

Apprentissage Automatique MI 203

**Généralisation
10/3/2020**

S. Herbin, A. Chan Hon Tong

Apprentissage supervisé

- On veut construire une fonction de décision D à partir d'exemples
- On dispose d'un **ensemble d'apprentissage** \mathcal{L} sous la forme de paires $\{x_i, y_i\}$ où x_i est la donnée à classer et y_i est la classe vraie:

$$\mathcal{L} = \{\mathbf{x}_i, y_i\}_{i=1}^N$$

- L'apprentissage consiste à identifier cette fonction de classification dans un certain espace **paramétrique** W optimisant un certain **critère** E :

$$\hat{\mathbf{w}} = \arg \max_{\mathbf{w} \in W} \mathcal{E}(\mathcal{L}, \mathbf{w})$$

- On l'applique ensuite à de nouvelles données.

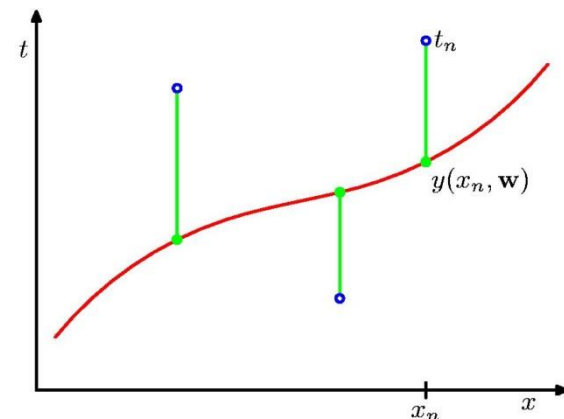
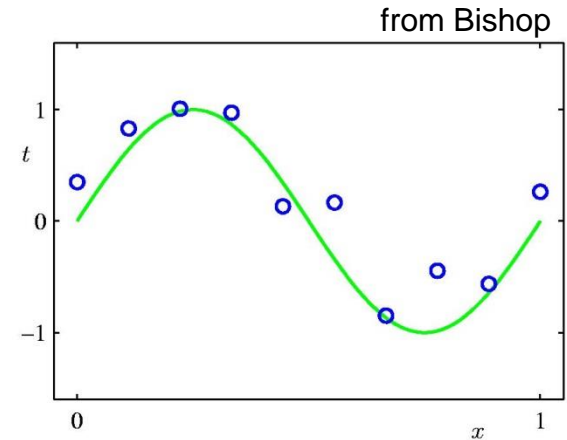
$$y = D(\mathbf{x}; \hat{\mathbf{w}})$$

Sources d'erreur

- Apprentissage = interpolation sur base de données
= « généralisation » à partir d'exemples
≠ mémorisation (apprentissage par cœur)
- La mesure de bon fonctionnement est l'erreur de généralisation
→ erreur sur des données nouvelles
- Problème: les données nouvelles sont par nature inconnues! (sinon, elles seraient utilisées)
➔ Il est nécessaire de faire des hypothèses sur leur nature.
- Une des hypothèses les plus simples est de supposer un certain niveau de régularité.

Exemple illustratif: régression polynomiale

- La courbe verte est la véritable fonction à estimer (non polynomiale)
- Les données sont uniformément échantillonnées en x mais bruitées en y .
- L'erreur de régression est mesurée par la distance au carré entre les points vrais et le polynôme estimé.



Modèles linéaires généralisés (cas scalaire $y \in \mathbb{R}$)

biais

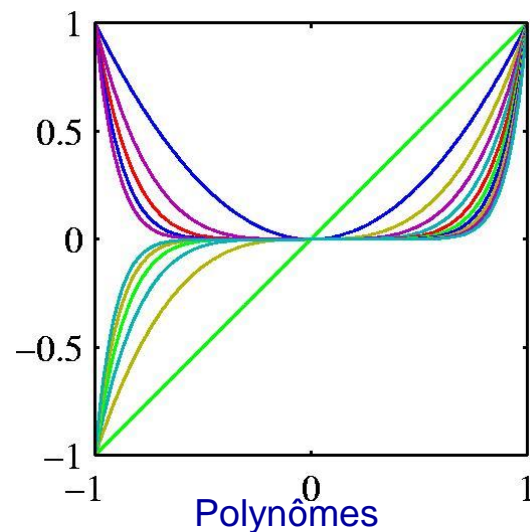
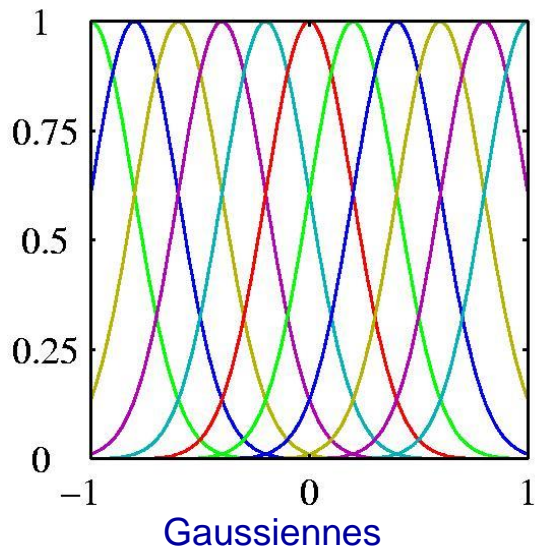
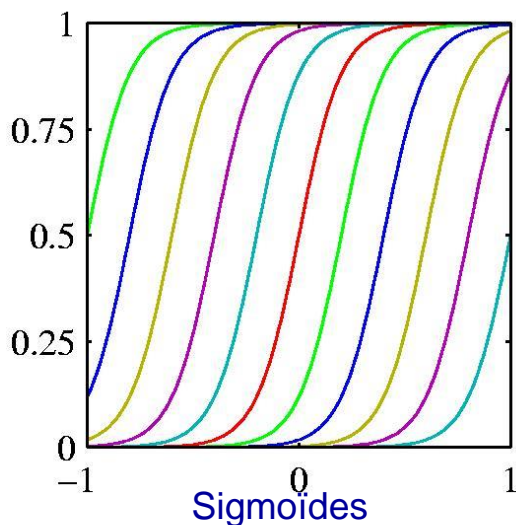


$$y(\mathbf{x}, \mathbf{w}) = w_0 + w_1 x_1 + w_2 x_2 + \dots = \mathbf{w}^T \mathbf{x}$$

$$y(\mathbf{x}, \mathbf{w}) = w_0 + w_1 \phi_1(\mathbf{x}) + w_2 \phi_2(\mathbf{x}) + \dots = \mathbf{w}^T \Phi(\mathbf{x})$$

- Principe simple: utiliser plusieurs **fonctions de base** ϕ encodant les données source (« features »)
- Une fois définies les fonctions, le problème reste **linéaire!**
- Comment trouver ces fonctions?
 - Se les donner
 - Les apprendre

Exemples classiques de fonctions de base 1D



Remarque: les sigmoïdes et gaussiennes sont des fonctions d'activation usuelles dans les réseaux de neurones

➔ les RN permettent d'apprendre les ϕ

Apprentissage = trouver W_{ML}

- Forme du prédicteur

$$y(\mathbf{x}, \mathbf{w}) = \sum_{j=0}^{M-1} w_j \phi_j(\mathbf{x}) = \mathbf{w}^T \boldsymbol{\phi}(\mathbf{x})$$

- Critère d'erreur (évaluation)

$$\mathcal{E}_{\text{RMS}}(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^N (\mathbf{w}^T \cdot \boldsymbol{\phi}(x_n) - t_n)^2$$

- Principe statistique: maximum de vraisemblance

$$W_{ML} = \underset{W}{\operatorname{argmax}} P(\mathbf{t} | \mathbf{x}, W)$$

Maximum de vraisemblance et moindres carrés

- Hypothèse de bruit gaussien:

$$t = y(\mathbf{x}, \mathbf{w}) + \epsilon \quad \text{where} \quad p(\epsilon|\beta) = \mathcal{N}(\epsilon|0, \beta^{-1})$$

- donne comme vraisemblance globale,

$$p(\mathbf{t}|\mathbf{X}, \mathbf{w}, \beta) = \prod_{n=1}^N \mathcal{N}(t_n | \mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}_n), \beta^{-1}).$$

- Si on calcule la log-vraisemblance

$$\begin{aligned} \ln p(\mathbf{t}|\mathbf{w}, \beta) &= \sum_{n=1}^N \ln \mathcal{N}(t_n | \mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}_n), \beta^{-1}) \\ &= \frac{N}{2} \ln \beta - \frac{N}{2} \ln(2\pi) - \beta E_D(\mathbf{w}) \end{aligned}$$

- On obtient le critère quadratique à optimiser

$$E_D(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^N \{t_n - \mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}_n)\}^2$$

Solution du Maximum de Vraisemblance

- On dérive le critère:

$$\nabla_{\mathbf{w}} \ln p(\mathbf{t}|\mathbf{w}, \beta) = \beta \sum_{n=1}^N \{t_n - \mathbf{w}^T \phi(\mathbf{x}_n)\} \phi(\mathbf{x}_n)^T = \mathbf{0}.$$

- La résolution de l'équation donne

- où

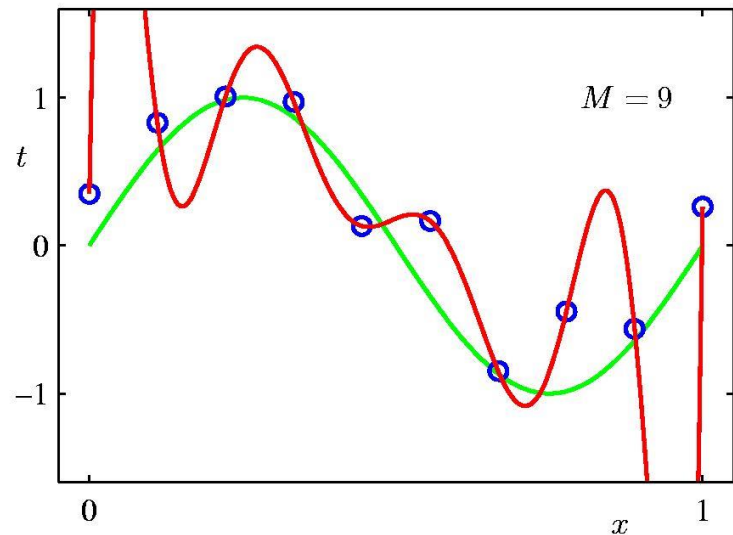
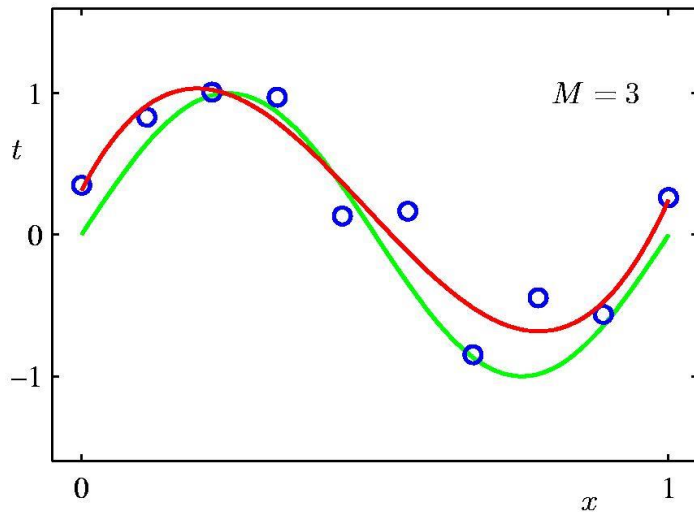
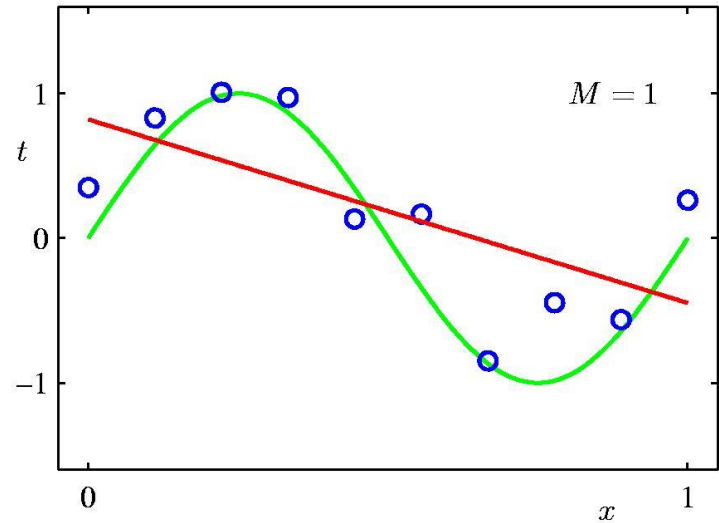
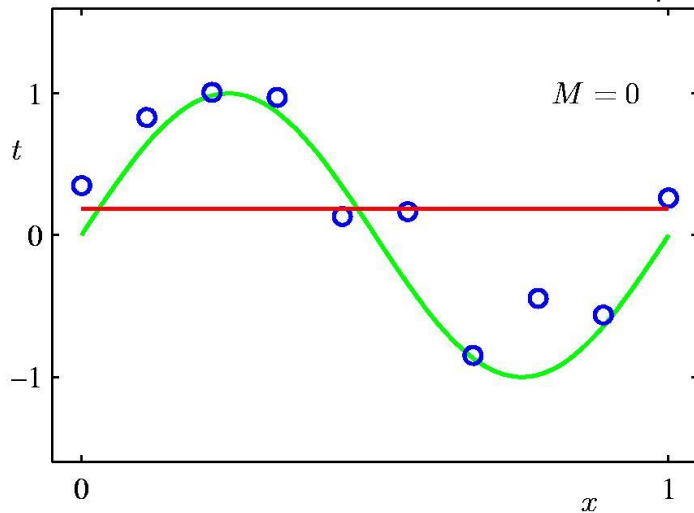
$$\mathbf{w}_{\text{ML}} = \left(\Phi^T \Phi \right)^{-1} \Phi^T \mathbf{t}$$

Moore-Penrose
pseudo-inverse, Φ^\dagger .

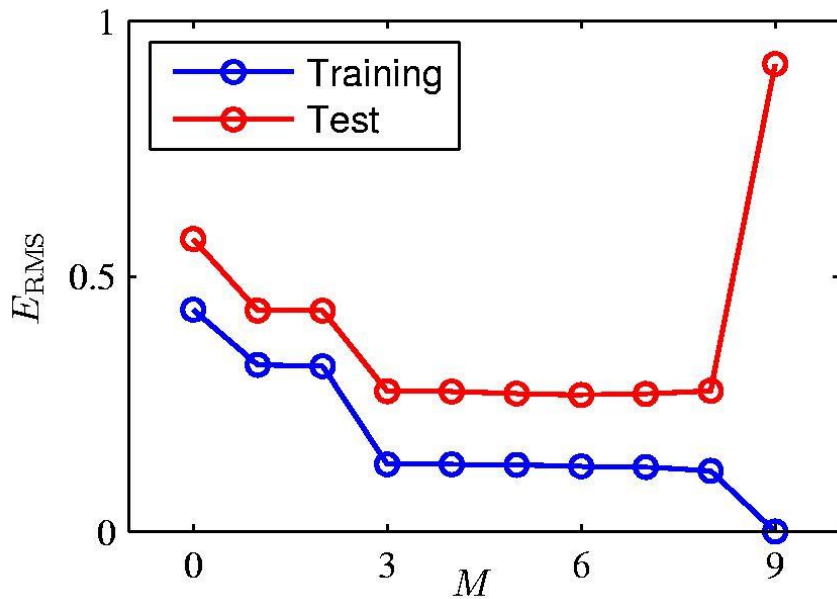
$$\Phi = \begin{pmatrix} \phi_0(\mathbf{x}_1) & \phi_1(\mathbf{x}_1) & \cdots & \phi_{M-1}(\mathbf{x}_1) \\ \phi_0(\mathbf{x}_2) & \phi_1(\mathbf{x}_2) & \cdots & \phi_{M-1}(\mathbf{x}_2) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_0(\mathbf{x}_N) & \phi_1(\mathbf{x}_N) & \cdots & \phi_{M-1}(\mathbf{x}_N) \end{pmatrix}.$$

Quelles sont les meilleures régressions?

from Bishop



Erreurs et valeurs des coefficients



Comparaison des erreurs de test et d'apprentissage

	$M = 0$	$M = 1$	$M = 3$	$M = 9$
w_0^*	0.19	0.82	0.31	0.35
w_1^*		-1.27	7.99	232.37
w_2^*			-25.43	-5321.83
w_3^*			17.37	48568.31
w_4^*				-231639.30
w_5^*				640042.26
w_6^*				-1061800.52
w_7^*				1042400.18
w_8^*				-557682.99
w_9^*				125201.43

Coefficients des polynômes

Moindre carrés régularisés

Idée: on rajoute une pénalisation à la fonction de coût:

Coût d'attache
aux données

$$\frac{1}{2} \sum_{n=1}^N \{t_n - \mathbf{w}^T \phi(\mathbf{x}_n)\}^2 + \frac{\lambda}{2} \mathbf{w}^T \mathbf{w}$$

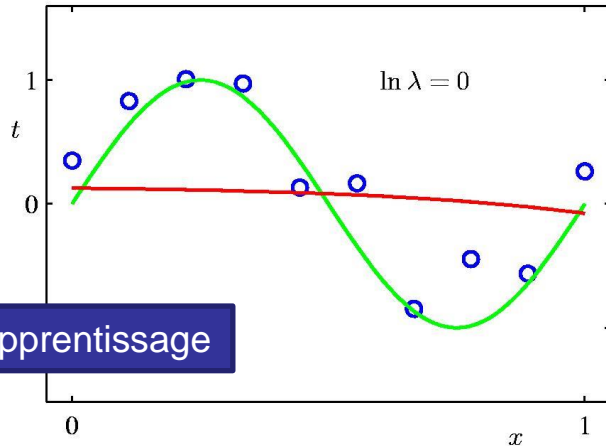
Dont l'optimum exact est alors:

$$\mathbf{w} = \left(\lambda \mathbf{I} + \Phi^T \Phi \right)^{-1} \Phi^T \mathbf{t}.$$

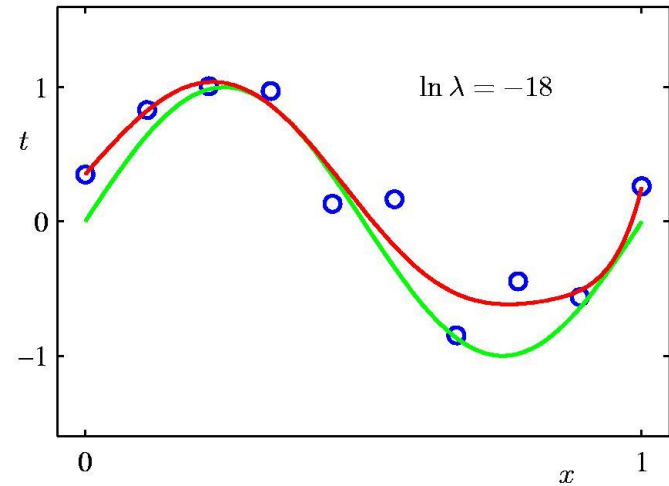
Paramètre de
régularisation

Si on pénalise les grandes valeurs des coefficients du polynôme, on obtient une fonction moins « zigzagante »

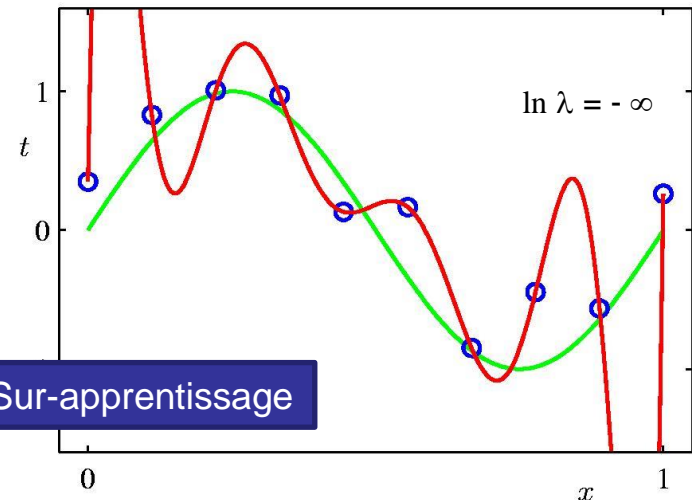
Effet de la régularisation



Sous-apprentissage



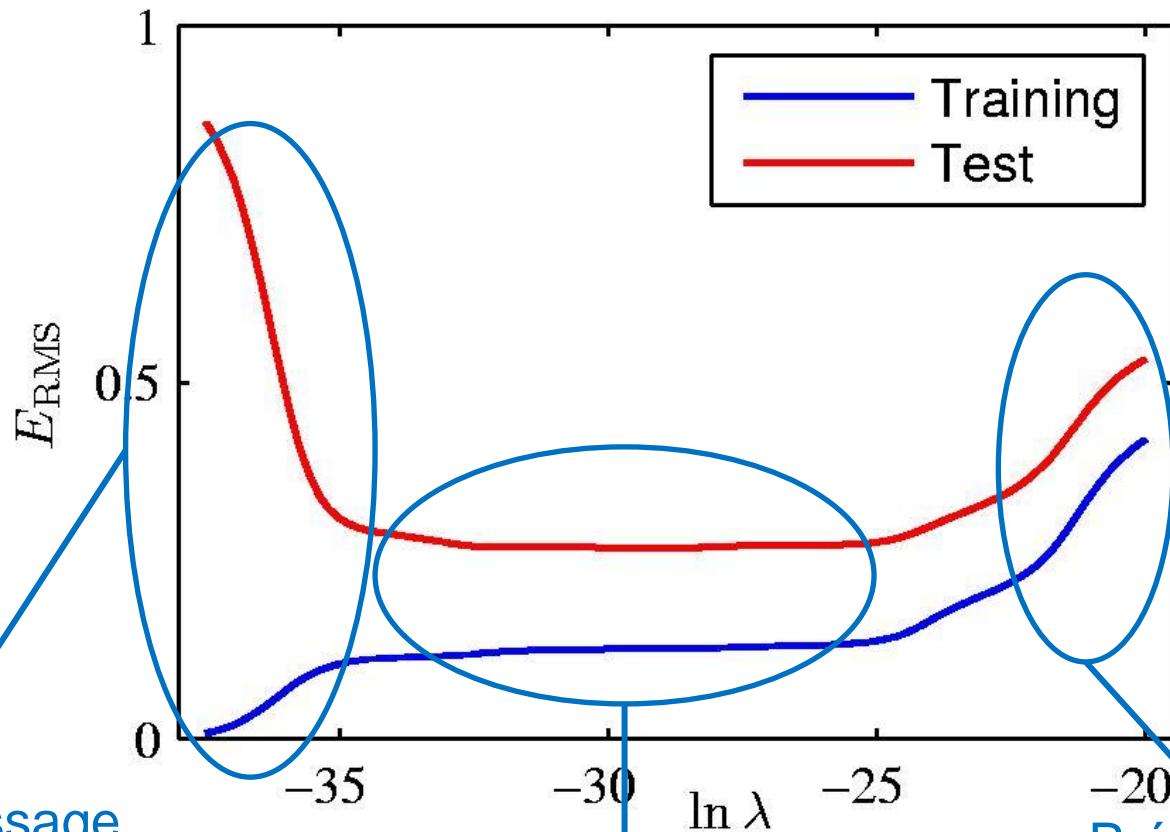
	$\ln \lambda = -\infty$	$\ln \lambda = -18$	$\ln \lambda = 0$
w_0^*	0.35	0.35	0.13
w_1^*	232.37	4.74	-0.05
w_2^*	-5321.83	-0.77	-0.06
w_3^*	48568.31	-31.97	-0.05
w_4^*	-231639.30	-3.89	-0.03
w_5^*	640042.26	55.28	-0.02
w_6^*	-1061800.52	41.32	-0.01
w_7^*	1042400.18	-45.95	-0.00
w_8^*	-557682.99	-91.53	0.00
w_9^*	125201.43	72.68	0.01



Sur-apprentissage

Régularisation: \mathcal{E}_{RMS} vs. $\ln(\lambda)$

$$\mathcal{E}_{\text{RMS}}(\mathbf{w}) = \frac{1}{2} \sum_{i=1}^N (D(\mathbf{x}_i, \mathbf{w}) - t_i)^2$$



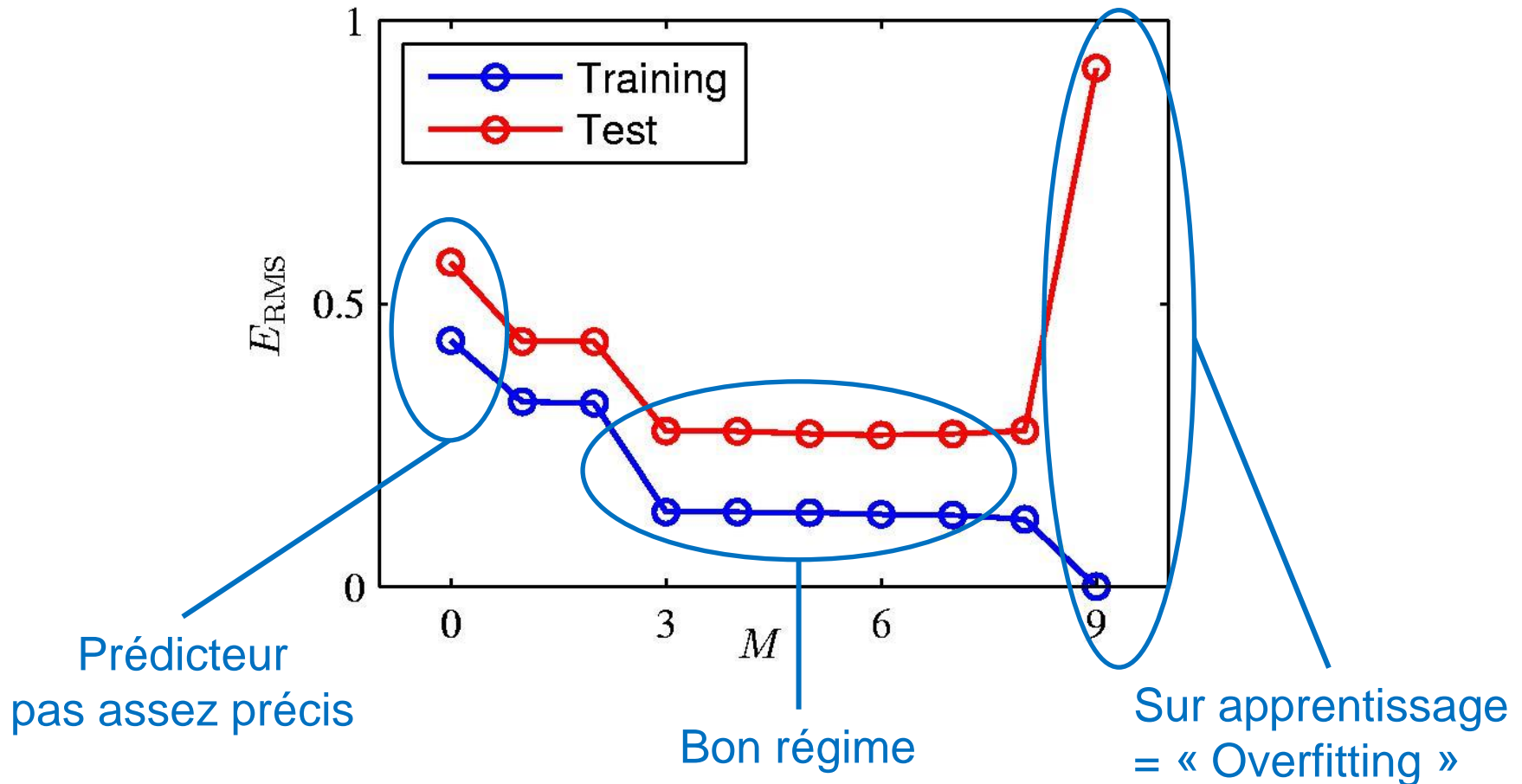
Sur apprentissage
= « Overfitting »

Bon régime

Prédicteur
pas assez précis

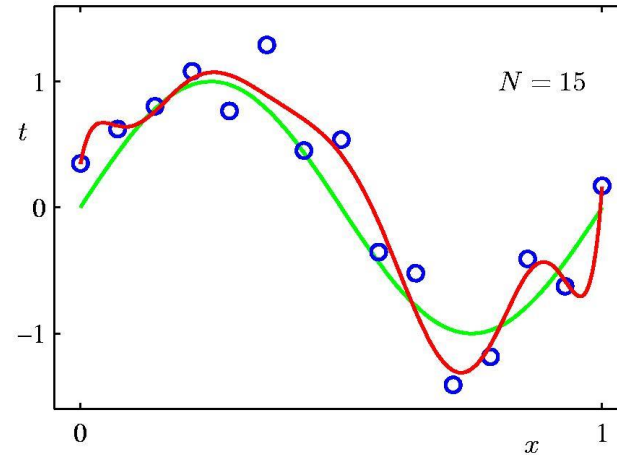
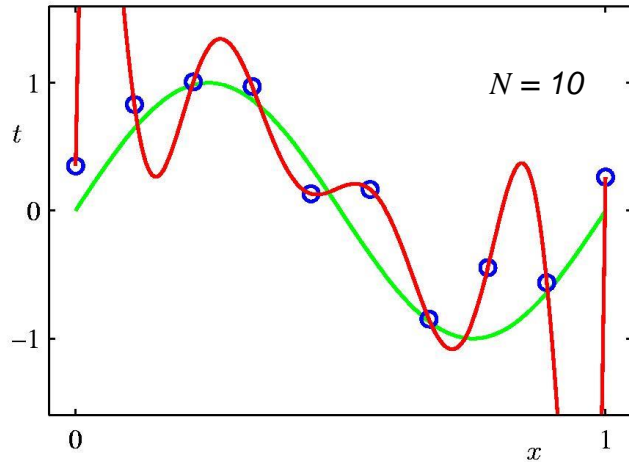
Régularisation: \mathcal{E}_{RMS} vs. $\ln(\lambda)$

$$\mathcal{E}_{\text{RMS}}(\mathbf{w}) = \frac{1}{2} \sum_{i=1}^N (D(\mathbf{x}_i, \mathbf{w}) - t_i)^2$$

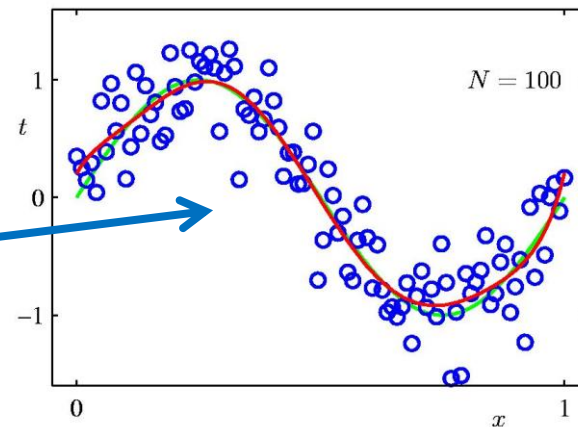


Influence de la quantité de données

Polynôme d'ordre 9



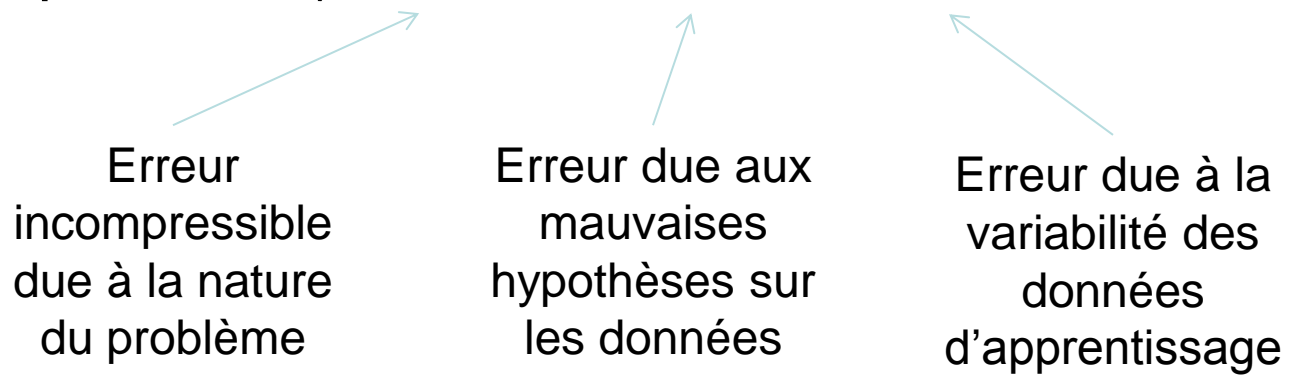
C'est aussi un moyen de
contrôler la régression



Compromis Biais-Variance

On peut montrer:

$$E(\text{erreur prédiction}) = \text{bruit}^2 + \text{biais}^2 + \text{variance}$$



Erreur
incompressible
due à la nature
du problème

Erreur due aux
mauvaises
hypothèses sur
les données

Erreur due à la
variabilité des
données
d'apprentissage

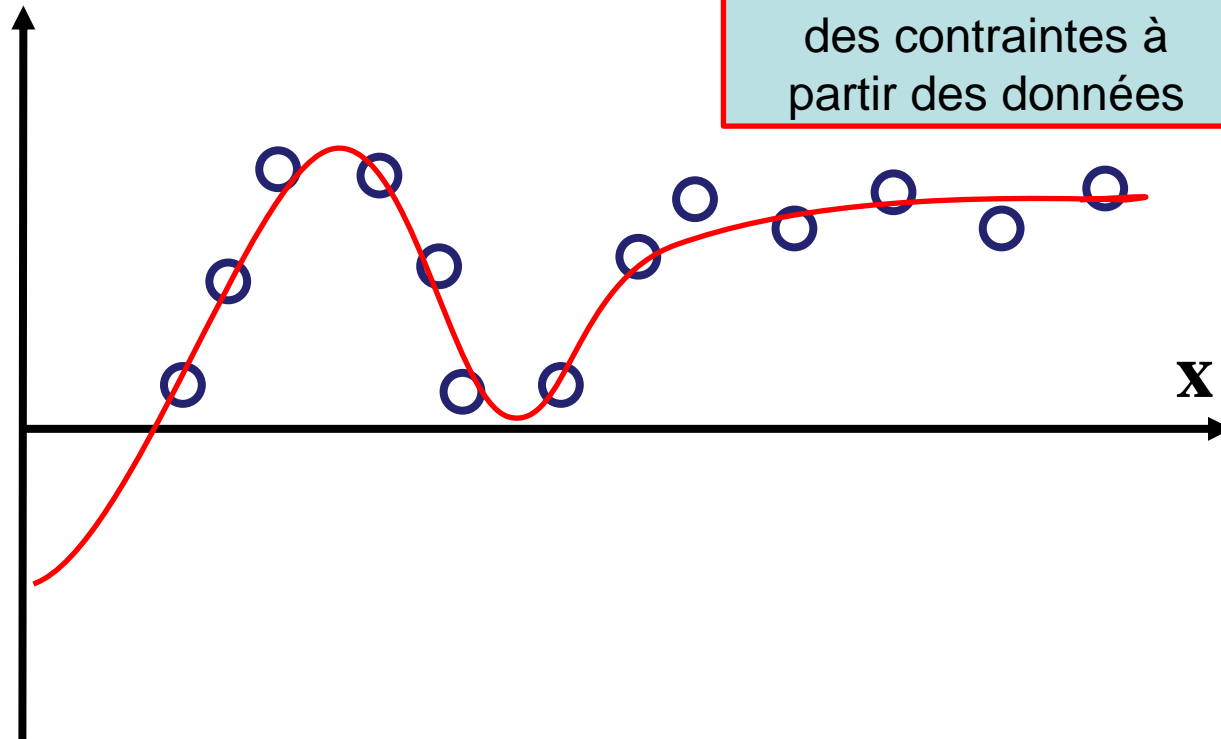
L'erreur de généralisation est un compromis entre
bonnes hypothèses sur les données et qualité des
données d'apprentissage

Erreur de généralisation

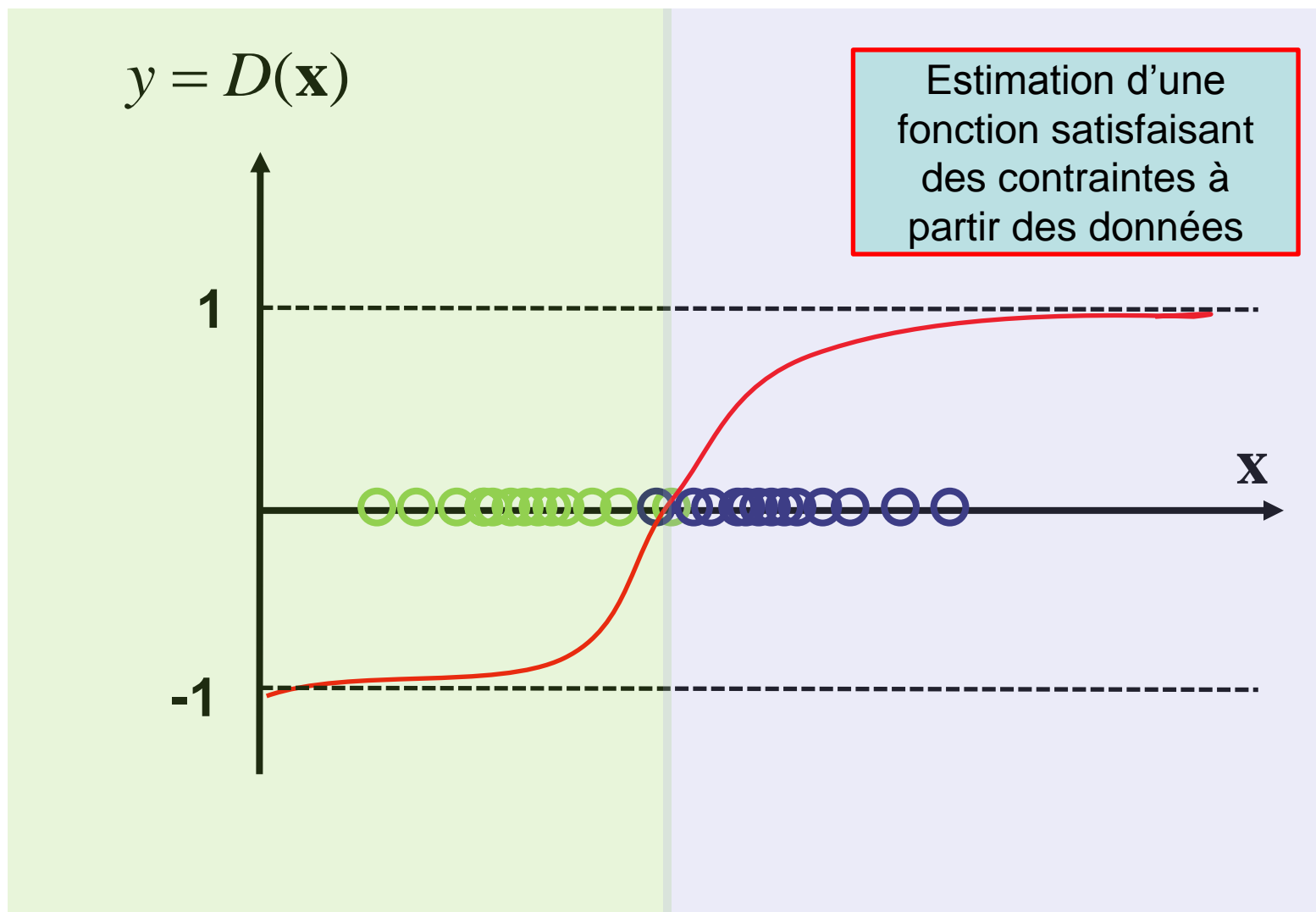
- Structure
 - **Biais:** écart entre hypothèse de modèle et « vraie » distribution des données
 - **Variance:** écarts générés par différents jeux d'apprentissage.
- Deux phénomènes à contrôler
 - **Simplisme:** modélisation trop grossière pour rendre compte de la variété des données
 - Biais++, Var –
 - Erreur d'apprentissage et de test grandes
 - **Sur-apprentissage (« Overfitting »):** modèle trop complexe se spécialisant sur les données d'apprentissage
 - Biais--, Var++
 - Ecart entre erreur d'apprentissage et erreur de test

Classification et Régression

$$y = R(\mathbf{x})$$



Classification et Régression



Critères statistiques pour la classification

- Risque ou erreur empirique

$$\mathcal{E}_{\text{train}}(\mathbf{w}, \mathcal{L}) = \frac{1}{N} \sum_{i=1}^N \{D(\mathbf{x}_i, \mathbf{w}) \neq y_i\}$$

- Erreur de généralisation (ou de test, ou idéale...)

$$\mathcal{E}_{\text{test}}(\mathbf{w}) = E_{\mathbf{x}, y} [\{D(\mathbf{x}, \mathbf{w}) \neq y\}]$$

- Critère à optimiser (forme assez générique)

$$\text{loss}(\mathbf{w}, \mathcal{L}) = \frac{1}{N} \sum_{i=1}^N l(D(\mathbf{x}_i, \mathbf{w}), y_i) + r(\mathbf{w})$$

Adéquation aux données

Régularisation

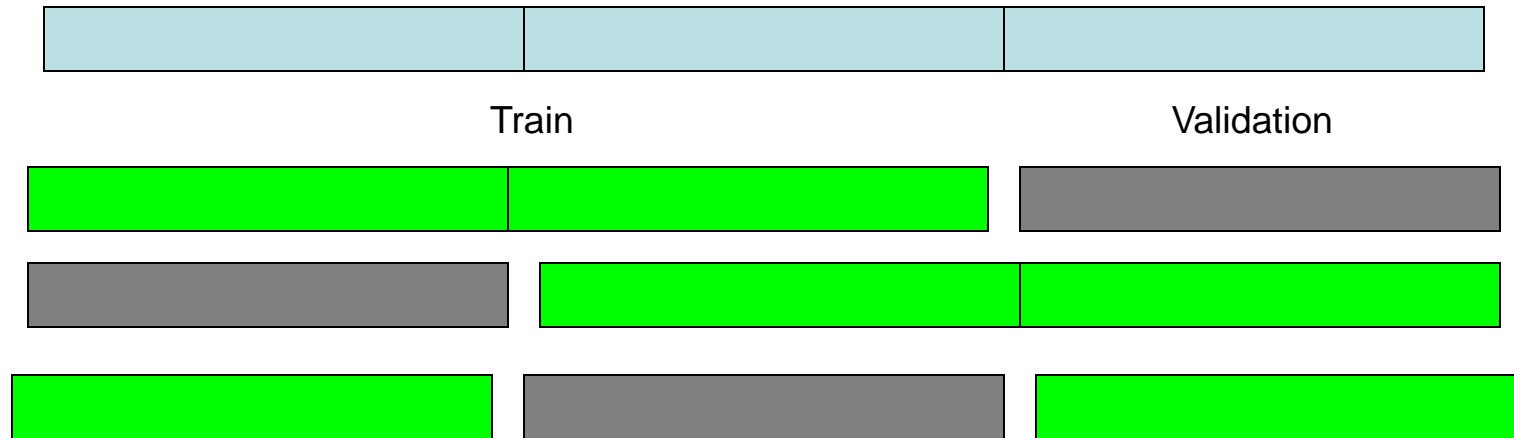
Validation croisée

- Permet d'estimer l'erreur de généralisation à partir des données d'apprentissage (« astuce »)
- Principe:
 - Division des données en k sous ensembles (« fold »)
 - Choix d'une partie comme ensemble de *validation* fictif, les autres comme *train*
 - Apprentissage sur l'ensemble *train*
 - Estimation des erreurs sur *validation*
- On fait tourner l'ensemble de *validation* sur chacune des parties
- L'erreur de généralisation estimée est la **moyenne** des erreurs sur chaque ensemble de *validation*

Stratégies de partitionnement

- k-fold

Données



- Leave-one-out



Rappel: différents types de données

$$\mathcal{L} = \{\mathbf{x}_i, y_i\}_{i=1}^N$$

- Apprentissage (« train »)
 - Exploité pour calculer le prédicteur à partir du critère « loss »
- Validation
 - Utilisé pour estimer l'erreur de généralisation et l'optimisation des hyper paramètres (λ) (par ex. par validation croisée)
- Evaluation (« test »)
 - Utilisé pour estimer l'erreur de généralisation une fois l'apprentissage achevé
 - NE PAS UTILISER POUR L'APPRENTISSAGE

Garantie théorique

Les données cachées

Les données disponibles

$$\boxed{\mathcal{E}_{test}} \leq \boxed{\mathcal{E}_{train}} + \left(\frac{h + h \log(2N / h) - \log(p / 4)}{N} \right)^{\frac{1}{2}}$$

Où N = nombre de données
 h = indicateur de complexité des classifieurs (VC dimension)
 p = probabilité que la borne soit fausse

En jouant sur la complexité des classes de classifieurs, on peut optimiser la borne d'erreur d'estimation.

Cette borne est plutôt lâche

En pratique, la démarche est plutôt « experte », et repose sur un certain savoir faire et une connaissance des données.