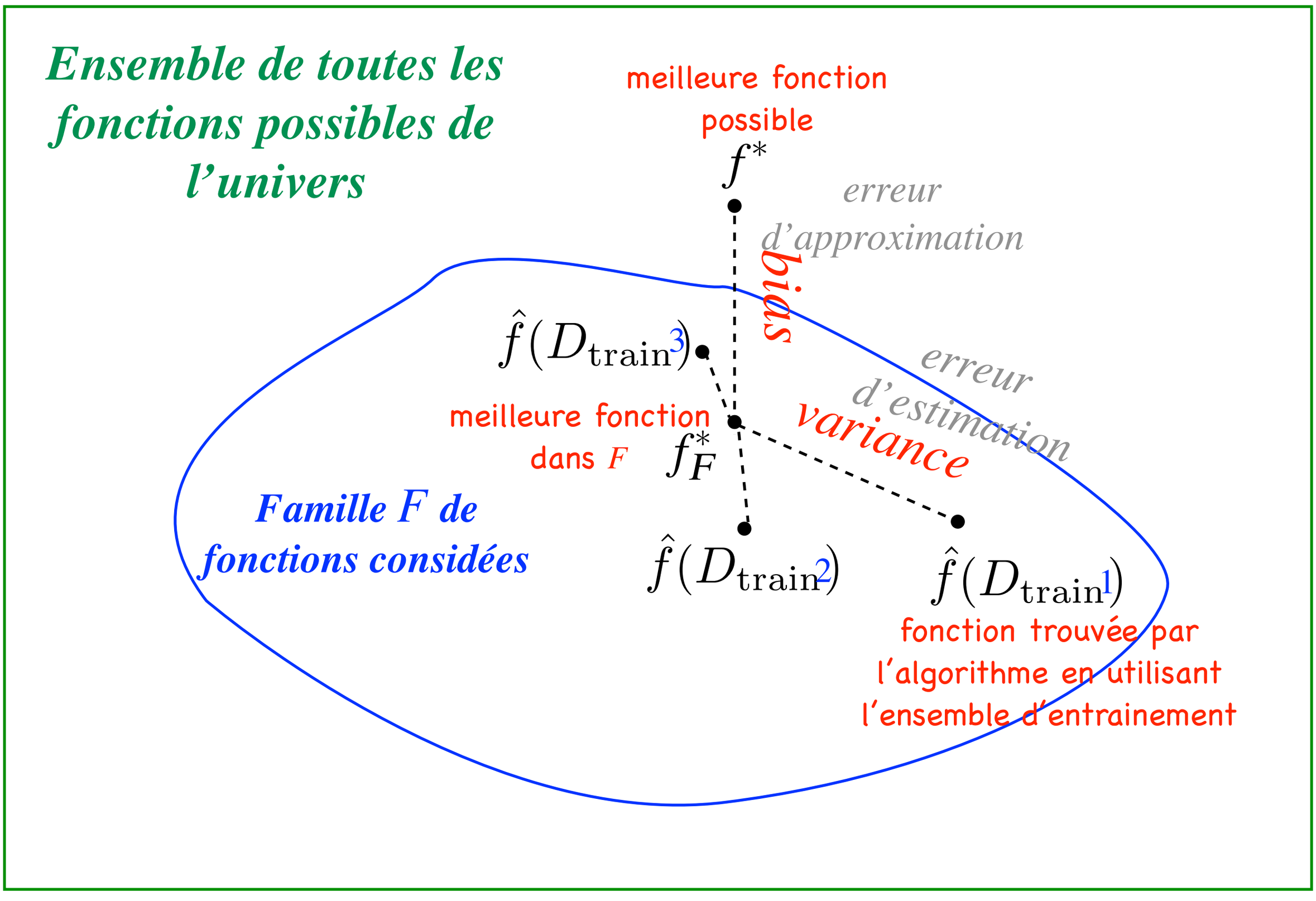
Mais problème: plus la famille de fonction est riche (**complexe**) plus les données d'entraînement sont bien «apprises»...

Décomposition de l'erreur de généralisation

biais + variance



Quelle est la source de la variance?



Quelle est la bonne capacité?
Le **dilemme biais-variance**

- Si on choisit un ensemble F plus riche: capacité ↑
  ⇨ biais ↓ *mais* variance ↑.

- Si on choisit un ensemble F plus petit: capacité ↓
  ⇨ variance ↓ *mais* biais ↑.

- Le compromis optimal... dépendra de nombre d'exemples d'entrainement ***n***

- Plus grand ***n*** ⇨ variance ↓
  On peut alors se permettre d'augmenter la capacité (pour diminuer le bias)
  ⇨ on peut se permettre des modèles plus complexes

- Le meilleur régularizeur est davantage de données!

Sélection de modèle
Sélection d'hyper-paramètres

en pratique

Sélection de modèle

On va considérer:

- Plusieurs algos d'apprentissage.
- Pour chaque algo, plusieurs choix de valeurs de ses hyper-paramètres
  (ex. nb de voisins k dans k-NN, nb de neurones dans un réseau de neurones, ...)

Pour chaque cas, on va entraîner un modèle (optimiser ses paramètres sur un ensemble d'entraînement) puis évaluer sa performance de généralisation hors-échantillon (sur un ensemble de validation)

Sélection de modèle
(et hyper-paramètres)

S'assurer que les exemples dans D sont dans un ordre aléatoire. Diviser D en 3: D_{train} D_{valid} D_{test}

$D=$

(x_1, y_1)

(x_2, y_2)

$\vdots$

(x_N, y_N)

Ensemble d'entrainement D_{train}

Ensemble de validation D_{valid}

Ensemble de test D_{test}

Méta-Algorithme de sélection de modèle:

Pour chaque modèle considéré model (algo) $\mathbf{A}$:

Pour chaque configuration d'hyper-paramètres λ:

- entrainer $\mathbf{A}$ avec hyperparams λ sur D_{train}

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

- évaluer ce prédicteur sur D_{valid} (avec la mesure d'évaluation d'intérêt)

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

Repérer $\mathbf{A}^*, \lambda^*$ qui donnaient le meilleur $e_{\mathbf{A}_\lambda}$

Soit retourner $f^* = f_{\mathbf{A}^*_{\lambda^*}}$

Ou réentrainer et retourner

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

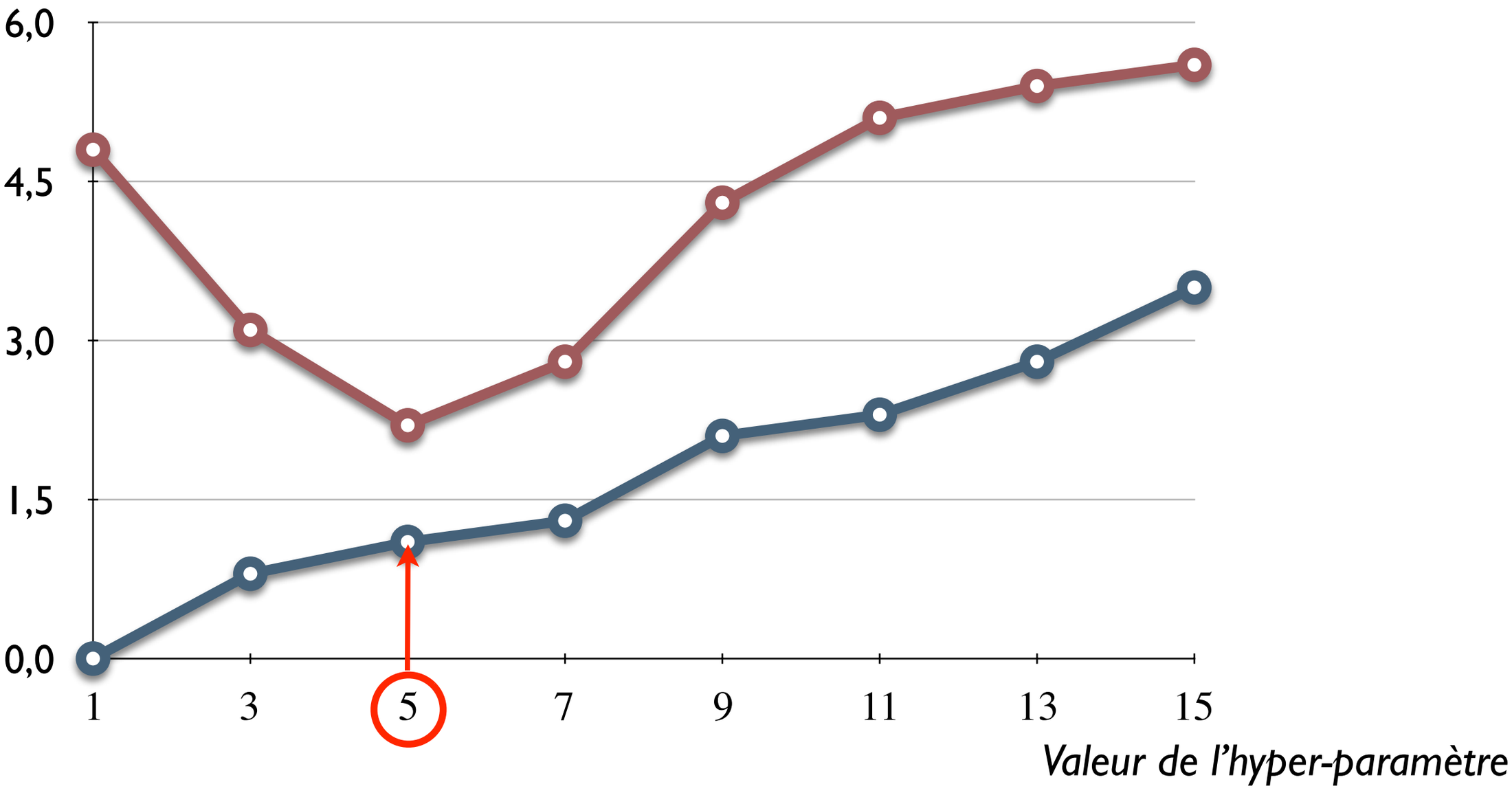
Pour finir: calculer un estimation non-biaisée de la performance de généralisation de f^* en utilisant D_{test}

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

D_{test} ne doit jamais avoir été utilisé durant l'entrainement ou la sélection du modèle pour pour sélectionner, apprendre, ou ajuster quoi que ce soit

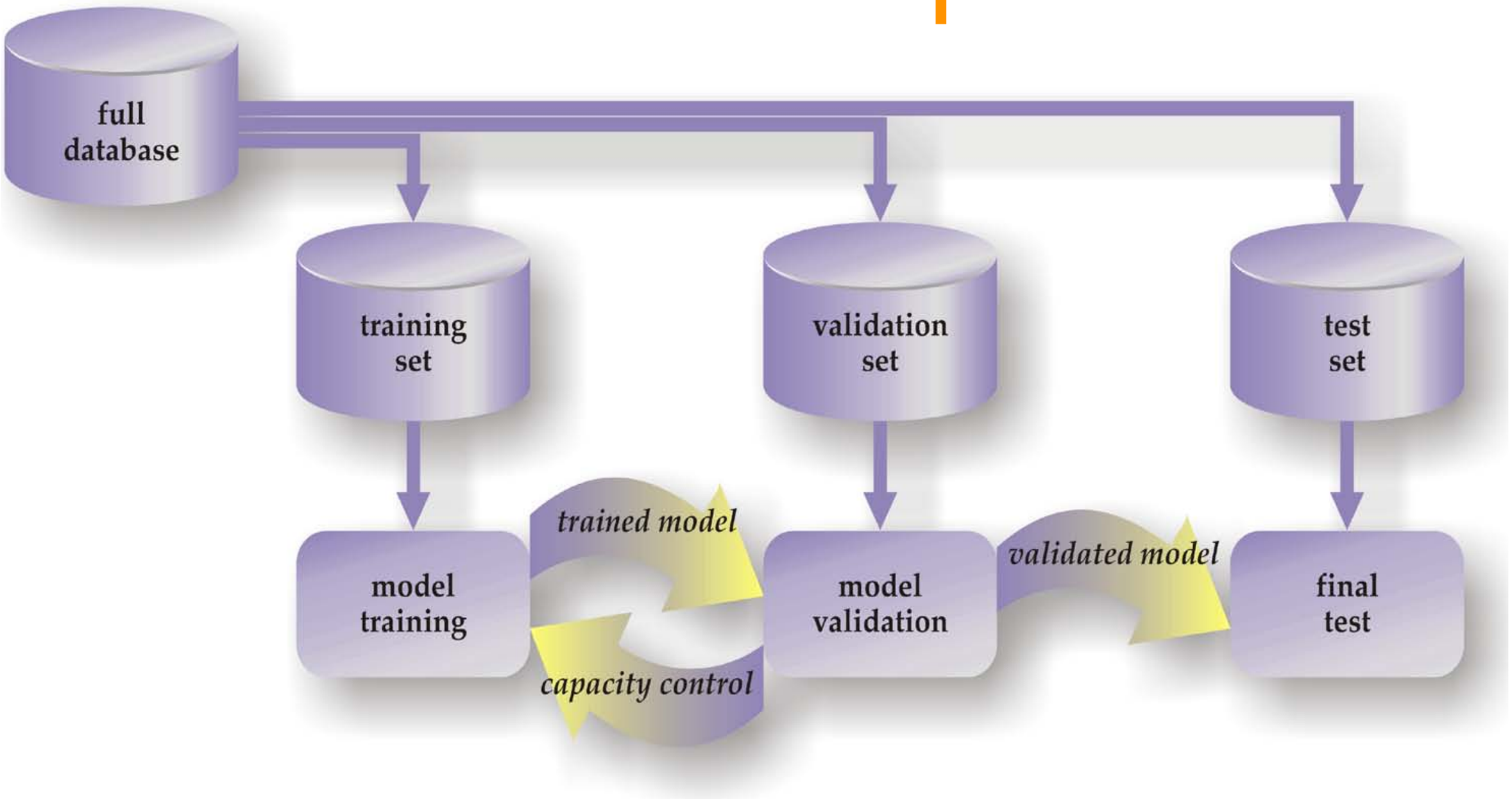
Courbes d'apprentissage

- Erreur d'entraînement
- Erreur de validation



La valeur de l'hyper-paramètre donnant l'erreur minimale
sur l'ensemble de validation est **5** (alors que c'est **1** pour l'ensemble d'entraînement)

Sélection de modèle: comment on procède



L'évaluation de performance finale doit s'effectuer sur des données qui n'ont servi **ni** pour l'entrainement **ni** pour la sélection de modèles ou de capacité.

Paramètres v.s. hyper-paramètres?

- Les paramètres sont optimisés sur l'ensemble d'entraînement
- Les hyper-paramètres sont fixés avant l'entrainement.
  On les optimise sur un ensemble de validation (boucle externe)
  Généralement ils contrôlent la capacité
- Que se passerait-il si on choisissait à la fois les hyper-paramètres et paramètres optimaux sur l'ensemble d'entraînement?

  ➡ Tendrait à choisir la capacité la plus élevé possible (pour faire le moins d'erreur sur l'ensemble d'entraînement)

  ➡ Sur-apprentissage

  ➡ Mauvaise généralisation

Paramètres?
Hyper-paramètres?

Ex: Pour les histogrammes

- Que sont les hyper-paramètres?
  - Nombre de subdivisions (cases)
- Que sont les paramètres?
  - Les comptes dans chaque case

Paramètres?
Hyper-paramètres?

Ex: Pour k-PPV

- Que sont les hyper-paramètres?
  - Taille du voisinage: k
- Que sont les paramètres?
  - On mémorise l'ensemble de données d'entraînement

Paramètres?
Hyper-paramètres?

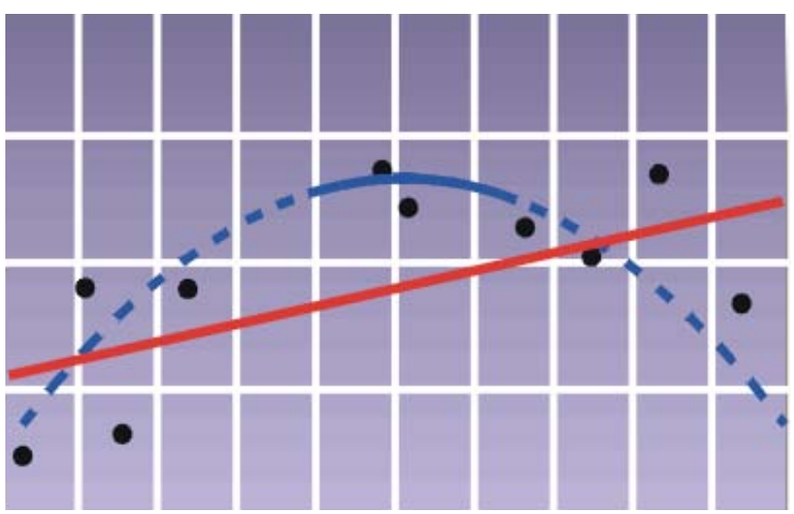
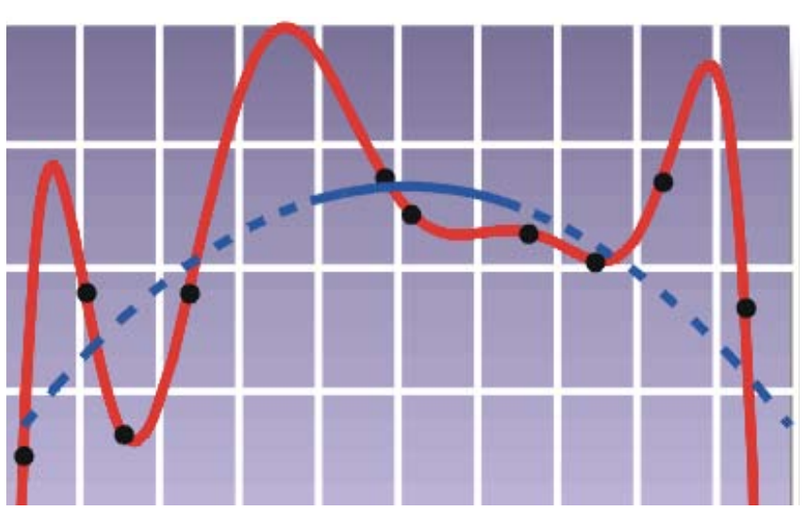
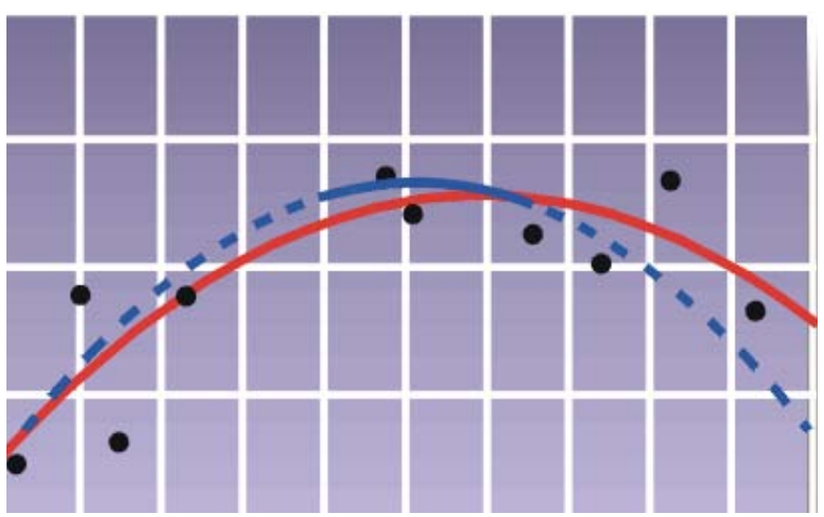
Ex: Pour fenêtres de Parzen

- Que sont les hyper-paramètres?
  - Taille du voisinage: largeur du noyau
- Que sont les paramètres?
  - On mémorise l'ensemble de données d'entraînement

Paramètres?
Hyper-paramètres?

Ex: Pour régression polynomiale

- Que sont les hyper-paramètres?
  - degré du polynôme
- Que sont les paramètres?
  - coefficients appris du polynôme

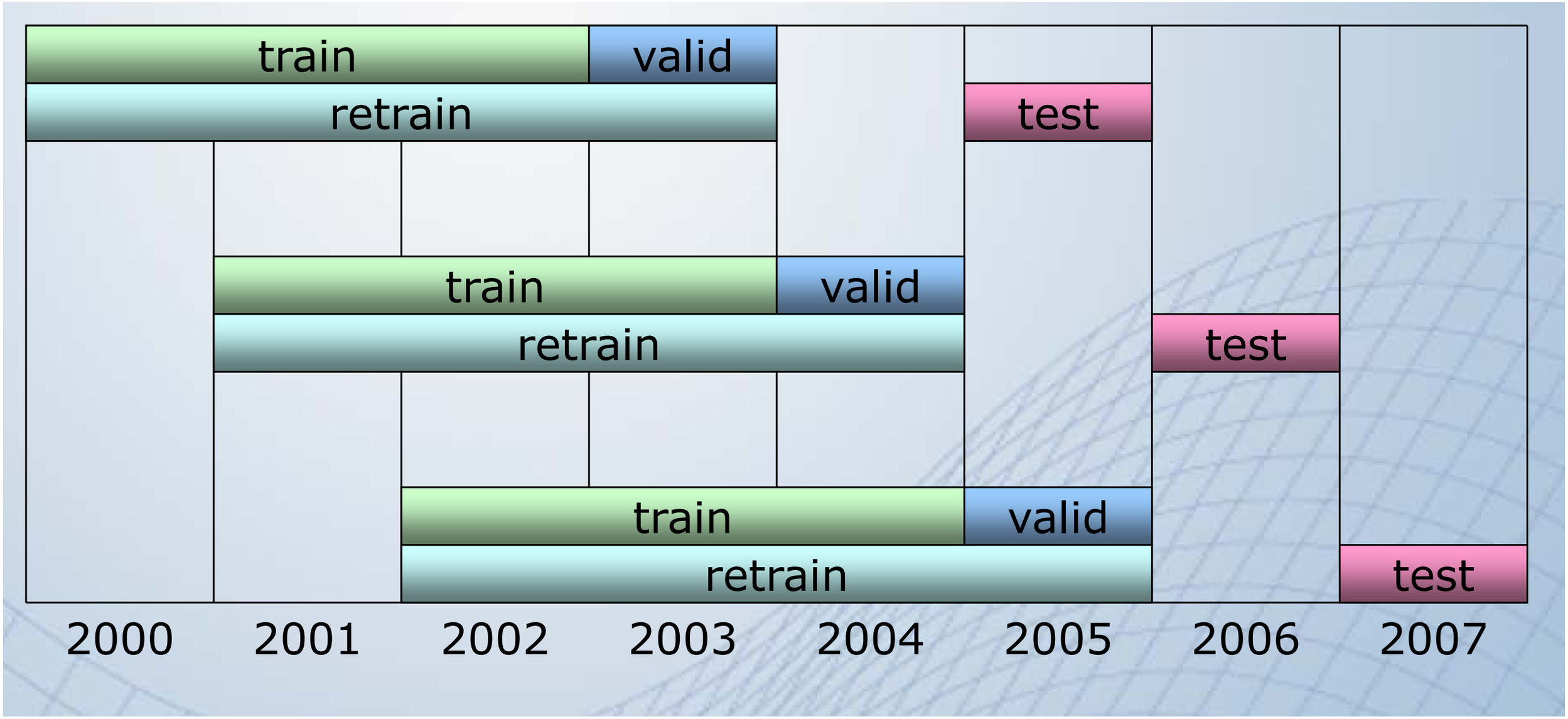




Hyper-paramètres?

Il peut y avoir plusieurs hyper-paramètres

- Contrôlant la «taille» de la famille de fonction
- Induisant une préférence plus-ou-moins restrictive parmi les fonctions d'une famille
- Choisissant une famille parmi plusieurs
- Contrôlant certains aspects de l'optimisation des paramètres (qui sera effectuée sur l'ensemble d'entraînement).

Pour la prédiction de séries temporelles: validation séquentielle



Double validation croisée

- Si on a peu de données et qu'on veut faire de la sélection d'hyper-paramètre ET obtenir un estimé non biaisé de l'erreur de généralisation de la procédure, on peut faire une double validation croisée:
  - Un premier niveau de validation croisée est utilisé pour obtenir des paires d'ensembles entraînement/ test
  - Un deuxième niveau (imbriqué) de validation croisée où l'on re-subdivise l'ensemble d'entraînement en entraînement+validation sert à sélectionner les hyper-paramètres.