Chapitre 1: Les Concepts Fondamentaux

Ibrahima Sy

Institut Supérieur de Finance (ISF)
Data Science pour Actuaire
Master II en Actuariat

July 28, 2021

Plan

Motivation

- Comment développer une intelligence artificielle ?

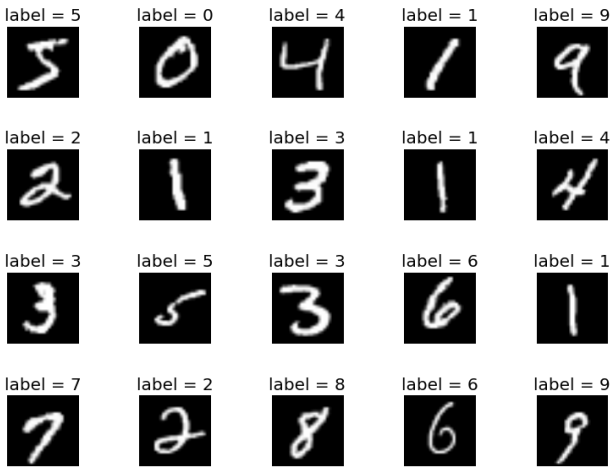


Figure 1: reconnaissance de chiffres manuscrits

- **Première Possibilité**: Par énumération de règles
  - Par énumération de règles : par exemple en faisant des hypothèse sur l'intensité des pixels , leurs position etc..
  - trop fastidieux, difficile de couvrir tous les cas d'espèce

Motivation

- **<u>Deuxième Possibilité</u>**: laisser l'ordinateur faire des essais et apprendre de ses erreurs
  - Machine Learning / Apprentissage Automatique : le domaine s'intéressant à l'étude de tels algorithmes
  - **Proposition de Définition :**
    L'apprentissage automatique(machine learning) ou apprentissage statistique est un champ d'étude de l'intelligence artificielle qui se fonde sur des approches mathématiques et statistiques pour donner aux ordinateurs la capacité ***d'apprendre*** *à partir de données, c'est-à-dire d'améliorer leurs performances à résoudre des tâches sans être explicitement programmés pour chacune. Plus largement, il concerne la conception, l'analyse, l'optimisation, le développement et l'implémentation de telles méthodes.*

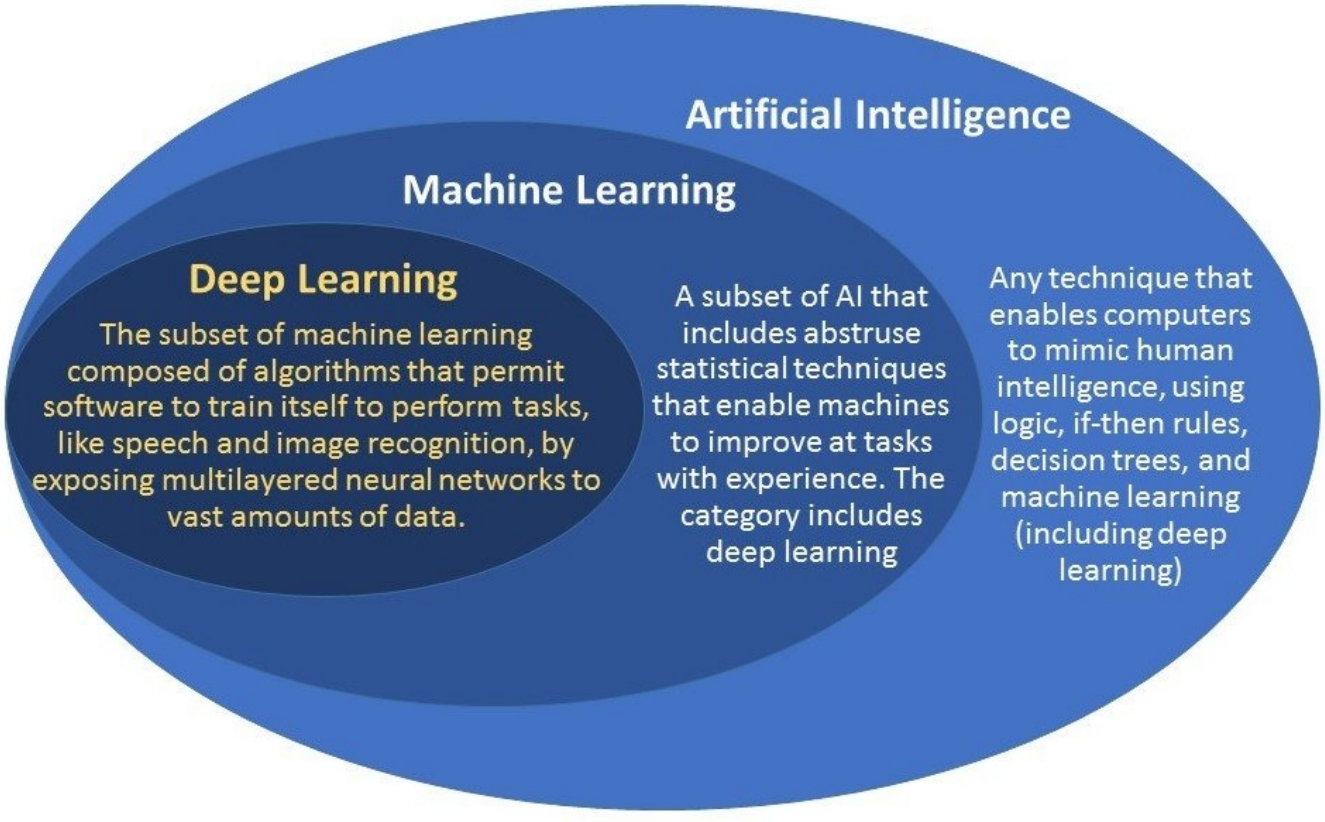
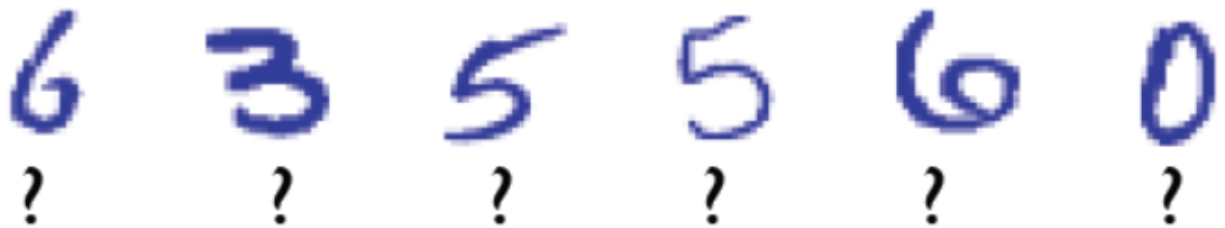


Figure 2: Intelligence Artificielle/Deep Learning/Machine Learning

Données d'entraînement vs. généralisation

- Les algorithmes d'apprentissage procèdent comme suit :
  - on fournit à l'algorithme des **données d'entraînement ...**

  - ... et l'algorithme retourne un «***programme***» capable de **généraliser** à de nouvelles données



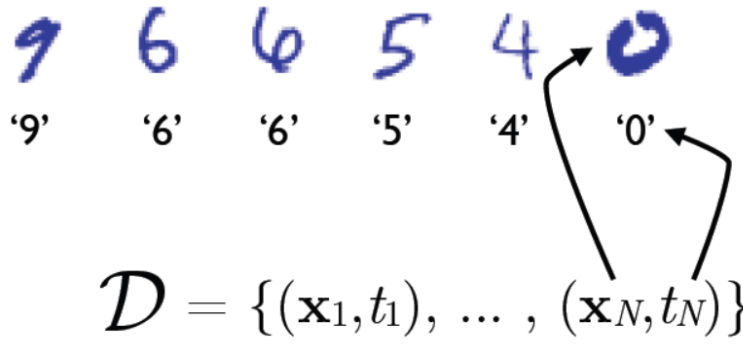
Ensemble d'entraînement, entrée, cible

- on note l'ensemble d'entraînement

$$\mathcal{D} = \left\{ (\mathbf{x_1}, t_1), (\mathbf{x_2}, t_2), \ldots, (\mathbf{x_N}, t_N) \right\}$$

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	househ
0	-122.23	37.88	41.0	880.0	129.0	322.0	126.0
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0
2	-122.24	37.85	52.0	1467.0	190.0	496.0	177.0
3	-122.25	37.85	52.0	1274.0	235.0	558.0	219.0
4	-122.25	37.85	52.0	1627.0	280.0	565.0	259.0
5	-122.25	37.85	52.0	919.0	213.0	413.0	193.0
6	-122.25	37.84	52.0	2535.0	489.0	1094.0	514.0
7	-122.25	37.84	52.0	3104.0	687.0	1157.0	647.0
8	-122.26	37.84	42.0	2555.0	665.0	1206.0	595.0

(fichier csv)



(images)

- on appelle $\mathbf{x_n}$ **une entrée** et t_n la **cible**

Modèle

- On note le «programme» généré par l'algorithme d'apprentissage $y(\mathbf{x})$
  - on va aussi appeler $y(\mathbf{x})$ un **modèle**

- $y(\mathbf{x})$ est une **fonction**

Ensemble de test

- L' objectif de L'apprentissage est la généralisation
- on utilise un **ensemble de test** $\mathcal{D}_{test}$ pour mesurer la performance de **généralisation** de notre modèle

Types d'apprentissage

il existe différents types d'apprentissage en machine learning

- apprentissage supervisé(**supervised learning**) : il y a une cible à prédire

$$\mathcal{D} = \left\{ (\mathbf{x_1}, t_1), (\mathbf{x_2}, t_2), \ldots, (\mathbf{x_N}, t_N) \right\}$$

- apprentissage non-supervisé(**unsupervised learning**) : cible n'est pas fournie

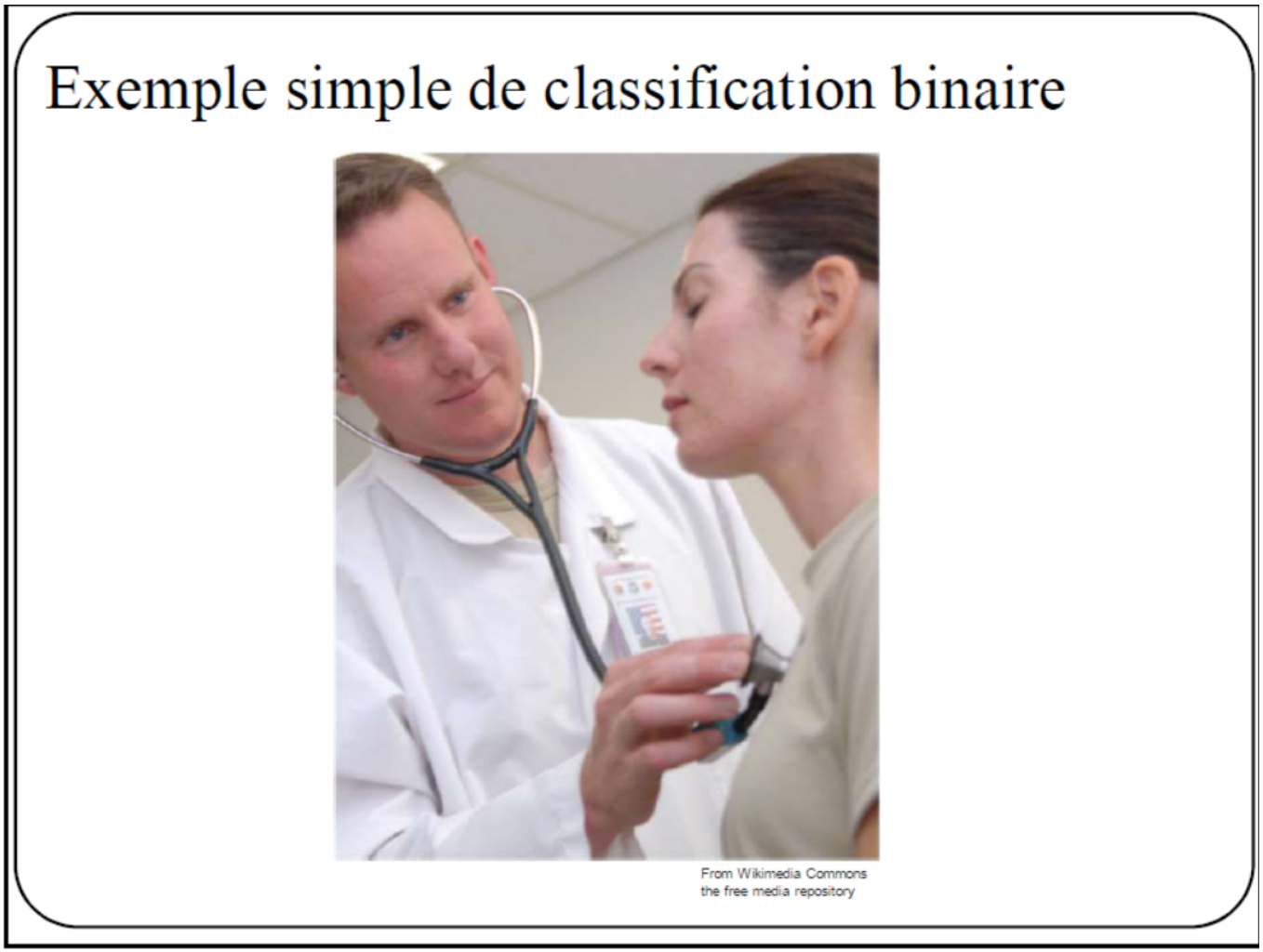
$$\mathcal{D} = \left\{ \mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_N} \right\}$$

- apprentissage par renforcement

Apprentissage supervisé, classification, régression

- L'apprentissage supervisé est lorsqu'on a une cible à prédire
  - **classification** : la cible est un indice de classe $t \in \{1, 2, ...K\}$
    - exemple : reconnaissance de caractères
      - $\mathbf{x}$: vecteur des intensités de tous les pixels de l'image
      - t : identité du caractère
  - **régression** : la cible est un nombre réel $t \in \mathbb{R}$
    - exemple : prédiction de la valeur d'une action à la bourse
      - $\mathbf{x}$: vecteur contenant l'information sur l'activité économique de la journée
      - t : valeur d'une action à la bourse le lendemain

Exemple simple de classification binaire

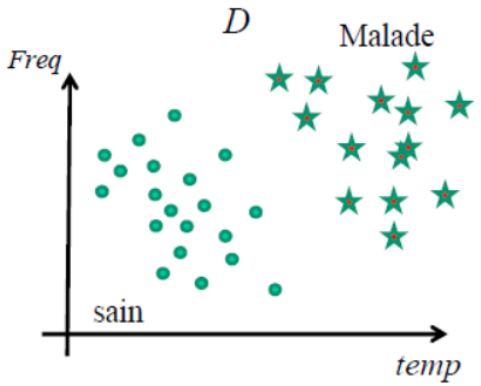


Exemple simple de classification binaire

Exemple simple de classification binaire

D

	(temp, freq)	diagnostique
Patient 1	(37.5, 72)	Sain
Patient 2	(39.1, 103)	Malade
Patient 3	(38.3, 100)	Malade
	(…)	…
Patient N	(36.7, 88)	Sain
	$\vec{x}$	t

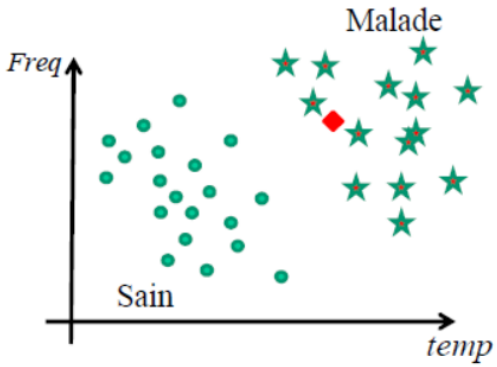


Exemple simple de classification binaire

Exemple simple de classification binaire

Un nouveau patient vient à l'hôpital
Comment peut-on en déduire son état?

From Wikimedia Commons
the free media repository

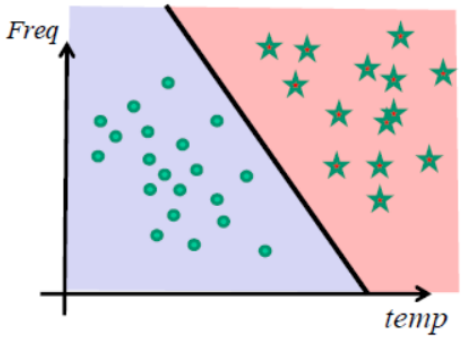


Exemple simple de classification binaire

Solution

From Wikimedia Commons
the free media repository

Diviser l'espace des caractéristiques en deux régions : sain et malade

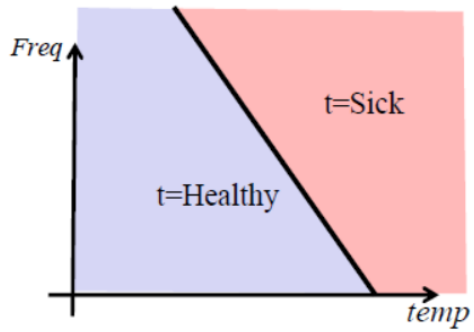


Exemple simple de classification binaire

Plus formellement

From Wikimedia Commons
the free media repository

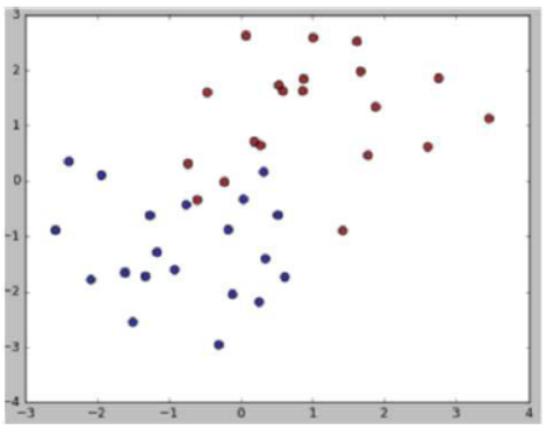
$$y_{\vec{w}}(\vec{x}) = \begin{cases} Sain \text{ si } \vec{x} \text{ est dans la région bleue} \\ Malade \quad \text{sinon} \end{cases}$$



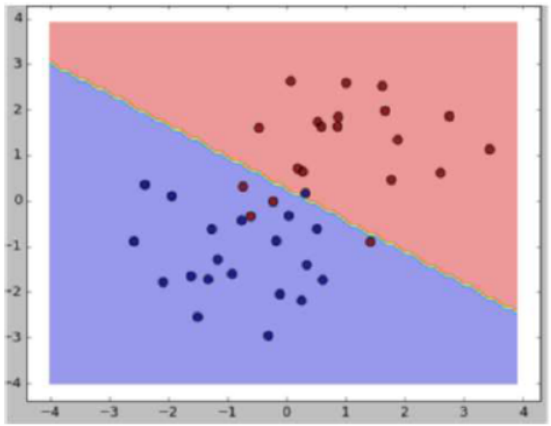
Exemple simple de classification binaire

Exemple de classification

Cas 2 classes



Soit des données issues de 2 classes ● et ●
dans un espace à 2 dimensions



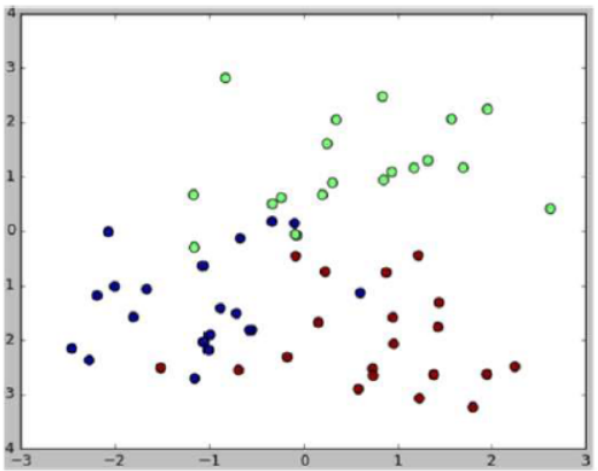
Une fois l'entraînement terminé

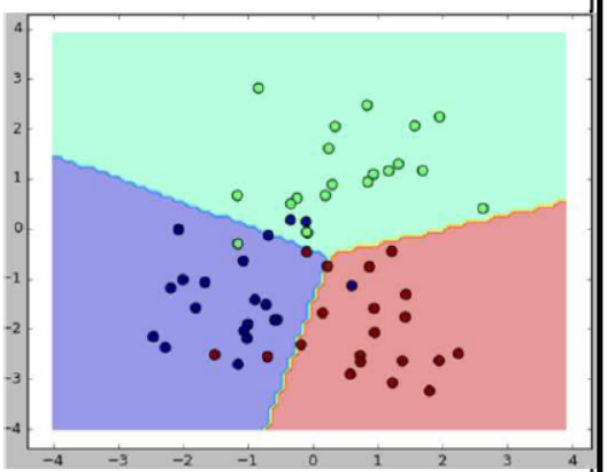
y(●) = class 1
y(●) = class 2

Exemple simple de classification binaire

Exemple de classification

Cas 3 classes





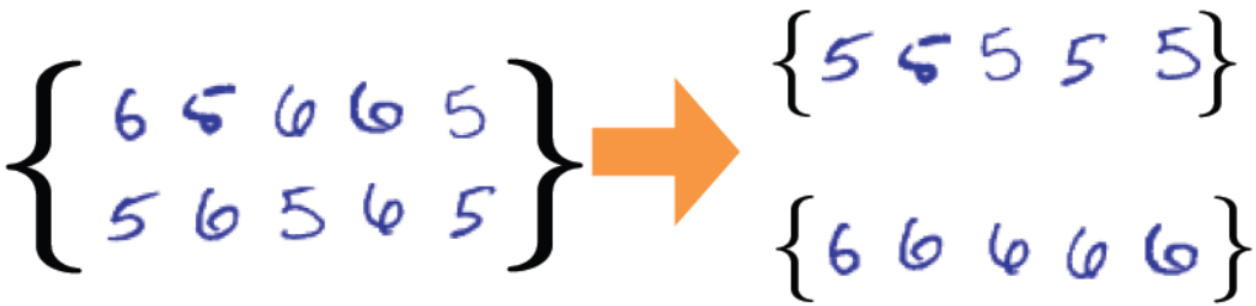
Soit des données issues de 3 classes ●, ●, ●
dans un espace à 2 dimensions

Une fois l'entraînement terminé

y(●) = class 1
y(●) = class 2
y(●) = class 3

Apprentissage non-supervisé, partitionnement

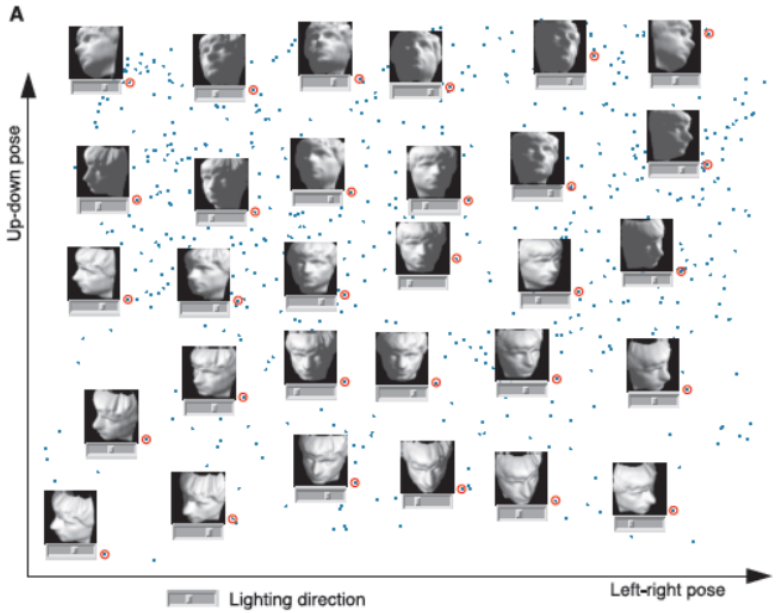
- L'apprentissage non-supervisé est lorsqu'une cible n'est pas explicitement donnée
  - **partitionnement de données / clustering**



Apprentissage non-supervisé, visualisation

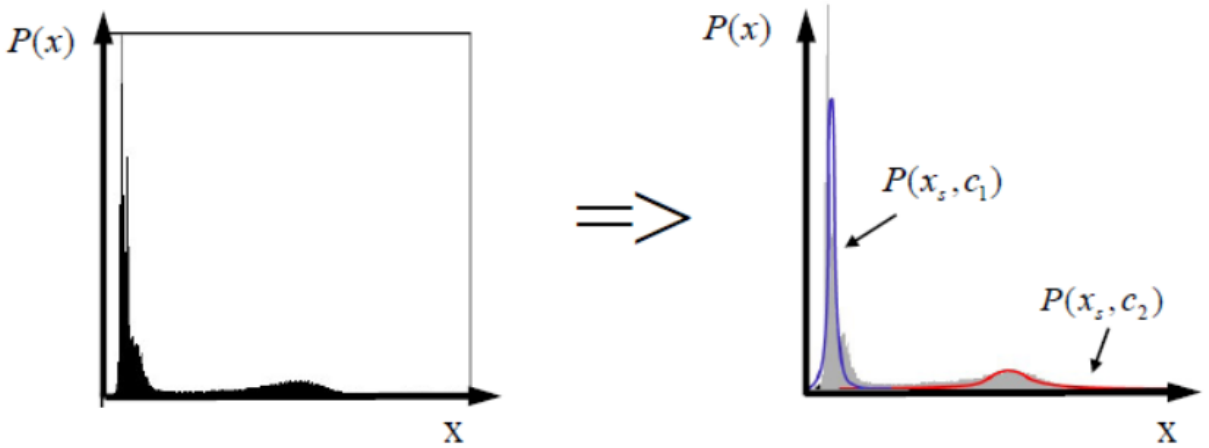
- **visualisation de données**

Tenenbaum, de Silva, Langford, (2000)



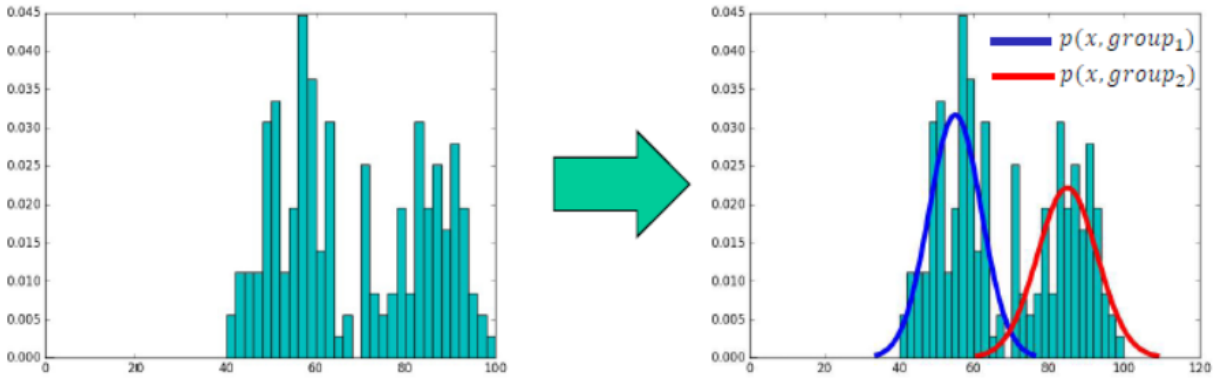
Apprentissage non-supervisé, estimation de densité

- **C'est a dire apprendre la loi de probabilité $p(x)$ dont les données sont issues**



Apprentissage non-supervisé, estimation de densité

- **Exemple : trouver 2 groupes d'étudiants suite à un examen**

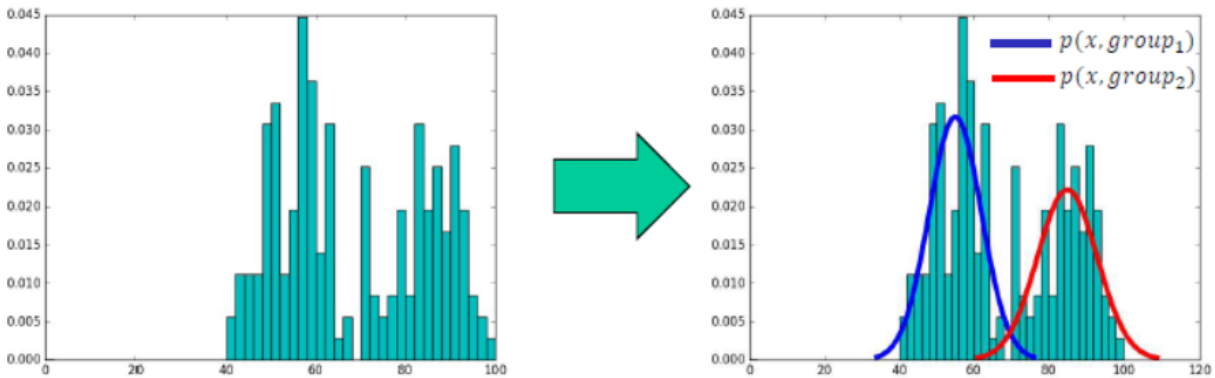


Apprentissage non-supervisé, estimation de densité

- **Autres Applications**
  - pour générer de nouvelles données réalistes
  - pour distinguer les «vrais» données des «fausses» données (spam filtering)
  - compression de données

Apprentissage non-supervisé, estimation de densité

- **Exemple : trouver 2 groupes d'étudiants suite à un examen**

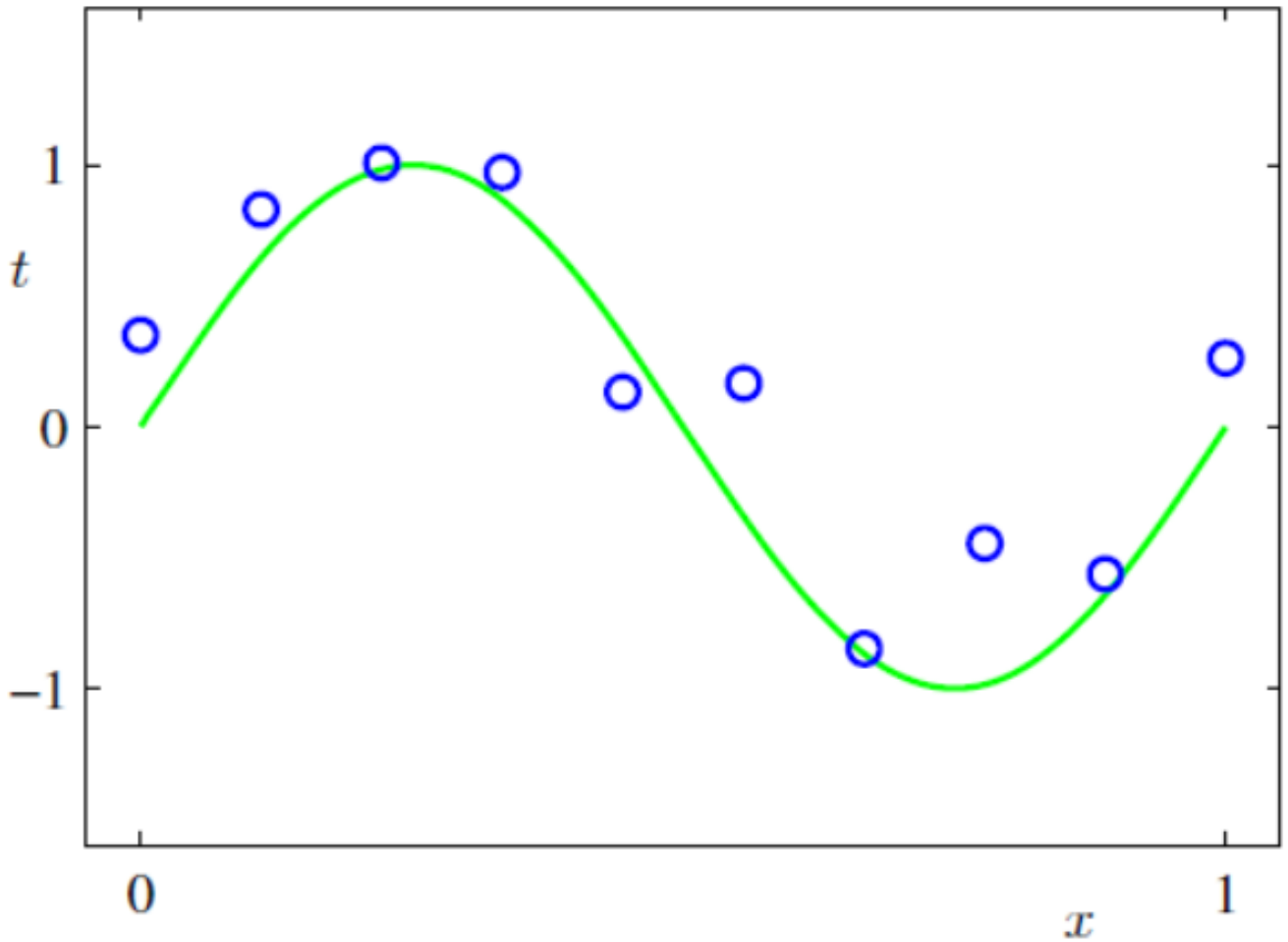


- **Autres Applications**
  - pour générer de nouvelles données réalistes
  - pour distinguer les «vrais» données des «fausses» données (spam filtering)
  - compression de données

Régression en $1D$

- **Exemple simple: régression en une dimension :**
  - entrée : scalaire x
  - cible : scalaire t
- **Données d'entrainement $\mathcal{D}$ contiennent:**
  - $\mathbf{X} \equiv (x_1, \ldots, x_N)^T$
  - $\mathbf{t} \equiv (t_1, \ldots, t_N)^T$
- **Objectif:**
  - faire une prédiction $\hat{t}$ pour une nouvelle entrée $\hat{x}$

Régression en $1D$



Régression polynomiale, modèle

- On va supposer qu'une bonne prédiction aurait une forme polynomiale

$$y(x, \mathbf{w}) = \omega_0 x + \omega_2 x^2 + \cdots + \omega_M x^M$$

- $y(x, \mathbf{w})$ est notre modèle
  - représente nos hypothèses sur le problème à résoudre
  - a normalement des paramètres, qu'on doit trouver $(\omega_0, \omega_1, \ldots, \omega_M)$
- On peut voir un modèle comme un programme définit mathématiquement

```
def predict(x,w):
    x_poly = x ** np.arange(len(w))
    return np.dot(x_poly,w)
```

Minimisation de perte (côut, erreur)

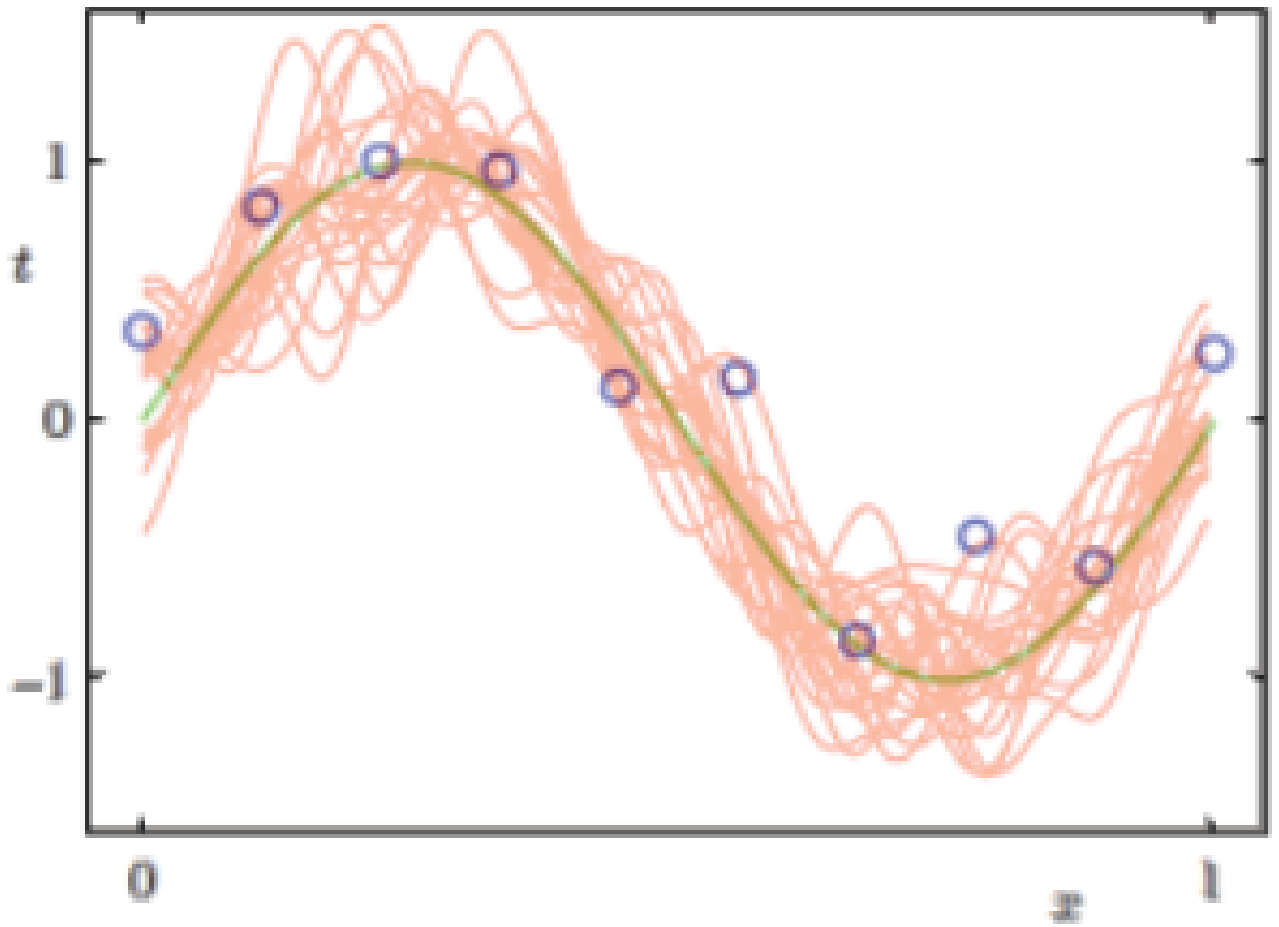
- Comment trouver $\mathbf{w}$? (**c'est un problème d'optimisation**)
  - On cherche le $\mathbf{w}^*$ qui minimise la somme de notre perte / erreur / côut sur l'ensemble d'entrainement

$$\mathbb{E}(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\{y(x_n, \mathbf{w}) - t_n\}^2$$

  - le terme $\ll \frac{1}{2} \gg$ permet juste de simplifier les calculs mais n'a pas un rôle capital
- Un algorithme d'apprentissage résoudrait ce problème
  - à partir des données, il va retourner $\mathbf{w}^*$
  -

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \mathbb{E}(\mathbf{w})$$

Minimisation de perte (côut, erreur)



Sur-apprentissage / sous-apprentissage

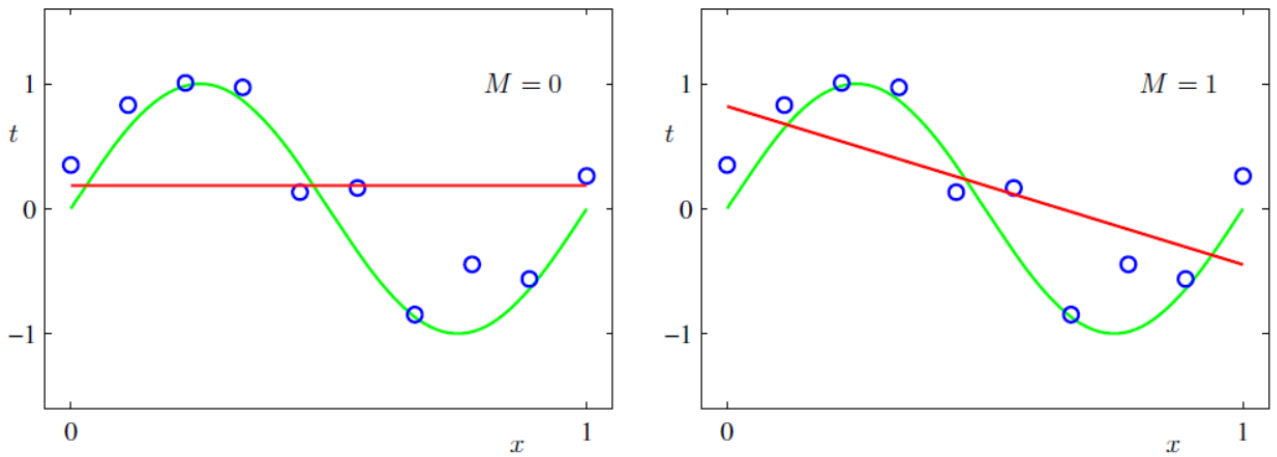
- Comment trouver le bon $\mathbf{M}$?

  Le problème avec les **hyper-paramètres** est qu'ils ne peuvent pas être estimés à l'aide des algorithmes d'optimisation classiques (**descente de gradient, méthode de Newton, etc.**) comme pour les paramètres .

  Par conséquent, on fixe souvent « à la main » les hyper-paramètres.

Sous-apprentissage (underfitting)

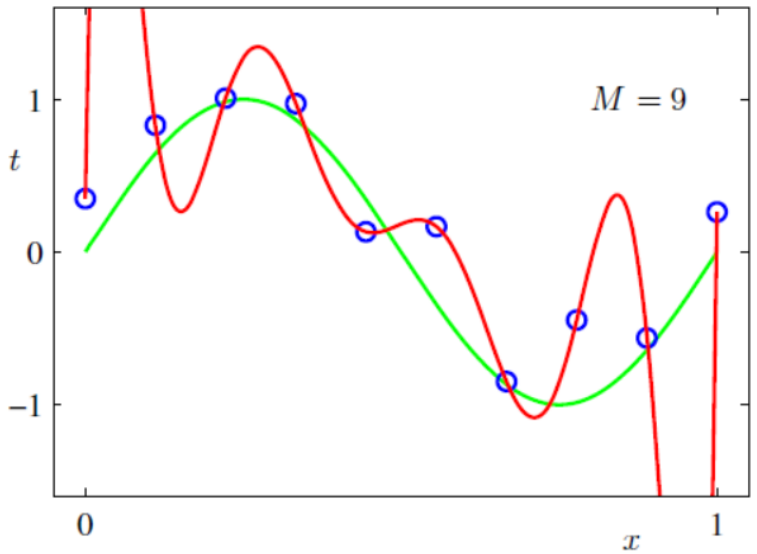
- Comment trouver le bon **M** ?
  Un petit **M** donne un modèle trop simple causant du **sous-apprentissage**



- Erreur sur l'ensemble entrainement est élevé
- Erreur sur ensemble de test est élevé

Sur-apprentissage (overfitting)

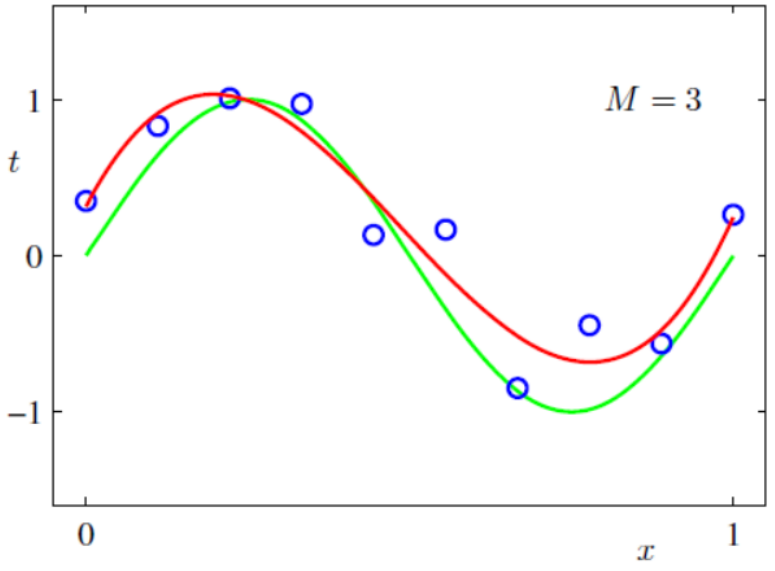
- Un grand **M** donne un modèle qui « apprend par coeur » les données d'apprentissage ce qui cause du **sur-apprentissage**



- Erreur sur l'ensemble entrainement est **faible**
- Erreur sur ensemble de test est **élevé**

Sélection de modèle

- on voudrait une valeur intermédiaire qui permet de retrouver la tendance générale de la relation entre x et t, sans le bruit
- c'est ce qui va permettre de bien généraliser à de nouvelles entrées !
- trouver cette meilleure valeur de M s'appelle de la **sélection de modèle**

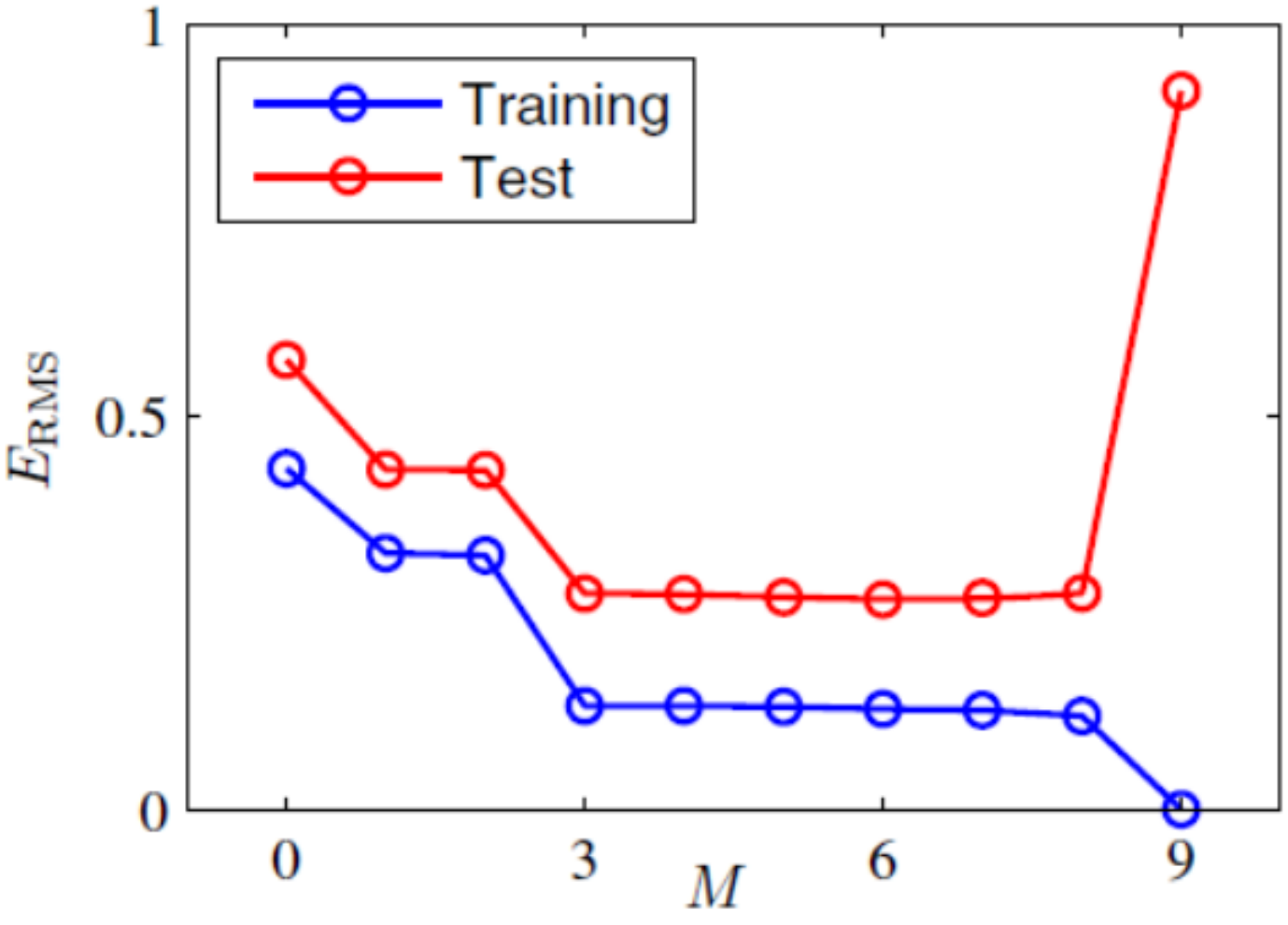


Capacité d'un modèle, performance

- **Capacité** d'un modèle
  - aptitude d'un modèle à apprendre «par coeur»
  - exemple : plus M est grand, plus le modèle a de capacité
- Plus la capacité est grande, plus la différence entre l'erreur d'entraînement et l'erreur de test augmente
  - *En régression, l'erreur sur tout un ensemble est souvent mesurée par la racine de la moyenne des erreurs au carré (root-mean-square error)*

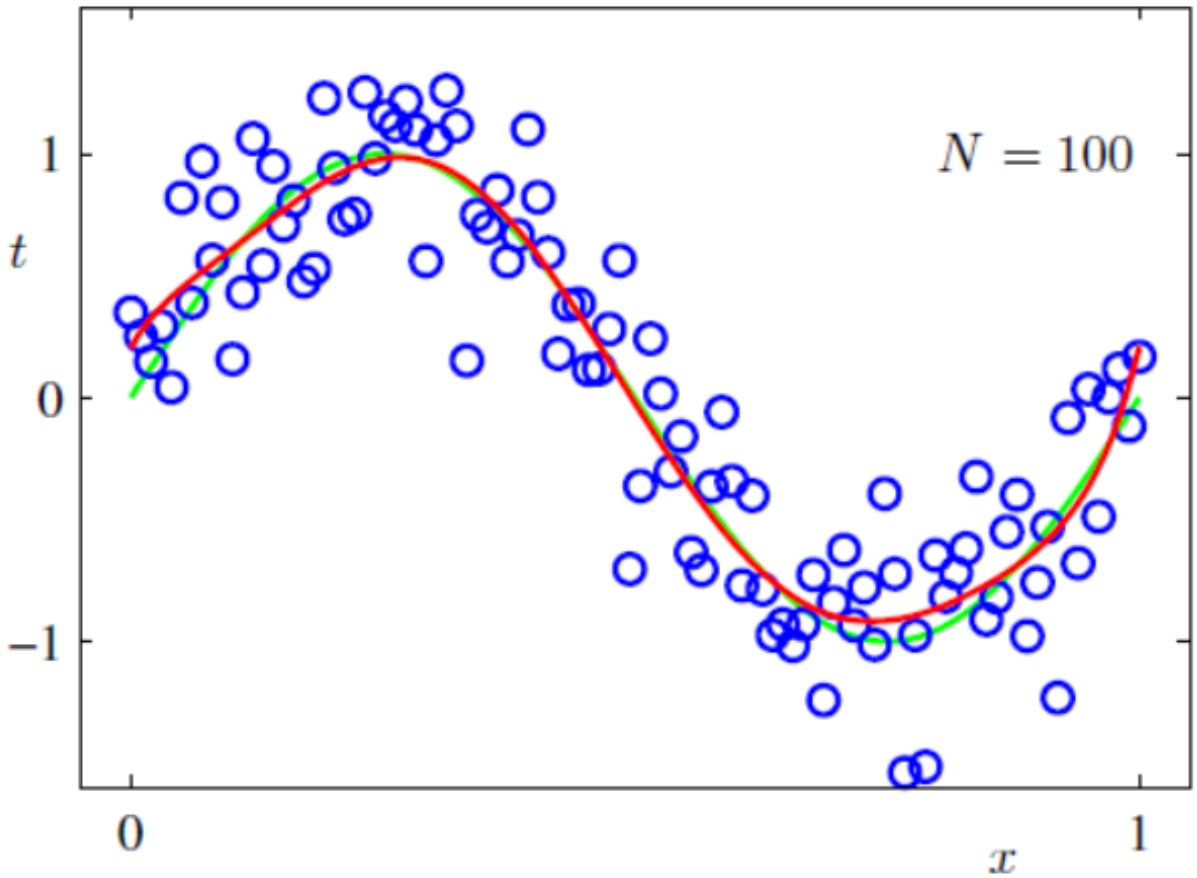
$$\mathbb{E}_{RMS} = \sqrt{\frac{2 \times \mathbb{E}(\mathbf{w})}{N}}$$

Capacité d'un modèle, performance



Généralisation vs. quantité de données

- Plus la quantité de données d'entraînement augmente, plus le modèle entraîné va bien généraliser



Régularisation

- Lorsqu'on souhaite éviter qu'on modèle **sur-apprenne** nous avons ces possibilités
  - On choisit un petit « **M** »
  - On réduit la capacité du modèle par **régularisation**: permettant l'utilisation des valeurs de « **M** » élevées

$$\mathbb{E}(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\{y(x_n, \mathbf{w}) - t_n\}^2 + \frac{\lambda}{2}||\mathbf{w}||^2$$

λ :contrôle la capacité du modèle

$||\mathbf{w}|| = \omega_1^2 + \omega_2^2 + \cdots + \omega_M^2$

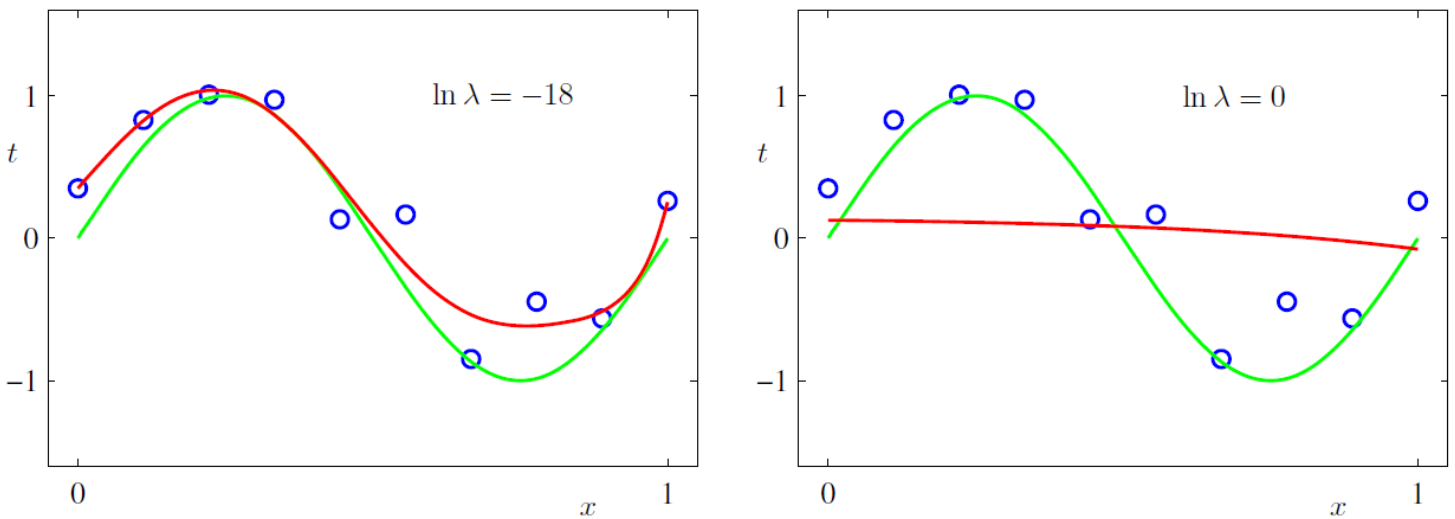
Régularisation : Sans Régularisation

- Simulation du modèle de régression polynomiale sur différent valeurs de **M**
- $\mathbf{w}^*$: valeurs de **w** après entrainement du modèle

	$M=0$	$M=1$	$M=6$	$M=9$
$w_0^\star$	0.19	0.82	0.31	0.35
$w_1^\star$		-1.27	7.99	232.37
$w_2^\star$			-25.43	-5321.83
$w_3^\star$			17.37	48568.31
$w_4^\star$				-231639.30
$w_5^\star$				640042.26
$w_6^\star$				-1061800.52
$w_7^\star$				1042400.18
$w_8^\star$				-557682.99
$w_9^\star$				125201.43

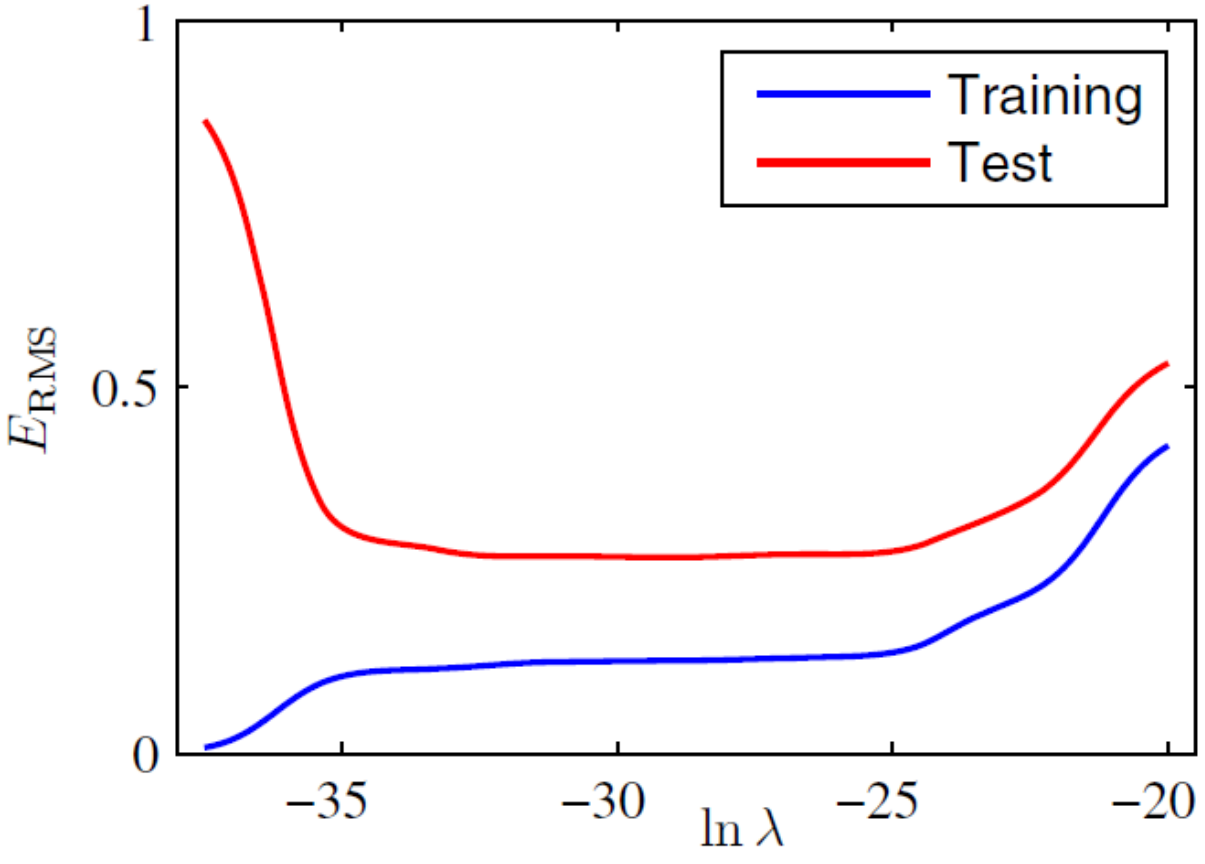
Régularisation

- Plus la régularisation augmente (λ augmente) ,plus la capacité du modèle diminue



Régularisation

- Comme avec M , les variations de **M** influence sur l'erreur entrainement et de test



Notion d'hyper-paramètres

- **d'hyper-paramètres** : ce sont les paramètres qui permettent de contrôler le processus d'apprentissage
- Dans le cadre de la régularisation précédente

$$\mathbb{E}(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\{y(x_n, \mathbf{w}) - t_n\}^2 + \frac{\lambda}{2}||\mathbf{w}||^2$$

  - λ et $\mathbf{M}$ sont des des hyper-paramètres
  - **Qu'on doit déterminer avant l'apprentissage**

Comment déterminer les bons hyper-paramètres ?

C'est dire λ et $\mathbf{M}$ en régression polynomiale

Pourquoi on devrait pas procéder comme suit ?

- **Très mauvaise solution** : choisir au hasard
- **Mauvaise solution** : prendre plusieurs paires $(\mathbf{M}, \lambda)$ et garder celle dont l'erreur d'entraînement est la plus faible
  - Sur-apprentissage
- **Mauvaise solution** : prendre plusieurs paires $(\mathbf{M}, \lambda)$et garder celle dont l'erreur de test est la plus faible
  - $\mathcal{D}_{test}$ ne doit pas être utilisé pour entraîner le modèle

Bonne pratique : prendre plusieurs paires $(\mathbf{M}, \lambda)$ et garder celle dont l'erreur de validation est la plus faible

Validation croisée (cross-validation)

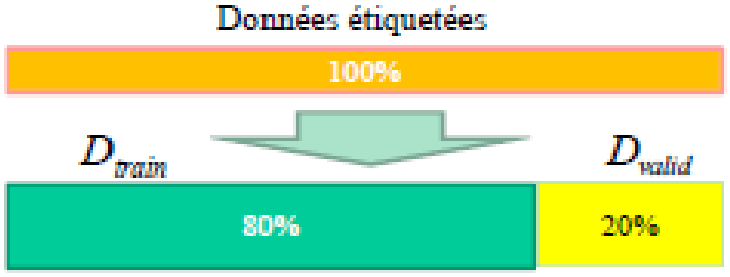
- **Option I** : on réserve des données d'entraînement pour comparer différentes valeurs
  - garde la majorité pour l'ensemble d'entraînement $\mathcal{D}_{train}$ (ex: 80 %)
  - le reste $\mathcal{D}_{val}$ (ex: 20 %) servira á comparer les hyper-paramètres

$\mathcal{D}_{val}$: ensemble de validation

Validation croisée (cross-validation)

Validation croisée *(cross-validation)*

1- Diviser au hasard les données d'entraînement en 2 groupes



2- Pour M allant de M_{min} à M_{max}
Pour λ allant de λ_{min} à λ_{max}

Entraîner le modèle sur D_{train}
Calculer l'erreur sur D_{valid}

3- Garder la paire (M, λ) dont **l'erreur de validation** est la plus faible

Validation croisée K fois (k-fold cross-validation)

Option II

- Lorsqu'on a peu de données, 20 % est trop peu pour estimer la performance de généralisation
- On pourrait répéter la procédure de séparation train/valid plus d'une fois
- ***k*-fold cross-validation** : divise les données en S portions différentes
- chaque portion est utilisée une fois en tant que $\mathcal{D}_{valid}$

Validation croisée K fois (k-fold cross-validation)

Exemple : Avec $k = 4$

Validation croisée K fois (k-fold cross-validation)

Validation croisée K fois *(k-fold cross-validation)*

Pour M allant de M_{min} à M_{max}
  Pour λ allant de λ_{min} à λ_{max}
    Pour j allant de 0 à K

      Diviser au hasard les données d'entraînement => $D_{train} D_{valid}$

      Entraîner le modèle sur D_{train}
      Calculer l'erreur sur D_{valid}

Garder la paire (M, λ) dont **l'erreur de validation MOYENNE** est la plus faible

- Si $k = N$: on parle alors de méthode **leave-one-out**

recherche sur une grille

- Comment déterminer la liste des valeurs d'hypermètres à comparer
- **recherche sur une grille** (grid search) :
  - détermine une liste de valeur pour chaque hyper-paramètre
  - construit la liste de toutes les combinaisons possibles

```
>>> M = [1,2]
>>> lba = [0,1e-6,1e-3]
>>> hypers = [ [ (m,l) for m in M ] for l in lba ]
>>> print hypers
[[(1, 0), (2, 0)], [(1, 1e-06), (2, 1e-06)], [(1, 0.001),
(2, 0.001)]]
```

References I

- **Hugo Larochelle**, Professeur associé, Université de Montréal, Google
- **Pierre-Marc Jodoin**, Professeur titulaire Université Sherbrooke
- Bayesian Reasoning and Machine Learning de David Barber
- The Elements of Statistical Learning de Trevor Hastie,
- Robert Tibshirani et Jerome Friedman
- Information Theory, Inference, and Learning Algorithms de David J.C. MacKay
- Convex Optimization de Stephen Boyd et Lieven Vandenberghe
- Natural Image Statistics de Aapo Hyvärinen, Jarmo Hurri et Patrik O. Hoyer
- The Quest for Artificial Intelligence - A History of Ideas and Achievements de Nils J. Nilsson
- Gaussian Processes for Machine Learning de Carl Edward Rasmussen et Christopher K. I. Williams
- Introduction to Information Retrieval de Christopher D. Manning, Prabhakar Raghavan et Hinrich Schütze