

Théorie de l'apprentissage

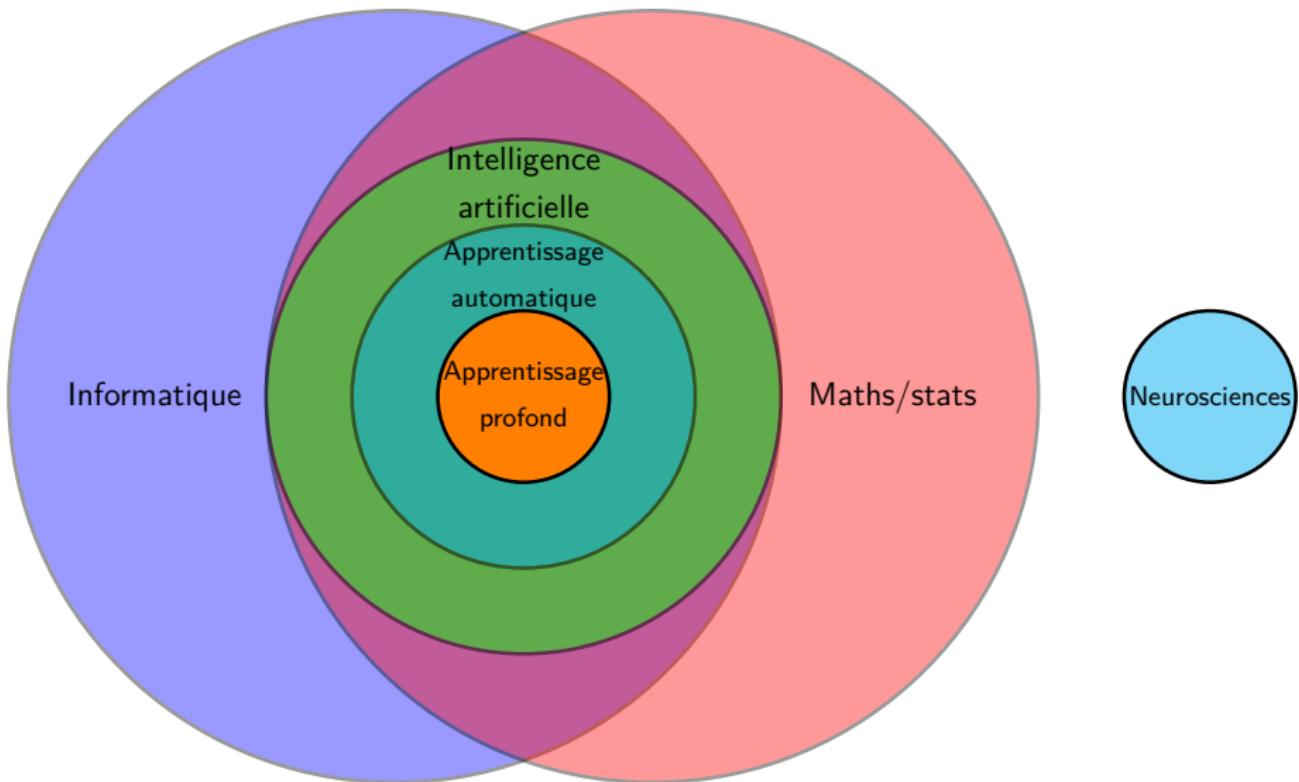
De l'apprentissage automatique à l'apprentissage profond

Nicolas Audebert nicolas.audebert@ign.fr

Géodata Paris

16 février 2026

Apprentissage automatique, IA, informatique



Qu'est-ce que l'apprentissage automatique ?

Objectif : automatiser une prédiction à partir de valeurs observées, uniquement à partir d'exemples (\approx méta-heuristique)

Deux grandes familles d'approche :

- ▶ Symbolique : systèmes experts, arbres de décision, logique floue, ontologies
 - ▶ Formalisation de connaissances expertes encodées dans le système
 - ▶ Règles de logiques, par ex. déductives pour tirer des inférences
 - ▶ Interprétable et permet d'intégrer des connaissances humaines
 - ▶ Limité à des tâches pour lesquelles on peut encoder beaucoup d'information existante
- ▶ Connexionniste : réseaux de neurones, inférence bayésienne
 - ▶ Modélisation d'une distribution statistique entre entrées et sorties
 - ▶ Nécessite beaucoup d'exemples
 - ▶ Pas facilement interprétable
 - ▶ Mais flexible et adaptable à de nombreux problèmes

Qu'est-ce que l'apprentissage automatique ?

Objectif : automatiser une prédiction à partir de valeurs observées, uniquement à partir d'exemples (\approx méta-heuristique)

Deux grandes familles d'approche :

- ▶ Symbolique : systèmes experts, arbres de décision, logique floue, ontologies
 - ▶ Formalisation de connaissances expertes encodées dans le système
 - ▶ Règles de logiques, par ex. déductives pour tirer des inférences
 - ▶ Interprétable et permet d'intégrer des connaissances humaines
 - ▶ Limité à des tâches pour lesquelles on peut encoder beaucoup d'information existante
- ▶ **Connexionniste : réseaux de neurones, inférence bayésienne**
 - ▶ Modélisation d'une distribution statistique entre entrées et sorties
 - ▶ Nécessite beaucoup d'exemples
 - ▶ Pas facilement interprétable
 - ▶ Mais flexible et adaptable à de nombreux problèmes

Apprentissage automatique à l'IGN

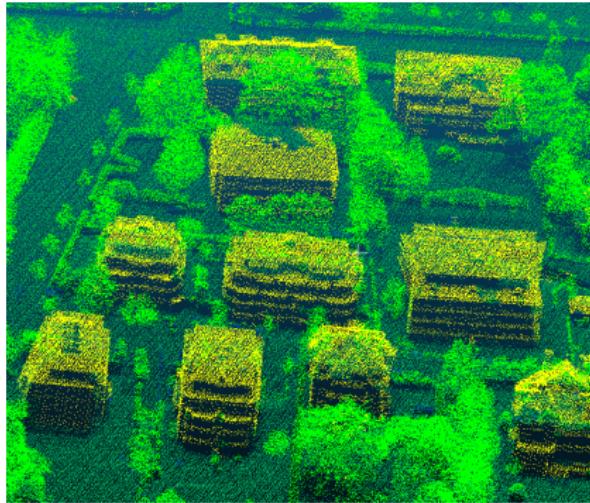
- ▶ Cartographie d'occupation des sols à partir d'images satellites
- ▶ Sémantisation de nuages de points massifs
- ▶ Numérisation/vectorisation de cartes historiques
- ▶ Reconnaissances des espèces d'arbres
- ▶ et bien d'autres



<https://geoservices.ign.fr/ressources-ia-de-couverture-du-sol>

Apprentissage automatique à l'IGN

- ▶ Cartographie d'occupation des sols à partir d'images satellites
- ▶ Sémantisation de nuages de points massifs
- ▶ Numérisation/vectorisation de cartes historiques
- ▶ Reconnaissances des espèces d'arbres
- ▶ et bien d'autres



<https://github.com/IGNF/myria3d>

Apprentissage automatique à l'IGN

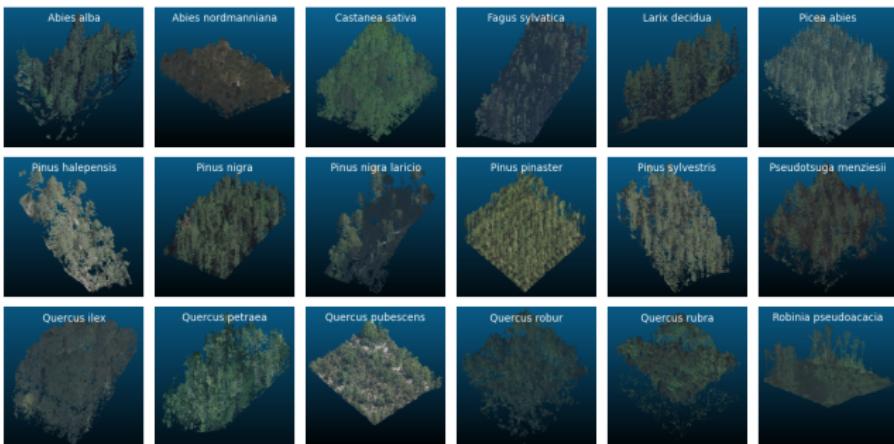
- ▶ Cartographie d'occupation des sols à partir d'images satellitaires
- ▶ Sémantisation de nuages de points massifs
- ▶ Numérisation/vectorisation de cartes historiques
- ▶ Reconnaissances des espèces d'arbres
- ▶ et bien d'autres



<https://icdar21-mapseg.github.io/>

Apprentissage automatique à l'IGN

- ▶ Cartographie d'occupation des sols à partir d'images satellites
- ▶ Sémantisation de nuages de points massifs
- ▶ Numérisation/vectorisation de cartes historiques
- ▶ Reconnaissances des espèces d'arbres
- ▶ et bien d'autres



<https://huggingface.co/datasets/IGNF/PureForest>

- ▶ Cartographie d'occupation des sols à partir d'images satellitaires
- ▶ Sémantisation de nuages de points massifs
- ▶ Numérisation/vectorisation de cartes historiques
- ▶ Reconnaissances des espèces d'arbres
- ▶ et bien d'autres
 - ▶ Reconstruction 3D par photogrammétrie
 - ▶ Classification de parcelles agricoles
 - ▶ Transfert de styles pour la cartographie
 - ▶ Localisation de personnes perdues en montagne
 - ▶ ...

Qu'est-ce que l'apprentissage automatique ?

Définition

Sous-discipline de l'intelligence artificielle utilisant des approches algorithmiques capables d'améliorer leurs performances sur une tâche à partir d'exemples, sans être explicitement programmées.

Programmation classique :  données +  algorithme =

résultat

Apprentissage automatique :

1. Apprentissage :  données + résultat =  algorithme

2. Inférence :  nouvelles données +  algorithme = résultat

En pratique, on appelle l'algorithme obtenu un **modèle** (en général, décisionnel).

Définition

Sous-discipline de l'intelligence artificielle utilisant des approches algorithmiques capables d'améliorer leurs performances sur une tâche à partir d'exemples, sans être explicitement programmées.

Programmation classique :  données +  algorithme =  résultat

Apprentissage automatique :

1. Apprentissage :  données +  résultat =  algorithme

2. Inférence :  nouvelles données +  algorithme =  résultat

En pratique, on appelle l'algorithme obtenu un **modèle** (en général, décisionnel).

Définition

Sous-discipline de l'intelligence artificielle utilisant des approches algorithmiques capables d'améliorer leurs performances sur une tâche à partir d'exemples, sans être explicitement programmées.

Programmation classique :  données +  algorithme =  résultat

Apprentissage automatique :

1. Apprentissage :  données +  résultat =  algorithme

2. Inférence :  nouvelles données +  algorithme =  résultat

En pratique, on appelle l'algorithme obtenu un **modèle** (en général, décisionnel).

- ▶ Observations décrites par les valeurs prises par un ensemble de variables
- Objectif : prédire, pour chaque donnée, la valeur d'une variable (**expliquée** ou « dépendante » ou « de sortie ») à partir des valeurs des autres variables (**explicatives** ou « d'entrée »)

Exemples

1. Une image représente un visage ou non ?
2. Les symptômes correspondent à la maladie A ou B ou C ou aucune ?
3. Quel sera le débit de la Loire à Tours dans 48h ?
4. Quelle est l'entité nommée dans « La Maison Blanche a démenti ces informations. » ?
5. Quelle est la région d'une image correspondant aux routes ?

- ▶ Observations décrites par les valeurs prises par un ensemble de variables
- Objectif : prédire, pour chaque donnée, la valeur d'une variable (**expliquée** ou « dépendante » ou « de sortie ») à partir des valeurs des autres variables (**explicatives** ou « d'entrée »)

Exemples

1. Une image représente un visage ou non ?
2. Les symptômes correspondent à la maladie A ou B ou C ou aucune ?
3. Quel sera le débit de la Loire à Tours dans 48h ?
4. Quelle est l'entité nommée dans « La Maison Blanche a démenti ces informations. » ?
5. Quelle est la région d'une image correspondant aux routes ?

- ▶ Observations décrites par les valeurs prises par un ensemble de variables
- Objectif : prédire, pour chaque donnée, la valeur d'une variable (**expliquée** ou « dépendante » ou « de sortie ») à partir des valeurs des autres variables (**explicatives** ou « d'entrée »)

Exemples

1. Une image représente un visage ou non ?
2. Les symptômes correspondent à la maladie A ou B ou C ou aucune ?
3. Quel sera le débit de la Loire à Tours dans 48h ?
4. Quelle est l'entité nommée dans « La Maison Blanche a démenti ces informations. » ?
5. Quelle est la région d'une image correspondant aux routes ?

- ▶ Observations décrites par les valeurs prises par un ensemble de variables
- Objectif : prédire, pour chaque donnée, la valeur d'une variable (**expliquée** ou « dépendante » ou « de sortie ») à partir des valeurs des autres variables (**explicatives** ou « d'entrée »)

Exemples

1. Une image représente un visage ou non ?
2. Les symptômes correspondent à la maladie A ou B ou C ou aucune ?
3. Quel sera le débit de la Loire à Tours dans 48h ?
4. Quelle est l'entité nommée dans « La Maison Blanche a démenti ces informations. » ?
5. Quelle est la région d'une image correspondant aux routes ?

- ▶ Observations décrites par les valeurs prises par un ensemble de variables
- Objectif : prédire, pour chaque donnée, la valeur d'une variable (**expliquée** ou « dépendante » ou « de sortie ») à partir des valeurs des autres variables (**explicatives** ou « d'entrée »)

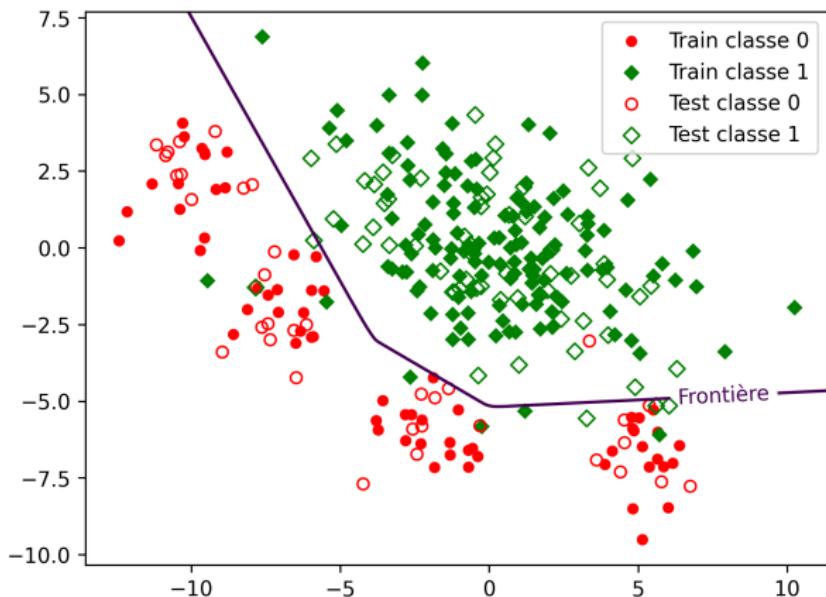
Exemples

1. Une image représente un visage ou non ?
2. Les symptômes correspondent à la maladie A ou B ou C ou aucune ?
3. Quel sera le débit de la Loire à Tours dans 48h ?
4. Quelle est l'entité nommée dans « La Maison Blanche a démenti ces informations. » ?
5. Quelle est la région d'une image correspondant aux routes ?

1. Classification : la variable expliquée est une variable nominale, chaque observation possède une modalité (appelée en général **classe**)
 - quel est le chiffre représenté par cette image ?
2. Régression : la variable expliquée est une variable quantitative (domaine $\subset \mathbb{R}$)
 - combien vaudra le CAC 40 dans une semaine ?
3. Prédiction structurée : la variable expliquée prend des valeurs dans un domaine de données **structurées** (les relations entre parties comptent)
 - quelle est la forme de la protéine compte-tenu des molécules qui la composent ?

Qu'est-ce qu'un modèle ?

- ▶ Modèle = règle de décision
- ▶ Exemple : frontière de discrimination pour une classification à 2 classes



Comment obtenir un modèle décisionnel

1. Construction analytique à partir des connaissances du phénomène

- ▶ Exemples :

- ▶ Temps de vol $t \leftarrow$ distance d et vitesse v , $t = v \cdot d$
- ▶ Concentration de produit de réaction \leftarrow concentration de réactif et température

- ▶ Néglige souvent l'impact de variables non contrôlables !

2. À partir de données : ensemble d'observations pour lesquelles les valeurs des variables explicatives et des variables expliquées sont connues

→ Apprentissage supervisé : il est nécessaire d'avoir des exemples du résultat recherché

Comment obtenir un modèle décisionnel

1. Construction analytique à partir des connaissances du phénomène

► Exemples :

- ▶ Temps de vol $t \leftarrow$ distance d et vitesse v , $t = v \cdot d$
- ▶ Concentration de produit de réaction \leftarrow concentration de réactif et température

▶ Néglige souvent l'impact de variables non contrôlables !

2. À partir de données : ensemble d'observations pour lesquelles les valeurs des variables explicatives et des variables expliquées sont connues

→ Apprentissage supervisé : il est nécessaire d'avoir des exemples du résultat recherché

Comment obtenir un modèle décisionnel

1. Construction analytique à partir des connaissances du phénomène

► Exemples :

- ▶ Temps de vol $t \leftarrow$ distance d et vitesse v , $t = v \cdot d$
- ▶ Concentration de produit de réaction \leftarrow concentration de réactif et température

▶ Néglige souvent l'impact de variables non contrôlables !

2. À partir de données : ensemble d'observations pour lesquelles les valeurs des variables explicatives et des variables expliquées sont connues

→ Apprentissage supervisé : il est nécessaire d'avoir des exemples du résultat recherché

1. Construction analytique à partir des connaissances du phénomène

- ▶ Exemples :

- ▶ Temps de vol $t \leftarrow$ distance d et vitesse v , $t = v \cdot d$
- ▶ Concentration de produit de réaction \leftarrow concentration de réactif et température

- ▶ Néglige souvent l'impact de variables non contrôlables !

2. À partir de données : ensemble d'observations pour lesquelles les valeurs des variables explicatives et des variables expliquées sont connues

→ Apprentissage supervisé : il est nécessaire d'avoir des exemples du résultat recherché

1. Construction analytique à partir des connaissances du phénomène

- ▶ Exemples :
 - ▶ Temps de vol $t \leftarrow$ distance d et vitesse v , $t = v \cdot d$
 - ▶ Concentration de produit de réaction \leftarrow concentration de réactif et température
- ▶ Néglige souvent l'impact de variables non contrôlables !

2. À partir de données : ensemble d'observations pour lesquelles les valeurs des variables explicatives et des variables expliquées sont connues

→ Apprentissage supervisé : il est nécessaire d'avoir des exemples du résultat recherché

1. Construction analytique à partir des connaissances du phénomène

- ▶ Exemples :

- ▶ Temps de vol $t \leftarrow$ distance d et vitesse v , $t = v \cdot d$
- ▶ Concentration de produit de réaction \leftarrow concentration de réactif et température

- ▶ Néglige souvent l'impact de variables non contrôlables !

2. À partir de données : ensemble d'observations pour lesquelles les valeurs des variables explicatives et des variables expliquées sont connues

→ Apprentissage supervisé : il est nécessaire d'avoir des exemples du résultat recherché

1. Construction analytique à partir des connaissances du phénomène

- ▶ Exemples :

- ▶ Temps de vol $t \leftarrow$ distance d et vitesse v , $t = v \cdot d$
- ▶ Concentration de produit de réaction \leftarrow concentration de réactif et température

- ▶ Néglige souvent l'impact de variables non contrôlables !

2. À partir de données : ensemble d'observations pour lesquelles les valeurs des variables explicatives et des variables expliquées sont connues

→ Apprentissage supervisé : il est nécessaire d'avoir des exemples du résultat recherché

Dans quels cas ai-je besoin de l'apprentissage automatique ?

Condition 1 : j'ai un problème décisionnel

Je cherche à prédire une variable y à partir d'un ensemble de variables x .

Condition 2 : il n'y a pas de solution analytique

Je ne peux pas construire manuellement une fonction $f: x \rightarrow y$.

Condition 3 : j'ai des données

Je dispose d'une collection d'exemples pour apprendre un modèle.

Construction d'un modèle à partir des données

- ▶ x_1, x_2, \dots, x_n des observations $x \in \mathcal{X}$
 - ▶ Typiquement, les observations $x \in \mathbb{R}^d$
- ▶ y_1, y_2, \dots, y_n des vérités terrain, $y \in \mathcal{Y}$
- ▶ une famille paramétrique de fonctions $f_\theta \in \mathcal{F}$, avec $f: \mathcal{X} \rightarrow \mathcal{Y}$
 - ▶ θ représente les paramètres du modèle
- ▶ une fonction de perte $\mathcal{L}(y, \hat{y})$

Problème d'optimisation

Formellement, on cherche à modéliser la fonction f qui permet de passer d'une observation x à la variable expliquée y :

$$\min_{f_\theta \in \mathcal{F}} \sum_{i=1}^n \mathcal{L}(y_i, f_\theta(x_i))$$

Apprendre un modèle = résoudre ce problème d'optimisation

Construction d'un modèle à partir des données

- ▶ x_1, x_2, \dots, x_n des observations $x \in \mathcal{X}$
 - ▶ Typiquement, les observations $x \in \mathbb{R}^d$
- ▶ y_1, y_2, \dots, y_n des vérités terrain, $y \in \mathcal{Y}$
- ▶ une famille paramétrique de fonctions $f_\theta \in \mathcal{F}$, avec $f: \mathcal{X} \rightarrow \mathcal{Y}$
 - ▶ θ représente les paramètres du modèle
- ▶ une fonction de perte $\mathcal{L}(y, \hat{y})$

Problème d'optimisation

Formellement, on cherche à modéliser la fonction f qui permet de passer d'une observation x à la variable expliquée y :

$$\min_{f_\theta \in \mathcal{F}} \sum_{i=1}^n \mathcal{L}(y_i, f_\theta(x_i))$$

Apprendre un modèle = résoudre ce problème d'optimisation

Apprentissage et généralisation

- ▶ Le modèle permet de prendre des décisions pour de futures données
- ▶ Erreur du modèle sur ces futures données = erreur de **généralisation**

Erreur d'entraînement/erreur de généralisation

En pratique, on veut minimiser l'erreur de généralisation mais on ne peut calculer que l'erreur d'entraînement !

- ▶ Erreur d'apprentissage facilement mesurable sur les données disponibles
- ▶ Données futures inconnues ⇒ erreur de généralisation ne peut pas être mesurée
- ▶ Hypothèse importante : la distribution des données d'apprentissage est représentative de celle des données futures !
- Minimiser l'erreur d'apprentissage = minimiser l'erreur de généralisation ?

Apprentissage et généralisation

- ▶ Le modèle permet de prendre des décisions pour de futures données
- ▶ Erreur du modèle sur ces futures données = erreur de **généralisation**

Erreur d'entraînement/erreur de généralisation

En pratique, on veut minimiser l'erreur de généralisation mais on ne peut calculer que l'erreur d'entraînement !

- ▶ Erreur d'apprentissage facilement mesurable sur les données disponibles
- ▶ Données futures inconnues ⇒ erreur de généralisation **ne peut pas être mesurée**
- ▶ **Hypothèse importante** : la distribution des données d'apprentissage est **représentative** de celle des données futures !
- Minimiser l'erreur d'apprentissage = minimiser l'erreur de généralisation ?

Apprentissage et généralisation

- ▶ Le modèle permet de prendre des décisions pour de futures données
- ▶ Erreur du modèle sur ces futures données = erreur de **généralisation**

Erreur d'entraînement/erreur de généralisation

En pratique, on veut minimiser l'erreur de généralisation mais on ne peut calculer que l'erreur d'entraînement !

- ▶ Erreur d'apprentissage facilement mesurable sur les données disponibles
- ▶ Données futures inconnues ⇒ erreur de généralisation **ne peut pas être mesurée**
- ▶ **Hypothèse importante** : la distribution des données d'apprentissage est **représentative** de celle des données futures !
 - Minimiser l'erreur d'apprentissage = minimiser l'erreur de généralisation ?

Apprentissage et généralisation

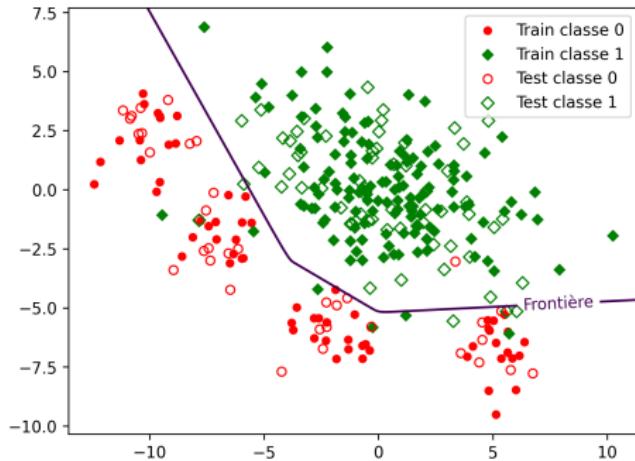
- ▶ Le modèle permet de prendre des décisions pour de futures données
- ▶ Erreur du modèle sur ces futures données = erreur de **généralisation**

Erreur d'entraînement/erreur de généralisation

En pratique, on veut minimiser l'erreur de généralisation mais on ne peut calculer que l'erreur d'entraînement !

- ▶ Erreur d'apprentissage facilement mesurable sur les données disponibles
- ▶ Données futures inconnues ⇒ erreur de généralisation **ne peut pas être mesurée**
- ▶ **Hypothèse importante** : la distribution des données d'apprentissage est **représentative** de celle des données futures !
- Minimiser l'erreur d'apprentissage = minimiser l'erreur de généralisation ?

Classification



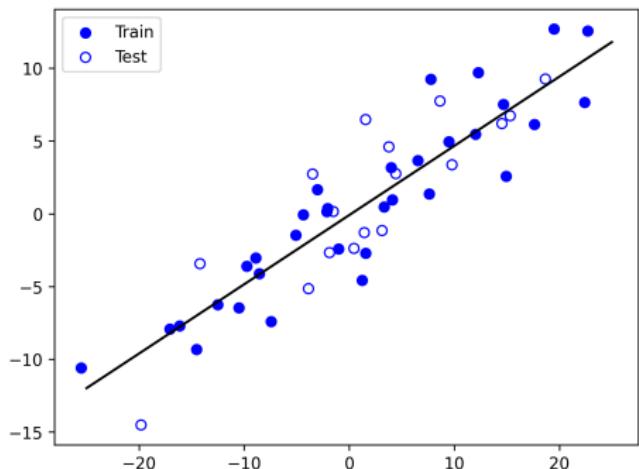
- ▶ Modèle : règle de classement, par ex. frontière de discrimination (trait violet)
- ▶ Exemple : observations (x_1, x_2) à deux dimensions (abscisse/ordonnée)
- ▶ $y \in \llbracket 1, k \rrbracket$ pour k classes

Fonction de coût :

$$\mathcal{L}_{01}(y_i, f_\theta(x_i)) = \begin{cases} 1 & \text{si } y_i \neq f_\theta(x_i) \\ 0 & \text{sinon} \end{cases}$$

⇒ métrique associée = taux de bonne classification (nombre d'erreurs)

Régression



- ▶ Modèle : règle de prédiction (trait noir)
 - ▶ Par ex. $y = Ax + b$ pour modèle linéaire
- ▶ Exemple : observations x à 1 dimension (abscisse), variable à prédire y en ordonnée
- ▶ $y \in \mathbb{R}$ ou \mathbb{R}^p

Fonction de coût :

$$\mathcal{L}_{\text{MSE}}(y_i, f_\theta(x_i)) = \|y_i - f_\theta(x_i)\|$$

⇒ erreur quadratique moyenne

Modélisation à partir de données : un cadre plus précis

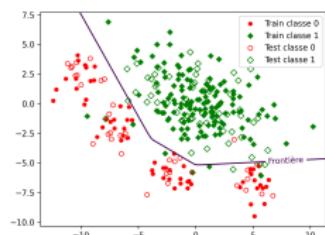
- ▶ Espace d'entrée (variables explicatives) : \mathcal{X} (par ex. \mathbb{R}^p)
- ▶ Espace de sortie (variable expliquée) : \mathcal{Y} (par ex. $\{-1; 1\}, \mathbb{R}$)
- ▶ Données à modéliser décrites par variables aléatoires $(X, Y) \in \mathcal{X} \times \mathcal{Y}$ suivant la distribution inconnue P
- ▶ Exemples

Classement :

$$\mathcal{X} \subset \mathbb{R}^2$$

$$\mathcal{Y} =$$

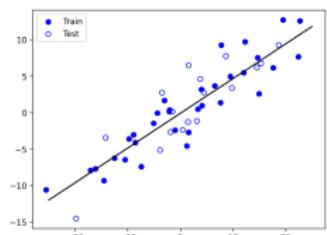
$$\{c_1, c_2\}$$



Régression :

$$\mathcal{X} \subset \mathbb{R}$$

$$\mathcal{Y} \subset \mathbb{R}$$



Jeu de données

Observations avec information de supervision : $\mathcal{D}_N = \{(\mathbf{x}_i, y_i)\}_{1 \leq i \leq N}$

- ▶ **Hypothèses** : ces observations sont des tirages identiquement distribués suivant P
- ▶ Sauf cas particuliers les tirages indépendants
 - ▶ Attention, ce n'est pas toujours vrai (séries temporelles, par exemple)

Objectif

Trouver, dans une famille \mathcal{F} , une fonction $f: \mathcal{X} \rightarrow \mathcal{Y}$ qui prédit y à partir de \mathbf{x}

- ▶ Avec l'erreur de généralisation $R(f) = \mathbb{E}_P[\mathcal{L}(X, Y, f)]$ la plus faible
- ▶ $\mathcal{L}()$ est la fonction de perte (ou d'erreur ou de coût)
- ▶ \mathbb{E}_P est l'espérance par rapport à la distribution inconnue P

Jeu de données

Observations avec information de supervision : $\mathcal{D}_N = \{(\mathbf{x}_i, y_i)\}_{1 \leq i \leq N}$

- ▶ **Hypothèses** : ces observations sont des tirages identiquement distribués suivant P
- ▶ Sauf cas particuliers les tirages indépendants
 - ▶ Attention, ce n'est pas toujours vrai (séries temporelles, par exemple)

Objectif

Trouver, dans une famille \mathcal{F} , une fonction $f: \mathcal{X} \rightarrow \mathcal{Y}$ qui prédit y à partir de \mathbf{x}

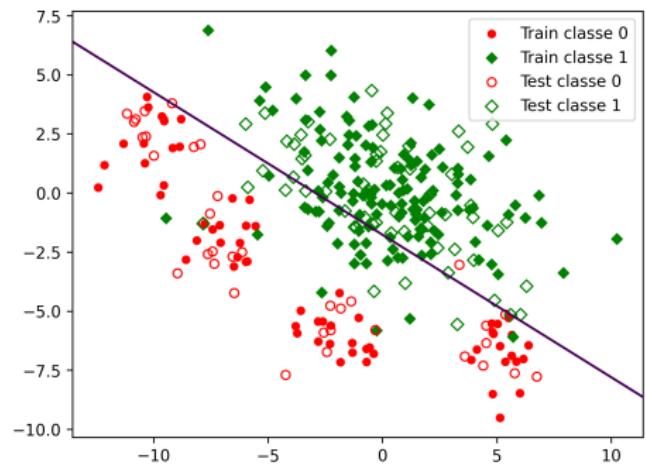
- ▶ Avec l'erreur de généralisation $R(f) = \mathbb{E}_P[\mathcal{L}(X, Y, f)]$ la plus faible
- ▶ $\mathcal{L}()$ est la fonction de perte (ou d'erreur ou de coût)
- ▶ \mathbb{E}_P est l'espérance par rapport à la distribution inconnue P

Modèles linéaires

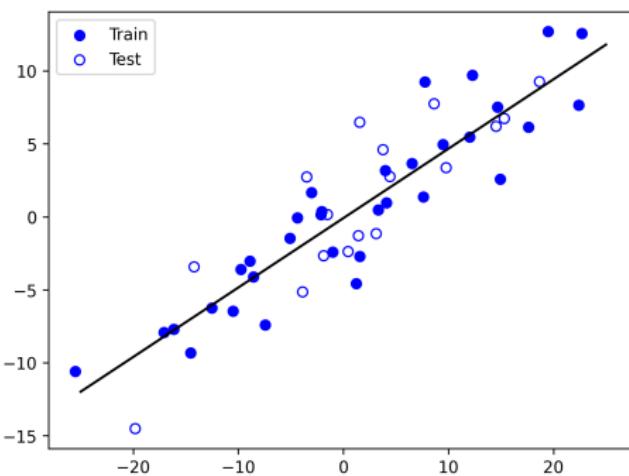
Prédiction = combinaison linéaire des variables

Classement : $f(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + w_0$

$$H(f(\mathbf{x})) \in \{-1, 1\}$$

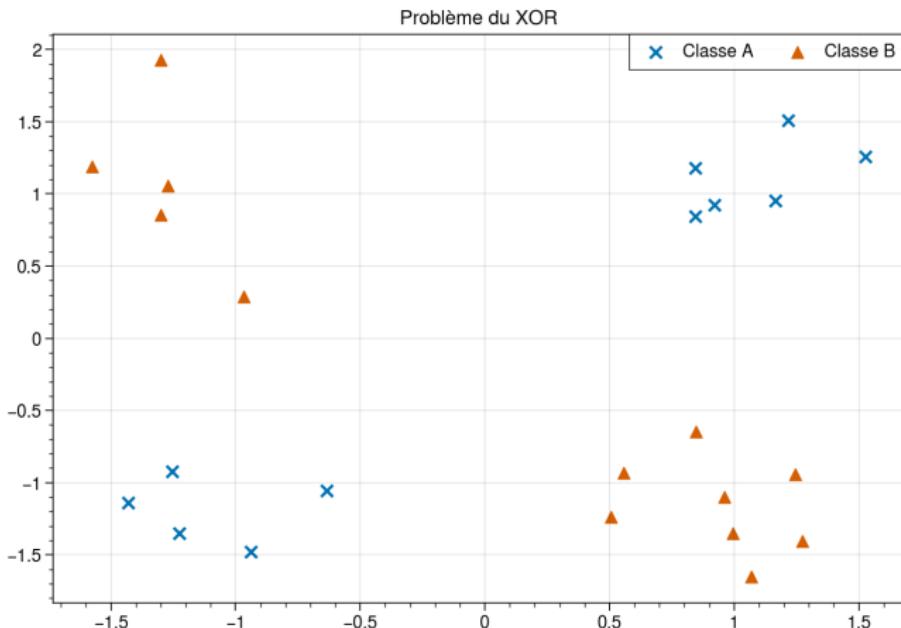


Régression : $f(x) = w_1 x + w_0$



Limites des modèles linéaires

Exemple simple de jeu de données non-linéairement séparable



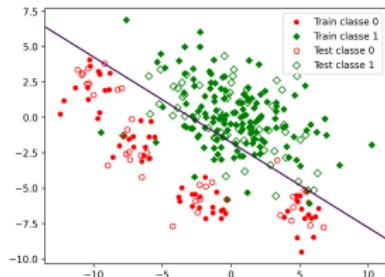
- ▶ Les modèles linéaires peuvent s'avérer insuffisants
 - ▶ Utile de commencer par un modèle linéaire, ne serait-ce que pour pouvoir comparer
- ▶ Modèles polynomiaux de degré borné : la capacité d'approximation (d'une frontière pour le classement, d'une dépendance pour la régression) augmente avec le degré
- ▶ Diverses familles de modèles non linéaires, par ex. réseaux de neurones d'architecture donnée, etc.

⇒ **Comment choisir ?**

⇒ **Pourquoi ne pas choisir systématiquement la famille de plus grande capacité ?**

Comment choisir la famille paramétrique ?

Linéaire

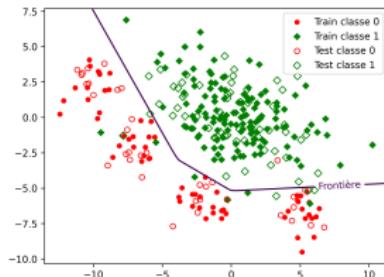


Err. app. 11,4%

Err. test 14,1%

Capacité \ominus

PMC $\alpha = 10$

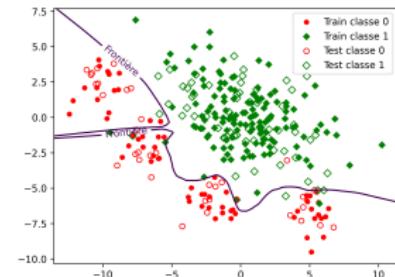


3,0%

6,1%

\oplus

PMC $\alpha = 0.5$



1,0%

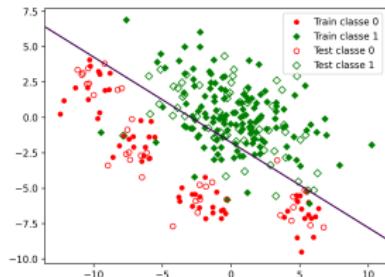
8,1%

$\oplus\oplus$

- Risque de **sur-apprentissage** (*overfitting*) : erreur d'apprentissage très faible mais erreur de test comparativement élevée
- ⇒ Ce n'est pas avec la capacité la plus grande qu'on obtient la meilleure généralisation
 - Quel lien entre capacité et généralisation ?

Comment choisir la famille paramétrique ?

Linéaire

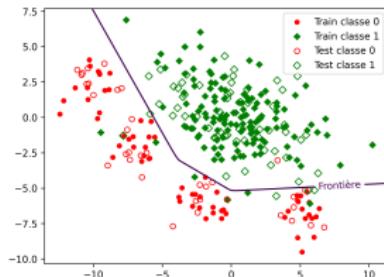


Err. app. 11,4%

Err. test 14,1%

Capacité \ominus

PMC $\alpha = 10$

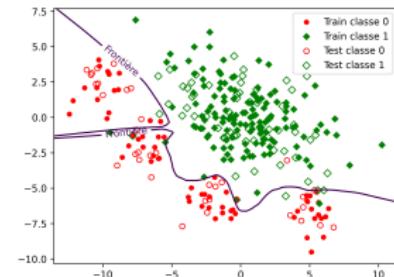


3,0%

6,1%

\oplus

PMC $\alpha = 0.5$



1,0%

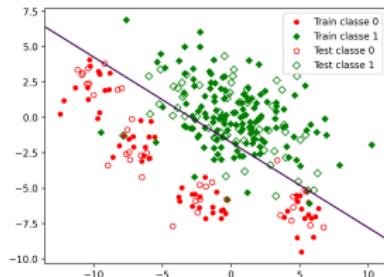
8,1%

$\oplus\oplus$

- Risque de **sur-apprentissage** (*overfitting*) : erreur d'apprentissage très faible mais erreur de test comparativement élevée
- ⇒ Ce n'est pas avec la capacité la plus grande qu'on obtient la meilleure généralisation
 - Quel lien entre capacité et généralisation ?

Comment choisir la famille paramétrique ?

Linéaire

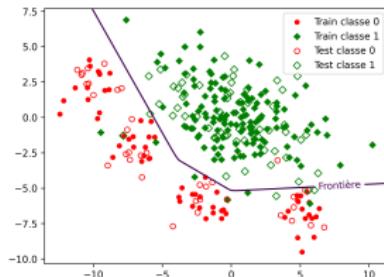


Err. app. 11,4%

Err. test 14,1%

Capacité \ominus

PMC $\alpha = 10$

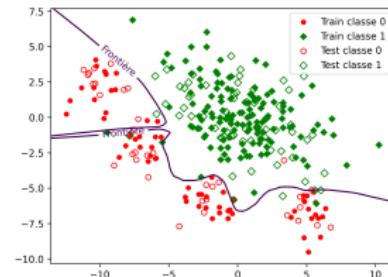


3,0%

6,1%

\oplus

PMC $\alpha = 0.5$



1,0%

8,1%

$\oplus\oplus$

- Risque de **sur-apprentissage** (*overfitting*) : erreur d'apprentissage très faible mais erreur de test comparativement élevée
- ⇒ Ce n'est pas avec la capacité la plus grande qu'on obtient la meilleure généralisation
 - Quel lien entre capacité et généralisation ?

Comment estimer le modèle ?

Objectif

Trouver, dans une famille \mathcal{F} choisie, une fonction (un modèle) $f: \mathcal{X} \rightarrow \mathcal{Y}$ qui prédit y à partir de x et présente l'erreur de généralisation

$$R(f) = \mathbb{E}_P[\mathcal{L}(X, Y, f)] \text{ la plus faible}$$

- ▶ $R(f)$ ne peut pas être évalué car on ne connaît pas la distribution des données P
- ▶ Mais on peut mesurer l'erreur d'apprentissage empirique
$$R_{\mathcal{D}_N}(f) = \frac{1}{N} \sum_{i=1}^N \mathcal{L}(x_i, y_i, f)$$
- ▶ Quels liens y-a-t-il entre ces deux erreurs ?

Objectif

Trouver, dans une famille \mathcal{F} choisie, une fonction (un modèle) $f: \mathcal{X} \rightarrow \mathcal{Y}$ qui prédit y à partir de x et présente l'erreur de généralisation

$$R(f) = \mathbb{E}_P[\mathcal{L}(X, Y, f)] \text{ la plus faible}$$

- ▶ $R(f)$ ne peut pas être évalué car on ne connaît pas la distribution des données P
- ▶ Mais on peut mesurer l'erreur d'apprentissage empirique
$$R_{\mathcal{D}_N}(f) = \frac{1}{N} \sum_{i=1}^N \mathcal{L}(\mathbf{x}_i, y_i, f)$$
- ▶ Quels liens y-a-t-il entre ces deux erreurs ?

Objectif

Trouver, dans une famille \mathcal{F} choisie, une fonction (un modèle) $f: \mathcal{X} \rightarrow \mathcal{Y}$ qui prédit y à partir de x et présente l'erreur de généralisation

$$R(f) = \mathbb{E}_P[\mathcal{L}(X, Y, f)] \text{ la plus faible}$$

- ▶ $R(f)$ ne peut pas être évalué car on ne connaît pas la distribution des données P
- ▶ Mais on peut mesurer l'erreur d'apprentissage empirique
$$R_{\mathcal{D}_N}(f) = \frac{1}{N} \sum_{i=1}^N \mathcal{L}(\mathbf{x}_i, y_i, f)$$
- ▶ Quels liens y-a-t-il entre ces deux erreurs ?

Analyse des composantes de l'erreur de généralisation

► Considérons

- ▶ $f_{\mathcal{D}_N}^*$ la fonction de \mathcal{F} qui minimise l'erreur empirique $R_{\mathcal{D}_N}$
- ▶ f^* la fonction de \mathcal{F} qui minimise l'erreur de généralisation R , alors

$$R(f_{\mathcal{D}_N}^*) = \underbrace{R^*}_{\text{Bayes}} + \underbrace{[R(f^*) - R^*]}_{\text{approx.}} + \underbrace{[R(f_{\mathcal{D}_N}^*) - R(f^*)]}_{\text{estim.}}$$

1. R^* est l'erreur résiduelle (ou erreur de Bayes), borne inférieure
 - ▶ Strictement positive en présence de bruit : suivant le bruit, à un même x peuvent correspondre plusieurs valeurs de y
2. $[R(f^*) - R^*]$ est l'erreur d'**approximation** (≥ 0) car \mathcal{F} ne contient pas nécessairement la « vraie » fonction de décision
 - ▶ Nulle seulement si R^* peut être atteint par une fonction de \mathcal{F}
3. $[R(f_{\mathcal{D}_N}^*) - R(f^*)]$ est l'erreur d'**estimation** (≥ 0)
 - ▶ La fonction de \mathcal{F} qui minimise l'erreur d'apprentissage n'est pas nécessairement celle qui minimise l'erreur de généralisation

Capacité, erreur d'approximation et erreur d'estimation

Capacité modèle linéaire < capacité PMC $\alpha = 10$ < capacité PMC $\alpha = 0.2$

1. Famille linéaire

- ▶ Erreur d'apprentissage élevée donc capacité insuffisante pour ce problème
⇒ Erreur d'approximation élevée (fort biais)

2. Famille définie par PMC 1 couche cachée de 300 neurones, avec coefficient « d'oubli » $\alpha = 0.2$

- ▶ Erreur d'approximation probablement faible car erreur d'apprentissage faible ⇒ capacité suffisante
- ▶ Erreur de test bien plus élevée
⇒ Erreur d'estimation élevée

3. Famille définie par PMC 1 couche cachée de 300 neurones, avec coefficient « d'oubli » $\alpha = 10$

- ▶ Somme assez faible entre erreur d'approximation et erreur d'estimation, meilleure généralisation que les deux autres familles
- ▶ Erreur de test assez faible et proche de l'erreur d'apprentissage

Capacité, erreur d'approximation et erreur d'estimation

Capacité modèle linéaire < capacité PMC $\alpha = 10$ < capacité PMC $\alpha = 0.2$

1. Famille linéaire

- ▶ Erreur d'apprentissage élevée donc capacité insuffisante pour ce problème
 - ⇒ Erreur d'approximation élevée (fort biais)

2. Famille définie par PMC 1 couche cachée de 300 neurones, avec coefficient « d'oubli » $\alpha = 0.2$

- ▶ Erreur d'approximation probablement faible car erreur d'apprentissage faible ⇒ capacité suffisante
- ▶ Erreur de test bien plus élevée
 - ⇒ Erreur d'estimation élevée

3. Famille définie par PMC 1 couche cachée de 300 neurones, avec coefficient « d'oubli » $\alpha = 10$

- ▶ Somme assez faible entre erreur d'approximation et erreur d'estimation, meilleure généralisation que les deux autres familles
- ▶ Erreur de test assez faible et proche de l'erreur d'apprentissage

Capacité, erreur d'approximation et erreur d'estimation

Capacité modèle linéaire < capacité PMC $\alpha = 10$ < capacité PMC $\alpha = 0.2$

1. Famille linéaire

- ▶ Erreur d'apprentissage élevée donc capacité insuffisante pour ce problème
- ⇒ Erreur d'approximation élevée (fort **biais**)

2. Famille définie par PMC 1 couche cachée de 300 neurones, avec coefficient « d'oubli » $\alpha = 0.2$

- ▶ Erreur d'approximation probablement faible car erreur d'apprentissage faible ⇒ capacité suffisante
- ▶ Erreur de test bien plus élevée
- ▶ Erreur d'estimation élevée

3. Famille définie par PMC 1 couche cachée de 300 neurones, avec coefficient « d'oubli » $\alpha = 10$

- ▶ Somme assez faible entre erreur d'approximation et erreur d'estimation, meilleure généralisation que les deux autres familles
- ▶ Erreur de test assez faible et proche de l'erreur d'apprentissage

Capacité, erreur d'approximation et erreur d'estimation

Capacité modèle linéaire < capacité PMC $\alpha = 10$ < capacité PMC $\alpha = 0.2$

1. Famille linéaire

- ▶ Erreur d'apprentissage élevée donc capacité insuffisante pour ce problème
- ⇒ Erreur d'approximation élevée (fort **biais**)

2. Famille définie par PMC 1 couche cachée de 300 neurones, avec coefficient « d'oubli » $\alpha = 0.2$

- ▶ Erreur d'approximation probablement faible car erreur d'apprentissage faible ⇒ capacité suffisante
- ▶ Erreur de test bien plus élevée
- ⇒ Erreur d'estimation élevée

3. Famille définie par PMC 1 couche cachée de 300 neurones, avec coefficient « d'oubli » $\alpha = 10$

- ▶ Somme assez faible entre erreur d'approximation et erreur d'estimation, meilleure généralisation que les deux autres familles
- ▶ Erreur de test assez faible et proche de l'erreur d'apprentissage

Capacité, erreur d'approximation et erreur d'estimation

Capacité modèle linéaire < capacité PMC $\alpha = 10$ < capacité PMC $\alpha = 0.2$

1. Famille linéaire

- ▶ Erreur d'apprentissage élevée donc capacité insuffisante pour ce problème
 - ⇒ Erreur d'approximation élevée (fort **biais**)

2. Famille définie par PMC 1 couche cachée de 300 neurones, avec coefficient « d'oubli » $\alpha = 0.2$

- ▶ Erreur d'approximation probablement faible car erreur d'apprentissage faible ⇒ capacité suffisante
- ▶ Erreur de test bien plus élevée
 - ⇒ Erreur d'estimation élevée

3. Famille définie par PMC 1 couche cachée de 300 neurones, avec coefficient « d'oubli » $\alpha = 10$

- ▶ Somme assez faible entre erreur d'approximation et erreur d'estimation, meilleure généralisation que les deux autres familles
- ▶ Erreur de test assez faible et proche de l'erreur d'apprentissage

Capacité, erreur d'approximation et erreur d'estimation

Capacité modèle linéaire < capacité PMC $\alpha = 10$ < capacité PMC $\alpha = 0.2$

1. Famille linéaire

- ▶ Erreur d'apprentissage élevée donc capacité insuffisante pour ce problème
- ⇒ Erreur d'approximation élevée (fort **biais**)

2. Famille définie par PMC 1 couche cachée de 300 neurones, avec coefficient « d'oubli » $\alpha = 0.2$

- ▶ Erreur d'approximation probablement faible car erreur d'apprentissage faible ⇒ capacité suffisante
- ▶ Erreur de test bien plus élevée
- ⇒ Erreur d'estimation élevée

3. Famille définie par PMC 1 couche cachée de 300 neurones, avec coefficient « d'oubli » $\alpha = 10$

- ▶ Somme assez faible entre erreur d'approximation et erreur d'estimation, meilleure généralisation que les deux autres familles
- ▶ Erreur de test assez faible et proche de l'erreur d'apprentissage

Capacité, erreur d'approximation et erreur d'estimation

Capacité modèle linéaire < capacité PMC $\alpha = 10$ < capacité PMC $\alpha = 0.2$

1. Famille linéaire

- ▶ Erreur d'apprentissage élevée donc capacité insuffisante pour ce problème
- ⇒ Erreur d'approximation élevée (fort **biais**)

2. Famille définie par PMC 1 couche cachée de 300 neurones, avec coefficient « d'oubli » $\alpha = 0.2$

- ▶ Erreur d'approximation probablement faible car erreur d'apprentissage faible ⇒ capacité suffisante
- ▶ Erreur de test bien plus élevée
- ⇒ Erreur d'estimation élevée

3. Famille définie par PMC 1 couche cachée de 300 neurones, avec coefficient « d'oubli » $\alpha = 10$

- ▶ Somme assez faible entre erreur d'approximation et erreur d'estimation, meilleure généralisation que les deux autres familles
- ▶ Erreur de test assez faible et proche de l'erreur d'apprentissage

Capacité, erreur d'approximation et erreur d'estimation

Capacité modèle linéaire < capacité PMC $\alpha = 10$ < capacité PMC $\alpha = 0.2$

1. Famille linéaire

- ▶ Erreur d'apprentissage élevée donc capacité insuffisante pour ce problème
- ⇒ Erreur d'approximation élevée (fort **biais**)

2. Famille définie par PMC 1 couche cachée de 300 neurones, avec coefficient « d'oubli » $\alpha = 0.2$

- ▶ Erreur d'approximation probablement faible car erreur d'apprentissage faible ⇒ capacité suffisante
- ▶ Erreur de test bien plus élevée
- ⇒ Erreur d'estimation élevée

3. Famille définie par PMC 1 couche cachée de 300 neurones, avec coefficient « d'oubli » $\alpha = 10$

- ▶ Somme assez faible entre erreur d'approximation et erreur d'estimation, meilleure généralisation que les deux autres familles
- ▶ Erreur de test assez faible et proche de l'erreur d'apprentissage

Capacité, erreur d'approximation et erreur d'estimation

Capacité modèle linéaire < capacité PMC $\alpha = 10$ < capacité PMC $\alpha = 0.2$

1. Famille linéaire

- ▶ Erreur d'apprentissage élevée donc capacité insuffisante pour ce problème
- ⇒ Erreur d'approximation élevée (fort **biais**)

2. Famille définie par PMC 1 couche cachée de 300 neurones, avec coefficient « d'oubli » $\alpha = 0.2$

- ▶ Erreur d'approximation probablement faible car erreur d'apprentissage faible ⇒ capacité suffisante
- ▶ Erreur de test bien plus élevée
- ⇒ Erreur d'estimation élevée

3. Famille définie par PMC 1 couche cachée de 300 neurones, avec coefficient « d'oubli » $\alpha = 10$

- ▶ Somme assez faible entre erreur d'approximation et erreur d'estimation, meilleure généralisation que les deux autres familles
- ▶ Erreur de test assez faible et proche de l'erreur d'apprentissage

Capacité modèle linéaire < capacité PMC $\alpha = 10$ < capacité PMC $\alpha = 0.2$

1. Famille linéaire

- ▶ Erreur d'apprentissage élevée donc capacité insuffisante pour ce problème
- ⇒ Erreur d'approximation élevée (fort **biais**)

2. Famille définie par PMC 1 couche cachée de 300 neurones, avec coefficient « d'oubli » $\alpha = 0.2$

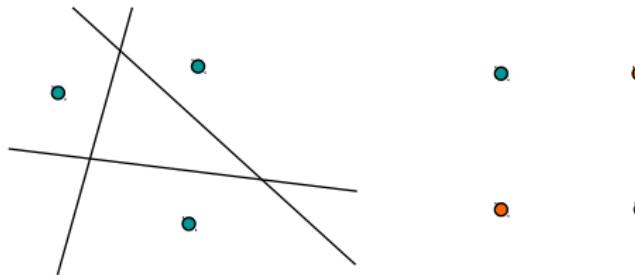
- ▶ Erreur d'approximation probablement faible car erreur d'apprentissage faible ⇒ capacité suffisante
- ▶ Erreur de test bien plus élevée
- ⇒ Erreur d'estimation élevée

3. Famille définie par PMC 1 couche cachée de 300 neurones, avec coefficient « d'oubli » $\alpha = 10$

- ▶ Somme assez faible entre erreur d'approximation et erreur d'estimation, meilleure généralisation que les deux autres familles
- ▶ Erreur de test assez faible et proche de l'erreur d'apprentissage

Comment mesurer la capacité ?

- ▶ Considérons un ensemble de N vecteurs $\{\mathbf{x}_i\}_{1 \leq i \leq N} \in \mathbb{R}^p \rightarrow$
 - ▶ il y a 2^N façons différentes de le séparer en 2 parties
- ▶ **Définition** : la famille \mathcal{F} de fonctions $f: \mathbb{R}^p \rightarrow \{-1, 1\}$ pulvérise $\{\mathbf{x}_i\}_{1 \leq i \leq N}$ si toutes les 2^N séparations peuvent être construites avec des fonctions de \mathcal{F}
- ▶ **Définition** (Vapnik-Chervonenkis) : \mathcal{F} est de VC-dimension h si elle pulvérise au moins h vecteurs et aucun ensemble de $h + 1$ vecteurs
- ▶ Exemple : la VC-dimension des hyperplans de \mathbb{R}^p est $h = p + 1$
 - ▶ Dans \mathbb{R}^2 , les droites pulvérissent le triplet à gauche mais aucun quadruplet



Théorème

Soit $R_{\mathcal{D}_N}(f)$ le risque empirique défini par la fonction de perte $L_{01}(\mathbf{x}, y, f) = \mathbf{1}_{f(\mathbf{x}) \neq y}$; si la VC-dimension de \mathcal{F} est $h < \infty$ alors pour toute $f \in \mathcal{F}$, avec une probabilité au moins égale à $1 - \delta$ ($0 < \delta < 1$), on a

$$R(f) \leq R_{\mathcal{D}_N}(f) + \underbrace{\sqrt{\frac{h \left(\log \frac{2N}{h} + 1 \right) - \log \frac{\delta}{4}}{N}}}_{B(N, \mathcal{F})} \quad \text{pour } N > h$$

- ▶ $B(N, \mathcal{F})$ diminue quand $N \uparrow$, quand $h \downarrow$ et quand $\delta \uparrow$
- ▶ $B(N, \mathcal{F})$ ne fait pas intervenir le nombre de variables
- ▶ $B(N, \mathcal{F})$ ne fait pas intervenir la loi conjointe P
- résultat dans le pire des cas, intéressant d'un point de vue théorique

Lien entre capacité et généralisation (2)

Conséquences de l'existence d'une borne :

$$\underbrace{R(f)}_{\text{err. test}} \leq \underbrace{R_{D_N}(f)}_{\text{err. app.}} + \underbrace{B(N, \mathcal{F})}_{\text{borne}}$$

et compte-tenu l'expression de la borne $B(N, \mathcal{F})$:

► Famille \mathcal{F} de capacité trop faible

- ⇒ $B(N, \mathcal{F})$ faible mais $R_{D_N}(f)$ (erreur d'apprentissage) élevé(e)
- ⇒ absence de garantie intéressante pour $R(f)$

► Famille \mathcal{F} de capacité trop élevée

- ⇒ $R_{D_N}(f)$ probablement faible mais $B(N, \mathcal{F})$ élevée
- ⇒ absence de garantie intéressante pour $R(f)$

► Famille \mathcal{F} de capacité « adéquate »

- ⇒ $R_{D_N}(f)$ probablement faible et $B(N, \mathcal{F})$ plutôt faible
- ⇒ garantie intéressante pour $R(f)$!

Lien entre capacité et généralisation (2)

Conséquences de l'existence d'une borne :

$$\underbrace{R(f)}_{\text{err. test}} \leq \underbrace{R_{\mathcal{D}_N}(f)}_{\text{err. app.}} + \underbrace{B(N, \mathcal{F})}_{\text{borne}}$$

et compte-tenu l'expression de la borne $B(N, \mathcal{F})$:

- ▶ Famille \mathcal{F} de capacité trop faible
 - ⇒ $B(N, \mathcal{F})$ faible mais $R_{\mathcal{D}_N}(f)$ (erreur d'apprentissage) élevé(e)
 - ⇒ absence de garantie intéressante pour $R(f)$
- ▶ Famille \mathcal{F} de capacité trop élevée
 - ⇒ $R_{\mathcal{D}_N}(f)$ probablement faible mais $B(N, \mathcal{F})$ élevée
 - ⇒ absence de garantie intéressante pour $R(f)$
- ▶ Famille \mathcal{F} de capacité « adéquate »
 - ⇒ $R_{\mathcal{D}_N}(f)$ probablement faible et $B(N, \mathcal{F})$ plutôt faible
 - ⇒ garantie intéressante pour $R(f)$!

Lien entre capacité et généralisation (2)

Conséquences de l'existence d'une borne :

$$\underbrace{R(f)}_{\text{err. test}} \leq \underbrace{R_{\mathcal{D}_N}(f)}_{\text{err. app.}} + \underbrace{B(N, \mathcal{F})}_{\text{borne}}$$

et compte-tenu l'expression de la borne $B(N, \mathcal{F})$:

- ▶ Famille \mathcal{F} de capacité trop faible
 - ⇒ $B(N, \mathcal{F})$ faible mais $R_{\mathcal{D}_N}(f)$ (erreur d'apprentissage) élevé(e)
 - ⇒ absence de garantie intéressante pour $R(f)$
- ▶ Famille \mathcal{F} de capacité trop élevée
 - ⇒ $R_{\mathcal{D}_N}(f)$ probablement faible mais $B(N, \mathcal{F})$ élevée
 - ⇒ absence de garantie intéressante pour $R(f)$
- ▶ Famille \mathcal{F} de capacité « adéquate »
 - ⇒ $R_{\mathcal{D}_N}(f)$ probablement faible et $B(N, \mathcal{F})$ plutôt faible
 - ⇒ garantie intéressante pour $R(f)$!

Lien entre capacité et généralisation (2)

Conséquences de l'existence d'une borne :

$$\underbrace{R(f)}_{\text{err. test}} \leq \underbrace{R_{\mathcal{D}_N}(f)}_{\text{err. app.}} + \underbrace{B(N, \mathcal{F})}_{\text{borne}}$$

et compte-tenu l'expression de la borne $B(N, \mathcal{F})$:

- ▶ Famille \mathcal{F} de capacité trop faible
 - ⇒ $B(N, \mathcal{F})$ faible mais $R_{\mathcal{D}_N}(f)$ (erreur d'apprentissage) élevé(e)
 - ⇒ absence de garantie intéressante pour $R(f)$
- ▶ Famille \mathcal{F} de capacité trop élevée
 - ⇒ $R_{\mathcal{D}_N}(f)$ probablement faible mais $B(N, \mathcal{F})$ élevée
 - ⇒ absence de garantie intéressante pour $R(f)$
- ▶ Famille \mathcal{F} de capacité « adéquate »
 - ⇒ $R_{\mathcal{D}_N}(f)$ probablement faible et $B(N, \mathcal{F})$ plutôt faible
 - ⇒ garantie intéressante pour $R(f)$!

Lien entre capacité et généralisation (2)

Conséquences de l'existence d'une borne :

$$\underbrace{R(f)}_{\text{err. test}} \leq \underbrace{R_{\mathcal{D}_N}(f)}_{\text{err. app.}} + \underbrace{B(N, \mathcal{F})}_{\text{borne}}$$

et compte-tenu l'expression de la borne $B(N, \mathcal{F})$:

- ▶ Famille \mathcal{F} de capacité trop faible
 - ⇒ $B(N, \mathcal{F})$ faible mais $R_{\mathcal{D}_N}(f)$ (erreur d'apprentissage) élevé(e)
 - ⇒ absence de garantie intéressante pour $R(f)$
- ▶ Famille \mathcal{F} de capacité trop élevée
 - ⇒ $R_{\mathcal{D}_N}(f)$ probablement faible mais $B(N, \mathcal{F})$ élevée
 - ⇒ absence de garantie intéressante pour $R(f)$
- ▶ Famille \mathcal{F} de capacité « adéquate »
 - ⇒ $R_{\mathcal{D}_N}(f)$ probablement faible et $B(N, \mathcal{F})$ plutôt faible
 - ⇒ garantie intéressante pour $R(f)$!

Lien entre capacité et généralisation (2)

Conséquences de l'existence d'une borne :

$$\underbrace{R(f)}_{\text{err. test}} \leq \underbrace{R_{\mathcal{D}_N}(f)}_{\text{err. app.}} + \underbrace{B(N, \mathcal{F})}_{\text{borne}}$$

et compte-tenu l'expression de la borne $B(N, \mathcal{F})$:

- ▶ Famille \mathcal{F} de capacité trop faible
 - ⇒ $B(N, \mathcal{F})$ faible mais $R_{\mathcal{D}_N}(f)$ (erreur d'apprentissage) élevé(e)
 - ⇒ absence de garantie intéressante pour $R(f)$
- ▶ Famille \mathcal{F} de capacité trop élevée
 - ⇒ $R_{\mathcal{D}_N}(f)$ probablement faible mais $B(N, \mathcal{F})$ élevée
 - ⇒ absence de garantie intéressante pour $R(f)$
- ▶ Famille \mathcal{F} de capacité « adéquate »
 - ⇒ $R_{\mathcal{D}_N}(f)$ probablement faible et $B(N, \mathcal{F})$ plutôt faible
 - ⇒ garantie intéressante pour $R(f)$!

Lien entre capacité et généralisation (2)

Conséquences de l'existence d'une borne :

$$\underbrace{R(f)}_{\text{err. test}} \leq \underbrace{R_{\mathcal{D}_N}(f)}_{\text{err. app.}} + \underbrace{B(N, \mathcal{F})}_{\text{borne}}$$

et compte-tenu l'expression de la borne $B(N, \mathcal{F})$:

- ▶ Famille \mathcal{F} de capacité trop faible
 - ⇒ $B(N, \mathcal{F})$ faible mais $R_{\mathcal{D}_N}(f)$ (erreur d'apprentissage) élevé(e)
 - ⇒ absence de garantie intéressante pour $R(f)$
- ▶ Famille \mathcal{F} de capacité trop élevée
 - ⇒ $R_{\mathcal{D}_N}(f)$ probablement faible mais $B(N, \mathcal{F})$ élevée
 - ⇒ absence de garantie intéressante pour $R(f)$
- ▶ Famille \mathcal{F} de capacité « adéquate »
 - ⇒ $R_{\mathcal{D}_N}(f)$ probablement faible et $B(N, \mathcal{F})$ plutôt faible
 - ⇒ garantie intéressante pour $R(f)$!

Lien entre capacité et généralisation (2)

Conséquences de l'existence d'une borne :

$$\underbrace{R(f)}_{\text{err. test}} \leq \underbrace{R_{\mathcal{D}_N}(f)}_{\text{err. app.}} + \underbrace{B(N, \mathcal{F})}_{\text{borne}}$$

et compte-tenu l'expression de la borne $B(N, \mathcal{F})$:

- ▶ Famille \mathcal{F} de capacité trop faible
 - ⇒ $B(N, \mathcal{F})$ faible mais $R_{\mathcal{D}_N}(f)$ (erreur d'apprentissage) élevé(e)
 - ⇒ absence de garantie intéressante pour $R(f)$
- ▶ Famille \mathcal{F} de capacité trop élevée
 - ⇒ $R_{\mathcal{D}_N}(f)$ probablement faible mais $B(N, \mathcal{F})$ élevée
 - ⇒ absence de garantie intéressante pour $R(f)$
- ▶ Famille \mathcal{F} de capacité « adéquate »
 - ⇒ $R_{\mathcal{D}_N}(f)$ probablement faible et $B(N, \mathcal{F})$ plutôt faible
 - ⇒ garantie intéressante pour $R(f)$!

Lien entre capacité et généralisation (2)

Conséquences de l'existence d'une borne :

$$\underbrace{R(f)}_{\text{err. test}} \leq \underbrace{R_{\mathcal{D}_N}(f)}_{\text{err. app.}} + \underbrace{B(N, \mathcal{F})}_{\text{borne}}$$

et compte-tenu l'expression de la borne $B(N, \mathcal{F})$:

- ▶ Famille \mathcal{F} de capacité trop faible
 - ⇒ $B(N, \mathcal{F})$ faible mais $R_{\mathcal{D}_N}(f)$ (erreur d'apprentissage) élevé(e)
 - ⇒ absence de garantie intéressante pour $R(f)$
- ▶ Famille \mathcal{F} de capacité trop élevée
 - ⇒ $R_{\mathcal{D}_N}(f)$ probablement faible mais $B(N, \mathcal{F})$ élevée
 - ⇒ absence de garantie intéressante pour $R(f)$
- ▶ Famille \mathcal{F} de capacité « adéquate »
 - ⇒ $R_{\mathcal{D}_N}(f)$ probablement faible et $B(N, \mathcal{F})$ plutôt faible
 - ⇒ garantie intéressante pour $R(f)$!

Comment minimiser l'erreur d'apprentissage ?

- ▶ Dans une famille paramétrique \mathcal{F} , un modèle est défini par les valeurs d'un ensemble de paramètres, par ex.
 - ▶ Modèle linéaire pour la régression $y = ax + b$: a et b
 - ▶ Arbre de décision : seuils des tests et choix des variables
 - ▶ Réseaux de neurones : poids des connexions
- Optimisation pour trouver les valeurs qui minimisent l'erreur
 - ▶ Solution analytique directe : cas assez rare, par ex. certains modèles linéaires
 - ▶ Algorithmes itératifs, par ex.
 - ▶ Optimisation quadratique sous contraintes d'inégalité : SVM
 - ▶ Optimisation non linéaire plus générale : réseaux de neurones

Exemple : régression linéaire

Problème de régression avec $\mathcal{X} = \mathbb{R}^p$, $\mathcal{Y} = \mathbb{R}$, $\mathcal{D}_N = \{(\mathbf{x}_i, y_i)\}_{1 \leq i \leq N}$

Régression linéaire

- ▶ Famille paramétrique : $\hat{y} = w_0 + \sum_{j=1}^p w_j x_{ji}$, où \hat{y} est la prédiction

$$\begin{pmatrix} x_{11} & \dots & x_{p1} & 1 \\ \vdots & \vdots & \vdots & \vdots \\ x_{1n} & \dots & x_{pn} & 1 \end{pmatrix} \begin{pmatrix} w_p \\ \vdots \\ w_0 \end{pmatrix}$$

- ▶ Sous forme matricielle : $\hat{\mathbf{y}} = \mathbf{X}\mathbf{w} =$
- ▶ On cherche le modèle (défini par le vecteur de paramètres \mathbf{w}^*) qui minimise

- ▶ MRE : l'erreur quadratique totale $\sum_{i=1}^N (\hat{y}_i - y_i)^2$ sur \mathcal{D}_N
- Solution $\mathbf{w}^* = \mathbf{X}^+ \mathbf{y}$, où \mathbf{X}^+ est la pseudo-inverse Moore-Penrose de \mathbf{X}
- ▶ Si $\mathbf{X}^T \mathbf{X}$ est inversible, alors $\mathbf{X}^+ = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T$

- ▶ Construire un modèle décisionnel à partir de données : supervision nécessaire
- ▶ Objectif : obtenir le modèle qui présente la meilleure généralisation
- ▶ Estimer la généralisation : non à partir de l'erreur d'apprentissage
- ▶ Chercher le bon compromis entre minimisation de la capacité de la famille de modèles et minimisation de l'erreur d'apprentissage

Évaluation de modèles

Comment estimer l'erreur de généralisation

Données de test

Par l'erreur sur des données de **test**, non utilisées pour l'apprentissage (par ex., 30% des données sont mises à l'écart pour le test)

- ▶ Apprentissage (estimation) du modèle sur les données d'apprentissage
- ▶ Estimation de l'erreur de généralisation sur les données de test
 - ▶ Réduit le nombre de données utilisées pour l'apprentissage
 - ▶ Variance élevée : autre partitionnement \implies estimation différente

Validation croisée

Cross-validation : plusieurs partitions apprentissage | test

- \Rightarrow estimateur de variance plus faible,
- ... tout en utilisant mieux les données disponibles !
- ▶ mais plusieurs modèles à entraîner

Comment estimer l'erreur de généralisation

Données de test

Par l'erreur sur des données de **test**, non utilisées pour l'apprentissage (par ex., 30% des données sont mises à l'écart pour le test)

- ▶ Apprentissage (estimation) du modèle sur les données d'apprentissage
- ▶ Estimation de l'erreur de généralisation sur les données de test
 - ▶ Réduit le nombre de données utilisées pour l'apprentissage
 - ▶ Variance élevée : autre partitionnement \implies estimation différente

Validation croisée

Cross-validation : plusieurs partitions apprentissage | test

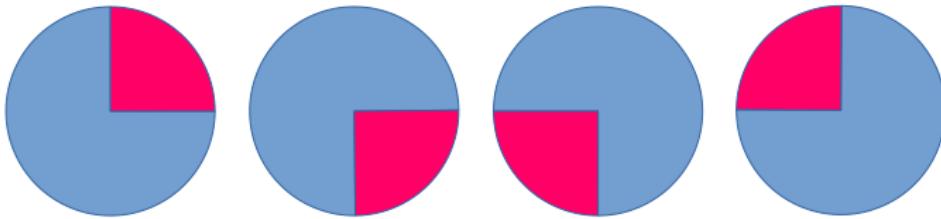
- ⇒ estimateur de variance plus faible,
- ... tout en utilisant mieux les données disponibles !
- ▶ mais plusieurs modèles à entraîner

1. Méthodes exhaustives :

- ▶ *Leave p out* (LPO) : $N - p$ données pour l'apprentissage et p pour la validation $\Rightarrow C_N^p$ découpages donc C_N^p modèles \Rightarrow coût excessif
- ▶ *Leave one out* (LOO) : $N - 1$ données pour l'apprentissage et 1 pour la validation $\Rightarrow C_N^1 = N$ découpages (donc N modèles) \Rightarrow coût élevé

2. Méthodes non exhaustives :

- ▶ *k-fold* : partitionnement fixé des N données en k parties, apprentissage sur $k - 1$ parties et validation sur la k -ème $\Rightarrow k$ modèles seulement



- ▶ Échantillonnage répété (*shuffle and split*) : échantillon aléatoire de p données pour le test (les autres $N - p$ pour l'apprentissage), on répète k fois $\Rightarrow k$ modèles

Validation croisée : quelle méthode préférer ?

- ▶ LPO très rarement employée car excessivement coûteuse
- ▶ LOO vs *k-fold* : *k-fold* préférée en général
 - ▶ LOO plus coûteuse car $N \gg k$
 - ▶ Variance en général supérieure pour LOO
 - ▶ Estimation *k-fold* pessimiste car chaque modèle apprend sur $\frac{k-1}{k}N < N - 1$ données
- ▶ *Shuffle and split* vs *k-fold*
 - ▶ Pour *k-fold* le nombre de modèles (k) est lié à la proportion de données de test ($1/k$), *shuffle and split* moins contraignante
 - ▶ Pour *shuffle and split* certaines données ne sont dans aucun échantillon alors que d'autres sont dans plusieurs échantillons
- ▶ Quelle que soit la méthode, tous les partitionnements peuvent être explorés en parallèle (sur processeurs multi cœur ou plateformes distribuées)

Validation croisée : précautions à prendre

- ▶ Classification avec classes déséquilibrées : pour s'assurer de conserver les rapports entre les classes dans tous les découpages, utiliser
 - ▶ Un partitionnement adapté pour *k-fold* (par ex. `StratifiedKFold` dans Scikit-learn)
 - ▶ Un échantillonnage *stratifié* pour *shuffle and split* (par ex. `StratifiedShuffleSplit` dans Scikit-learn)
 - ▶ LOO peut être employée telle quelle
- ▶ Observations qui ne sont pas indépendantes
 - ▶ Séries temporelles : les observations successives sont corrélées, le découpage doit être fait par séquences sur les observations ordonnées et non après *shuffle* sur les observations individuelles
 - ▶ Données groupées : dans un même groupe, les observations ne sont pas indépendantes; les données de test doivent provenir de groupes différents de ceux dont sont issues les données d'apprentissage

Validation croisée : précautions à prendre

- ▶ Classification avec classes déséquilibrées : pour s'assurer de conserver les rapports entre les classes dans tous les découpages, utiliser
 - ▶ Un partitionnement adapté pour *k-fold* (par ex. `StratifiedKFold` dans `Scikit-learn`)
 - ▶ Un échantillonnage *stratifié* pour *shuffle and split* (par ex. `StratifiedShuffleSplit` dans `Scikit-learn`)
 - ▶ LOO peut être employée telle quelle
- ▶ Observations qui ne sont pas indépendantes
 - ▶ Séries temporelles : les observations successives sont corrélées, le découpage doit être fait par séquences sur les observations ordonnées et non après *shuffle* sur les observations individuelles
 - ▶ Données groupées : dans un même groupe, les observations ne sont pas indépendantes; les données de test doivent provenir de groupes différents de ceux dont sont issues les données d'apprentissage

Validation croisée : précautions à prendre

- ▶ Classification avec classes déséquilibrées : pour s'assurer de conserver les rapports entre les classes dans tous les découpages, utiliser
 - ▶ Un partitionnement adapté pour *k-fold* (par ex. `StratifiedKFold` dans Scikit-learn)
 - ▶ Un échantillonnage **stratifié** pour *shuffle and split* (par ex. `StratifiedShuffleSplit` dans Scikit-learn)
 - ▶ LOO peut être employée telle quelle
- ▶ Observations qui ne sont pas indépendantes
 - ▶ Séries temporelles : les observations successives sont corrélées, le découpage doit être fait par séquences sur les observations ordonnées et non après *shuffle* sur les observations individuelles
 - ▶ Données groupées : dans un même groupe, les observations ne sont pas indépendantes; les données de test doivent provenir de groupes différents de ceux dont sont issues les données d'apprentissage

Validation croisée : précautions à prendre

- ▶ Classification avec classes déséquilibrées : pour s'assurer de conserver les rapports entre les classes dans tous les découpages, utiliser
 - ▶ Un partitionnement adapté pour *k-fold* (par ex. `StratifiedKFold` dans Scikit-learn)
 - ▶ Un échantillonnage *stratifié* pour *shuffle and split* (par ex. `StratifiedShuffleSplit` dans Scikit-learn)
 - ▶ LOO peut être employée telle quelle
- ▶ Observations qui ne sont pas indépendantes
 - ▶ Séries temporelles : les observations successives sont corrélées, le découpage doit être fait par séquences sur les observations ordonnées et non après *shuffle* sur les observations individuelles
 - ▶ Données groupées : dans un même groupe, les observations ne sont pas indépendantes; les données de test doivent provenir de groupes différents de ceux dont sont issues les données d'apprentissage

Validation croisée : précautions à prendre

- ▶ Classification avec classes déséquilibrées : pour s'assurer de conserver les rapports entre les classes dans tous les découpages, utiliser
 - ▶ Un partitionnement adapté pour *k-fold* (par ex. `StratifiedKFold` dans Scikit-learn)
 - ▶ Un échantillonnage *stratifié* pour *shuffle and split* (par ex. `StratifiedShuffleSplit` dans Scikit-learn)
 - ▶ LOO peut être employée telle quelle
- ▶ Observations qui ne sont pas indépendantes
 - ▶ Séries temporelles : les observations successives sont corrélées, le découpage doit être fait *par séquences* sur les observations ordonnées et non après *shuffle* sur les observations individuelles
 - ▶ Données groupées : dans un même groupe, les observations ne sont pas indépendantes ; les données de test doivent provenir de groupes différents de ceux dont sont issues les données d'apprentissage

Validation croisée : précautions à prendre

- ▶ Classification avec classes déséquilibrées : pour s'assurer de conserver les rapports entre les classes dans tous les découpages, utiliser
 - ▶ Un partitionnement adapté pour *k-fold* (par ex. StratifiedKFold dans Scikit-learn)
 - ▶ Un échantillonnage *stratifié* pour *shuffle and split* (par ex. StratifiedShuffleSplit dans Scikit-learn)
 - ▶ LOO peut être employée telle quelle
- ▶ Observations qui ne sont pas indépendantes
 - ▶ Séries temporelles : les observations successives sont corrélées, le découpage doit être fait *par séquences* sur les observations ordonnées et non après *shuffle* sur les observations individuelles
 - ▶ Données groupées : dans un même groupe, les observations ne sont pas indépendantes ; les données de test doivent provenir de groupes différents de ceux dont sont issues les données d'apprentissage

Validation croisée : précautions à prendre

- ▶ Classification avec classes déséquilibrées : pour s'assurer de conserver les rapports entre les classes dans tous les découpages, utiliser
 - ▶ Un partitionnement adapté pour *k-fold* (par ex. StratifiedKFold dans Scikit-learn)
 - ▶ Un échantillonnage *stratifié* pour *shuffle and split* (par ex. StratifiedShuffleSplit dans Scikit-learn)
 - ▶ LOO peut être employée telle quelle
- ▶ Observations qui ne sont pas indépendantes
 - ▶ Séries temporelles : les observations successives sont corrélées, le découpage doit être fait *par séquences* sur les observations ordonnées et non après *shuffle* sur les observations individuelles
 - ▶ Données groupées : dans un même groupe, les observations ne sont pas indépendantes ; les données de test doivent provenir de groupes *differents* de ceux dont sont issues les données d'apprentissage

- ▶ Les jeux de données géographiques ne sont presque jamais indépendants
 - ▶ A minima, ils présentent une **autocorrélation spatiale**
 - ▶ Et parfois une corrélation temporelle pour les observations dans le temps
- ➡ attention à l'évaluation !

Validation croisée spatiale

- ▶ Définir des zones spatialement disjointes pour la validation croisée
- ▶ Par ex. :
 - ▶ Validation croisée par blocs
 - ▶ *Leave-one-out* spatial

Évaluation pour la classification : matrice de confusion

Prédiction \ Vérité terrain	Classe 1	Classe 2	Classe 3
Classe 1	10	2	3
Classe 2	7	15	0
Classe 3	3	5	10

- ▶ k^{e} ligne, l^{e} colonne : combien d'observations de la classe l ont été prédites comme étant de la classe k ?
- ▶ $C[k, l] = |\{x_i \text{ tel que } z_i = l \text{ et } f_{\theta}(x_i) = k\}|$

Exactitude (*overall accuracy*) : % d'observations bien classées

Avec K le nombre de classes, l'exactitude s'obtient par :

$$OA = \frac{\sum_{k=1}^K C[k, k]}{\sum_{k=1}^K \sum_{l=1}^K C[k, l]} \in [0, 1]$$

Évaluation pour la classification : matrice de confusion

Prédiction \ Vérité terrain	Classe 1	Classe 2	Classe 3
Classe 1	10	2	3
Classe 2	7	15	0
Classe 3	3	5	10

Précision et rappel

- ▶ Précision : fraction de prédictions correctes pour la classe k
- ▶ Rappel : fraction de la classe k correctement prédite

$$P[k] = \frac{C[k, k]}{\sum_{i=1}^K C[i, k]}, R[k] = \frac{C[k, k]}{\sum_{i=1}^K C[k, i]}$$

Évaluation pour la classification : matrice de confusion

Prédiction \ Vérité terrain	Classe 1	Classe 2	Classe 3
Classe 1	10	2	3
Classe 2	7	15	0
Classe 3	3	5	10

Score F_1

Moyenne harmonique entre précision et rappel

$$F_1[k] = 2 \cdot \frac{P[k]R[k]}{P[k] + R[k]}$$

$$\text{Macro-}F_1 = \frac{1}{K} \sum_{i=1}^K F1[i]$$

Matrice de confusion binaire

Dans le cas spécifique d'un problème à seulement deux classes (les positifs et les négatifs) :

Prédiction \ Vérité terrain	Classe \oplus	Classe \ominus
Classe \oplus	Vrais positifs	Faux positifs
Classe \ominus	Faux négatifs	Vrais négatifs

Précision et rappel

En notant tp les vrais positifs, fp les faux positifs et fn les faux négatifs :

$$P = \frac{tp}{tp + fp}, \quad R = \frac{tp}{tp + fn}$$

Évaluation pour la régression

Coefficient de détermination linéaire R^2

$$R^2 = 1 - \frac{\sum_{i=1}^n (y_i - f_\theta(x_i))^2}{(y_i - \bar{y})^2} \in]-\infty, 1]$$

où $\bar{y} = \frac{1}{n} \sum_{i=1}^n y_i$ est la moyenne des vérités terrain.

- ▶ Un modèle qui renvoie systématiquement la moyenne a un $R^2 = 0$
- ▶ Un modèle moins bon a un coefficient $R^2 < 0$

Erreur quadratique moyenne

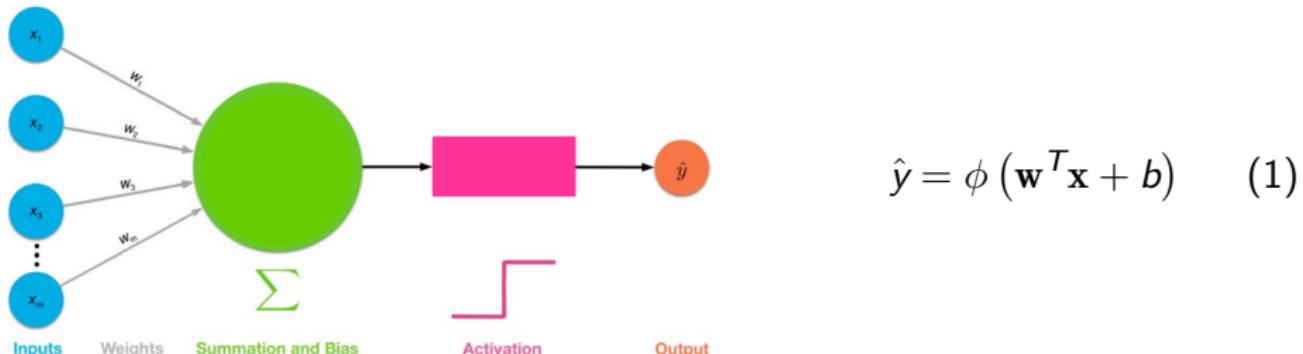
$$\text{RMSE} = \frac{1}{n} \sum_{i=1}^n \sqrt{(y_i - f_\theta(x_i))^2}$$

- ▶ Dans certains cas, on préfère l'erreur absolue qui est moins sensible aux données aberrantes.

Apprentissage profond

Neurone formel de McCulloch et Pitts 1943

- ▶ $f: \mathbf{x} \in \mathbb{R}^m \rightarrow \hat{y} \in \mathbb{R}$, entrée vecteur $\mathbf{x} = \{x_1, x_2, \dots, x_m\} \in \mathbb{R}^m$,
 - ▶ Poids $w_j \in \mathbb{R}$, coefficient de pondération sur l'entrée j , biais $b \in \mathbb{R}$
1. Fonction affine $s = \sum_{j=1}^m w_j x_j + b = \mathbf{w}^T \mathbf{x} + b$
 2. Fonction d'activation non-linéaire $\phi : \mathbb{R} \rightarrow \mathbb{R}$: sortie $\hat{y} = \phi(s)$

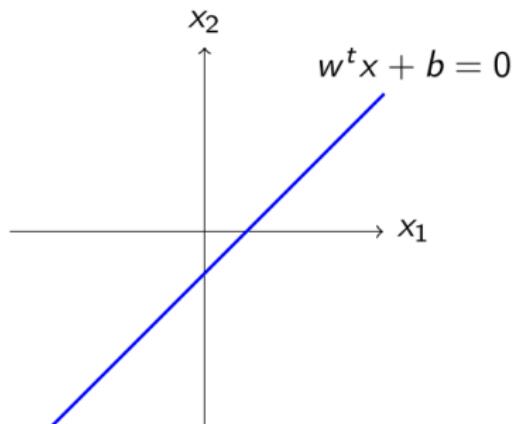


Neurone artificiel : partie linéaire

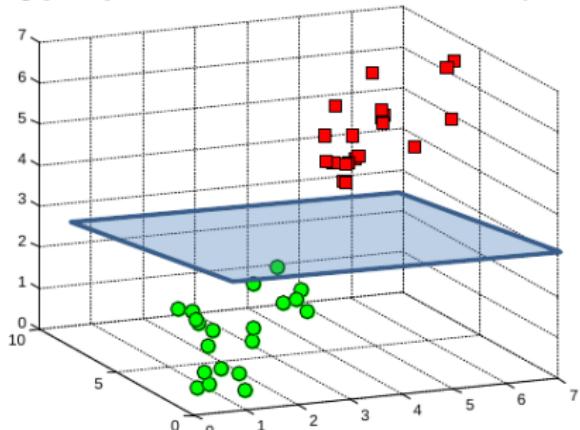
La partie linéaire est une projection affine : $s = \mathbf{w}^\top \mathbf{x} + b = \sum_{i=1}^m w_i x_i + b$

- ▶ $s = 0$ définit une **frontière linéaire**
- ▶ \mathbf{w} est le vecteur normal à la frontière
- ▶ biais b définit un écart par rapport à la frontière

Hyperplan en dimension 2 : droite



Hyperplan en dimension 3 : plan



$$\hat{y} = \phi(\mathbf{w}^T \mathbf{x} + b)$$

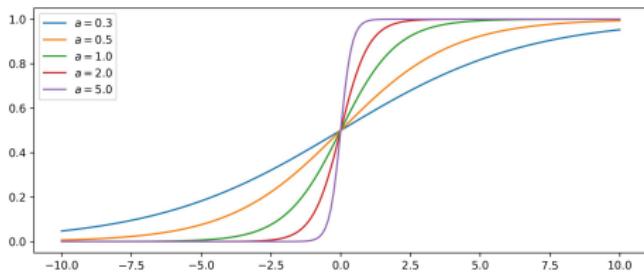
Plusieurs fonctions d'activation existent :

- ▶ Fonction identité : $\phi(x) = x$
- ▶ Fonction de Heaviside (échelon) : $\phi(x) = \begin{cases} 1 & \text{si } x \geq 0 \\ 0 & \text{sinon} \end{cases}$
- ▶ Sigmoïde : $\sigma(x) = \frac{1}{1+e^{-x}}$
- ▶ Tangente hyperbolique : $\tanh x = \frac{e^x - e^{-x}}{e^x + e^{-x}}$
- ▶ *Rectified Linear Unit (ReLU)* : $\phi(x) = \begin{cases} x & \text{si } x \geq 0 \\ 0 & \text{sinon} \end{cases}$

Activation sigmoïde

La sigmoïde est une fonction scalaire de $\sigma : \mathbb{R} \rightarrow [-1, 1]$

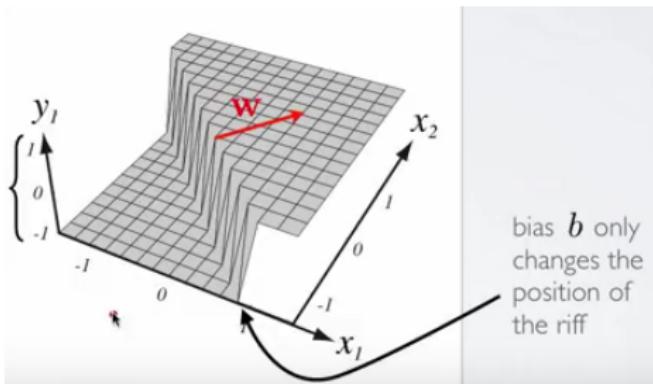
$$\sigma_a(z) = (1 + e^{-az})^{-1}$$



- ▶ $a \rightarrow \infty$: la sigmoïde se rapproche de l'échelon de Heaviside
- ▶ Sigmoïde : deux régimes
 - ▶ $x \approx 0 \rightarrow$ régime linéaire ($\sigma(x) \approx ax$)
 - ▶ $x \rightarrow \pm\infty \rightarrow$ régime de saturation ($\sigma(x) \approx \pm 1$)

Application à la classification binaire

- ▶ Chaque entrée x appartient soit à la classe 0, soit à la classe 1.
- ▶ Sortie du neurone artificiel (sigmoïde) : $\hat{y} = \frac{1}{1+e^{-a(w^T x + b)}}$
- ▶ Interprétation probabiliste $\Rightarrow \hat{y} \sim P(1|x)$
 - ▶ L'entrée x est classée comme classe 1 si $P(1|x) > 0.5 \Leftrightarrow w^T x + b > 0$
 - ▶ L'entrée x est classée comme classe 0 si $P(1|x) < 0.5 \Leftrightarrow w^T x + b < 0$
 $\Rightarrow \text{signe}(w^T x + b)$: la frontière de classification est linéaire dans l'espace d'entrée !

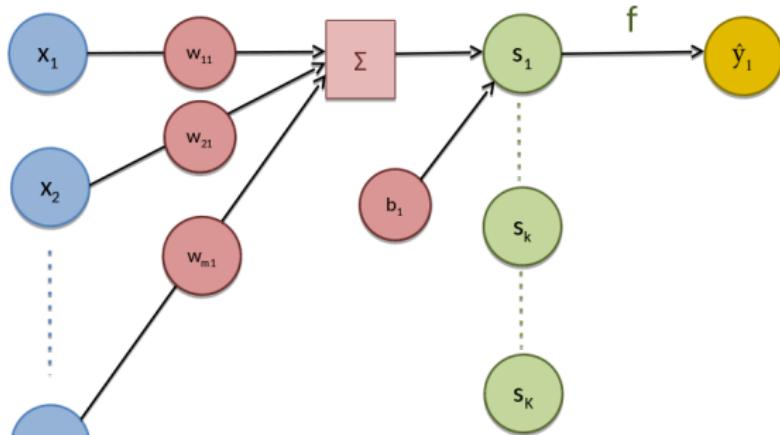


Du neurone formel au réseau de neurones

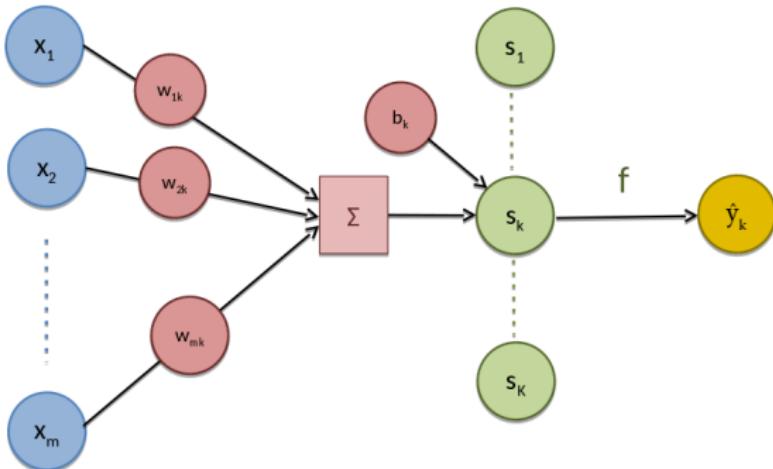
- ▶ Neurone artificiel :
 1. Une seule sortie scalaire \hat{y}
 2. Frontière linéaire pour la classification binaire
- ▶ Sortie scalaire unique : expressivité limitée pour la plupart des tâches
 - ▶ Comment gérer la classification multi-classe ?
 - ⇒ utiliser plusieurs neurones de sortie plutôt qu'un seul !
 - ⇒ modèle neuronal dit **Perceptron** (ROSENBLATT 1957)

Du neurone formel au réseau de neurones

- ▶ Neurone artificiel :
 1. Une seule sortie scalaire \hat{y}
 2. Frontière linéaire pour la classification binaire
- ▶ Sortie scalaire unique : expressivité limitée pour la plupart des tâches
 - ▶ Comment gérer la classification multi-classe ?
 - ⇒ utiliser plusieurs neurones de sortie plutôt qu'un seul !
 - ⇒ modèle neuronal dit **Perceptron** (ROSENBLATT 1957)



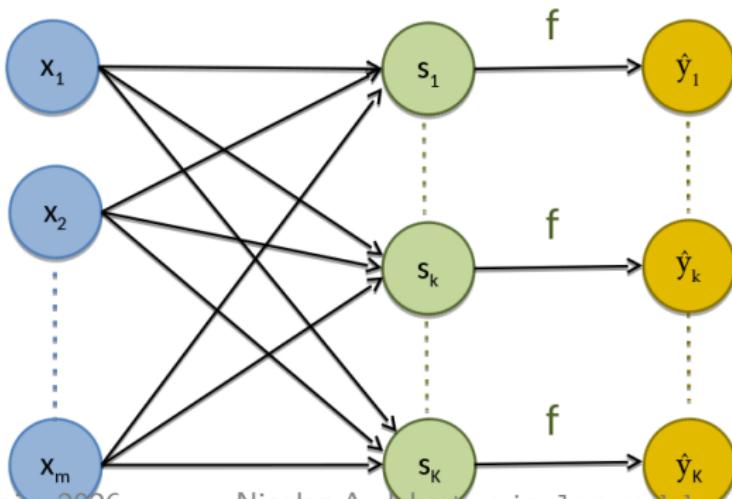
Perceptron et classification multi-classe



- ▶ Chaque sortie \hat{y}_k est un neurone artificiel avec sa propre :
 - ▶ Transformation affine : $s_k = \mathbf{w}_k^T \mathbf{x} + b_k$
 - ▶ poids $\mathbf{w}_k = \{w_{1,k}, w_{2,k}, \dots, w_{m,k}\} \in \mathbb{R}^m$, biais $b_k \in \mathbb{R}$
 - ▶ Activation non-linéaire σ : $\hat{y}_k = \sigma(s_k)$

Perceptron en résumé

- ▶ Entrée $\mathbf{x} \in \mathbb{R}^m$ ($1 \times m$), sortie $\hat{\mathbf{y}}$: concaténation de k neurones
- ▶ Transformation affine : $\mathbf{s} = \mathbf{W}\mathbf{x} + \mathbf{b}$
 - ▶ avec \mathbf{W} une matrice de poids de dimensions $m \times k$ – les colonnes sont les vecteurs \mathbf{w}_k ,
 - ▶ et \mathbf{b} le vecteur de biais – dimension $1 \times k$.
- ▶ Activation non-linéaire élément par élément (*pointwise*) : $\hat{\mathbf{y}} = \sigma(\mathbf{s})$

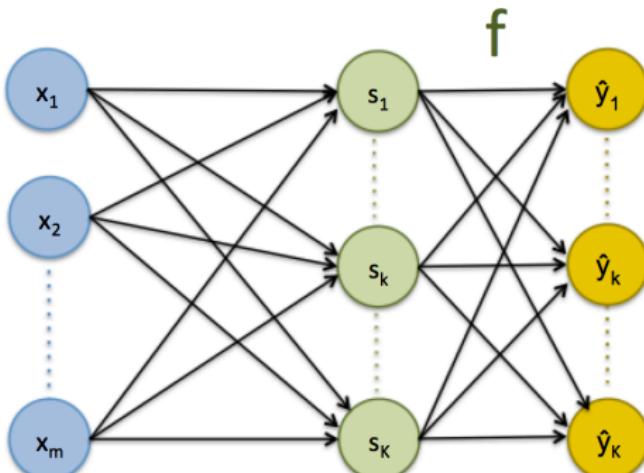


Application à la classification multi-classe

- ▶ Comment généraliser la sigmoïde au cas à plusieurs classes ?
 - ▶ on cherche f telle que $f(s_j) = P(j|x)$ (interprétation probabiliste)
 - ▶ contrainte : $\sum_{j=1}^k f(s_j) = 1$.

▶ Activation softmax :

$$\hat{y}_i = f(s_i) = \frac{e^{s_i}}{\sum_{j=1}^k e^{s_j}}$$



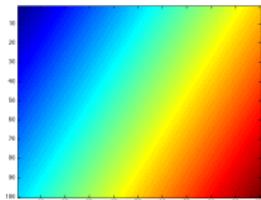
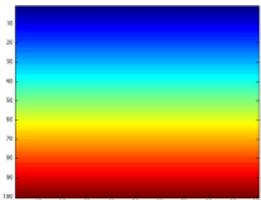
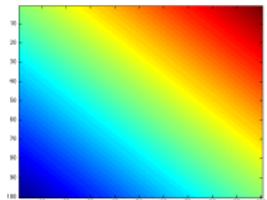
Exemple jouet en 2D pour trois classes

- ▶ $\mathbf{x} = \{x_1, x_2\} \in [-5; 5] \times [-5; 5]$, $\hat{y} \in \{0, 1, 2\}$ (3 classes)

Transformation affine : $\mathbf{w}_1 = [1; 1], b_1 = -2$ $\mathbf{w}_2 = [0; -1], b_2 = 1$ $\mathbf{w}_3 = [1; -0.5], b_3 = 10$

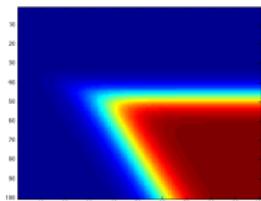
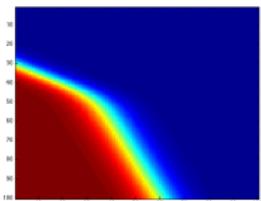
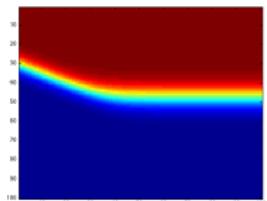
affine :

$$s_k = \mathbf{w}_k^T \mathbf{x} + b_k$$



Softmax :

$$P(k|\mathbf{x}, \mathbf{W})$$



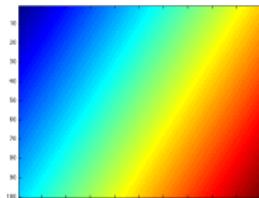
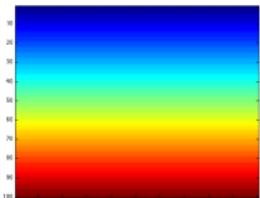
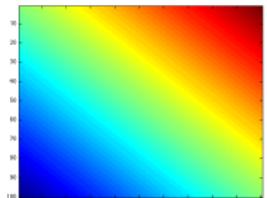
Exemple jouet en 2D pour trois classes

- ▶ $\mathbf{x} = \{x_1, x_2\} \in [-5; 5] \times [-5; 5]$, $\hat{y} \in \{0, 1, 2\}$ (3 classes)

Transformation affine : $\mathbf{w}_1 = [1; 1], b_1 = -2$ $\mathbf{w}_2 = [0; -1], b_2 = 1$ $\mathbf{w}_3 = [1; -0.5], b_3 = 10$

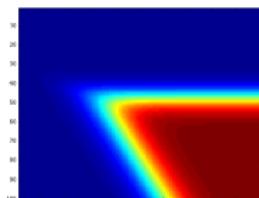
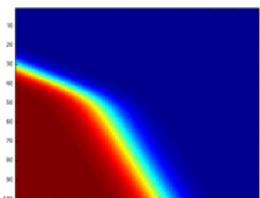
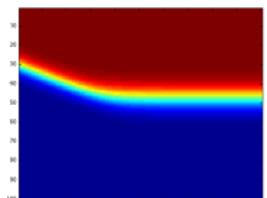
affine :

$$s_k = \mathbf{w}_k^T \mathbf{x} + b_k$$



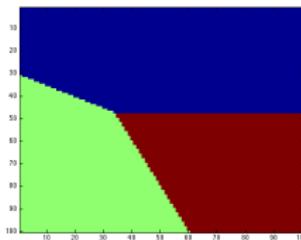
Softmax :

$$P(k|\mathbf{x}, \mathbf{W})$$



Classe prédite :

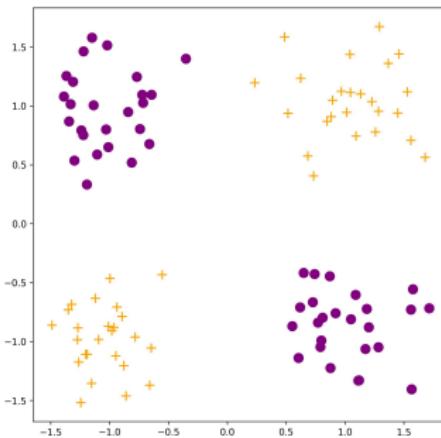
$$k^* = \arg \max_k P(k|\mathbf{x}, \mathbf{W})$$



Régression logistique

On appelle **régression logistique** ce modèle de perceptron à une couche d'entrée et une couche de sortie

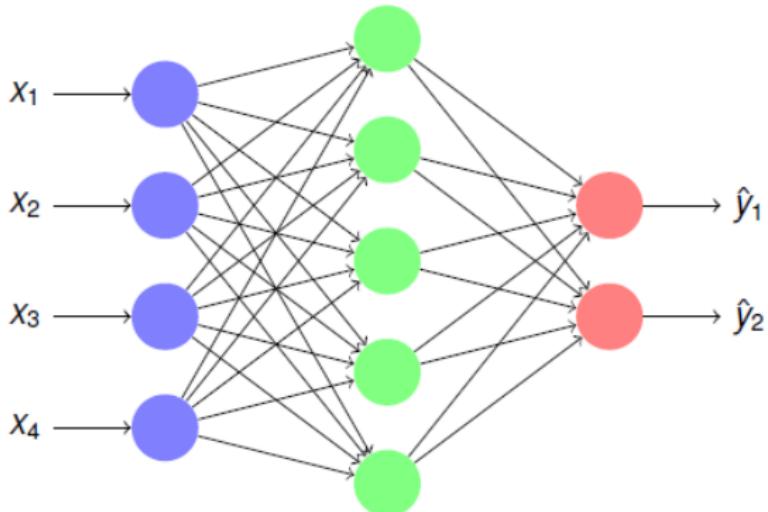
- ▶ Limite : la régression logistique ne peut exprimer que des frontières de décisions **linéaires**



⇒ le perceptron échoue sur un problème en apparence très simple...

Au-delà des frontières linéaires

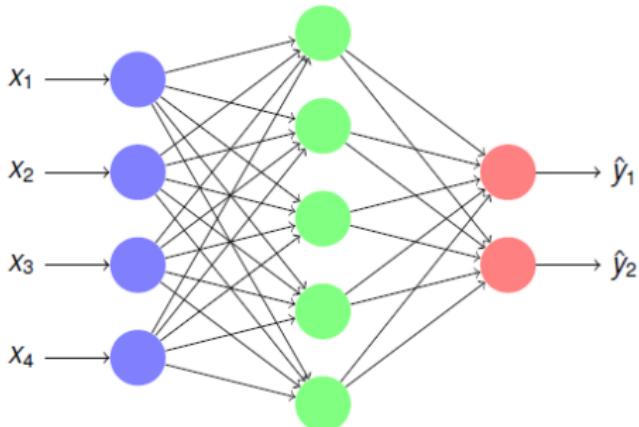
- ▶ La régression logistique est limitée aux frontières linéaires.
⇒ **Solution** : ajouter une couche de neurones !
- ▶ Entrée : $\mathbf{x} \in \mathbb{R}^m$, ici par exemple $m = 4$.
- ▶ Sortie : $\hat{\mathbf{y}} \in \mathbb{R}^k$ (k classes), par ex. $k = 2$.
- ▶ **Couche « cachée »** de taille l : $\mathbf{h} \in \mathbb{R}^l$, par ex. $l = 5$



Perceptron multi-couche

- ▶ **Couche cachée h** : projection de l'entrée x vers un nouvel espace \mathbb{R}^l

- ▶ $\hat{y} : h = \phi(W_s h + b_s)$
- ▶ h est une représentation intermédiaire de x qui sera ensuite classée par la régression logistique.



Un réseau de neurones avec (au moins) une couche cachée est un perceptron multi-couche (*multi-layer perceptron, MLP*)

Attention !

La non-linéarité de la fonction d'activation est indispensable pour le perceptron multi-couche.

- ▶ Peut-on approcher n'importe quelle fonction à l'aide d'un PMC ?

Théorème d'approximation universelle

Hypothèses : Soit f une fonction continue sur \mathbb{R}^d et ϕ une fonction d'activation non-polynomiale.

Alors, $\forall \epsilon > 0$, il existe un PMC à une couche cachée de $n \in \mathbb{N}$ neurones $\hat{f}: \mathbf{x} \rightarrow \mathbf{W}_s \phi(\mathbf{Wx} + \mathbf{b})$ qui approche f à ϵ près : $\max \|f(\mathbf{x}) - \hat{f}(\mathbf{x})\| < \epsilon$

- ▶ Démontré pour les fonctions d'activation sigmoïde par CYBENKO 1989
- ▶ Étendu à l'ensemble des perceptrons multi-couche par HORNIK 1991

Limitations

- ▶ Une seule couche cachée suffit mais n n'est pas borné !
- ▶ Pas de méthode pour déterminer les paramètres \mathbf{W}_s , \mathbf{W} et \mathbf{b} .

- ▶ Peut-on approcher n'importe quelle fonction à l'aide d'un PMC ?

Théorème d'approximation universelle

Hypothèses : Soit f une fonction continue sur \mathbb{R}^d et ϕ une fonction d'activation non-polynomiale.

Alors, $\forall \epsilon > 0$, il existe un PMC à une couche cachée de $n \in \mathbb{N}$ neurones $\hat{f}: \mathbf{x} \rightarrow \mathbf{W}_s \phi(\mathbf{Wx} + \mathbf{b})$ qui approche f à ϵ près : $\max \|f(\mathbf{x}) - \hat{f}(\mathbf{x})\| < \epsilon$

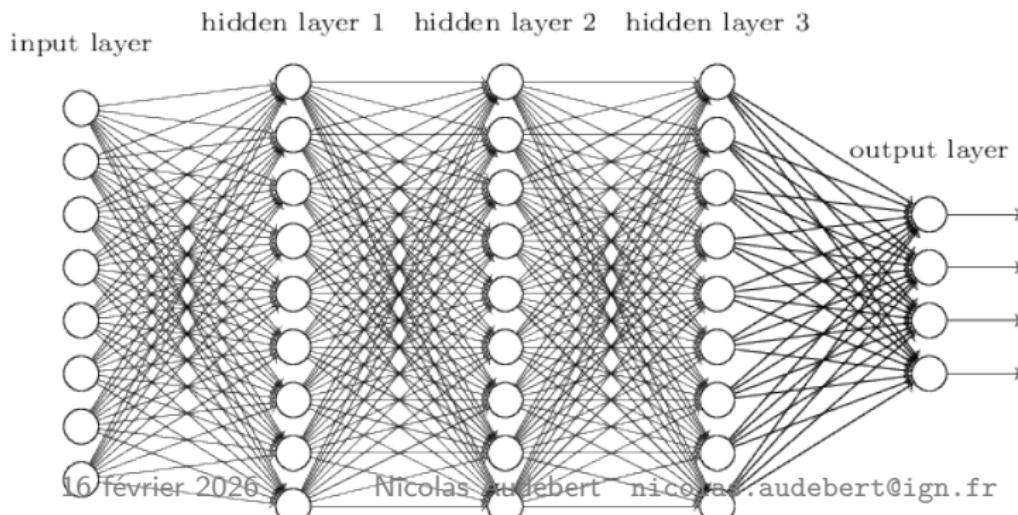
- ▶ Démontré pour les fonctions d'activation sigmoïde par CYBENKO 1989
- ▶ Étendu à l'ensemble des perceptrons multi-couche par HORNIK 1991

Limitations

- ▶ Une seule couche cachée suffit mais n n'est pas borné !
- ▶ Pas de méthode pour déterminer les paramètres \mathbf{W}_s , \mathbf{W} et \mathbf{b} .

Réseaux de neurones profonds

- ▶ Augmenter le nombre de couche cachées : réseaux de neurones profonds (*deep neural networks*)
- ▶ Chaque couche \mathbf{h}^l projette les activations de la couche précédente \mathbf{h}^{l-1} dans un nouvel espace intermédiaire
- ⇒ construction incrémentale d'un espace intermédiaire de représentation utile pour la tâche visée



- ▶ Neurone artificiel : représentation grossière d'un neurone biologique (combinaison linéaire + fonction d'activation)
- ▶ Perceptron : une matrice de poids projetant la couche d'entrée vers la couche de sortie \Rightarrow limité à des frontières de décision linéaires
- ▶ Réseaux de neurones profonds : perceptrons multi-couche \Rightarrow frontières non-linéaires

En pratique

- ▶ Comment déterminer les paramètres du modèle (W, b) ?
- ➡ **apprentissage supervisé** à l'aide de techniques d'optimisation

Soit $f: \mathbb{R}^d \rightarrow \mathbb{R}$ une fonction réelle à d variables **différentiable**.

- ▶ On note $\nabla f(\mathbf{x})$ le **gradient** de f au point $\mathbf{x} = (x_1, x_2, \dots, x_d) \in \mathbb{R}^d$.

$$\nabla f = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \dots, \frac{\partial f}{\partial x_d} \right)$$

- ▶ Si \mathbf{x}^* est un *extremum local* de f , alors $\nabla f(\mathbf{x}^*) = 0$.
 - ▶ Si f est convexe ou concave, gradient nul \Leftrightarrow *extremum global*.

Problème : on cherche à *minimiser* f , c'est-à-dire trouver \mathbf{x}^* tel que $f(\mathbf{x}^*) < f(\mathbf{x})$ pour $\mathbf{x} \in \mathbb{R}^d$.

Algorithme du gradient Cauchy 1847

Soit $\mathbf{x}^{(0)} \in \mathbb{R}^d$ un point de départ et $\epsilon > 0$. On définit la suite $\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \dots$ telle que :

$$\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} - \eta_t \nabla f(\mathbf{x}^{(t)})$$

Si $\|\nabla f(\mathbf{x}^{(t)})\| \leq \epsilon$, on s'arrête.

- ▶ η_t peut être obtenu par divers moyens, comme une recherche linéaire.
Dans notre cas, on fixera $\eta_t = \eta > 0$ constant.

Optimisation du *multi-layer perceptron* (MLP)

- ▶ Entrée $\mathbf{x} \in \mathbb{R}^m$, sortie $\mathbf{y} \in \mathbb{R}^p$
- ▶ Paramètres \mathbf{w} (poids et biais du modèle)
- ▶ Le MLP est un modèle paramétrique $\mathbf{x} \Rightarrow \mathbf{y} : f_{\mathbf{w}}(\mathbf{x}_i) = \hat{\mathbf{y}}_i$
- ▶ Apprentissage **supervisé** : jeu d'entraînement annoté
 $\mathcal{A} = \{(\mathbf{x}_i, \mathbf{y}_i^*)\}_{i \in \{1, 2, \dots, N\}}$
 - ▶ Fonction de coût $\mathcal{L} = \sum_{i=1}^N \mathcal{L}(\hat{\mathbf{y}}_i, \mathbf{y}_i^*)$ entre la prédiction du modèle et l'annotation
- ▶ *Hypothèses* : les paramètres $\mathbf{w} \in \mathbb{R}^d$ sont des réels, \mathcal{L} est différentiable.
- ▶ Gradient $\nabla_{\mathbf{w}} = \frac{\partial \mathcal{L}}{\partial \mathbf{w}}$: direction de plus forte pente dans l'*espace des paramètres* permettant de faire décroître le coût \mathcal{L}

Considérons le vecteur des paramètres $\mathbf{w} \in \mathbb{R}^d = (w_1, w_2, \dots, w_d)$.

Le *gradient* du coût \mathcal{L} par rapport à \mathbf{w} est le vecteur des dérivées partielles du coût par rapport à chaque paramètre w_i :

$$\nabla_{\mathbf{w}} \mathcal{L} = \left(\frac{\partial \mathcal{L}(\hat{\mathbf{y}}, \mathbf{y}^*; \mathbf{w})}{\partial w_1} \quad \frac{\partial \mathcal{L}(\hat{\mathbf{y}}, \mathbf{y}^*; \mathbf{w})}{\partial w_2} \quad \dots \quad \frac{\partial \mathcal{L}(\hat{\mathbf{y}}, \mathbf{y}^*; \mathbf{w})}{\partial w_d} \right)$$

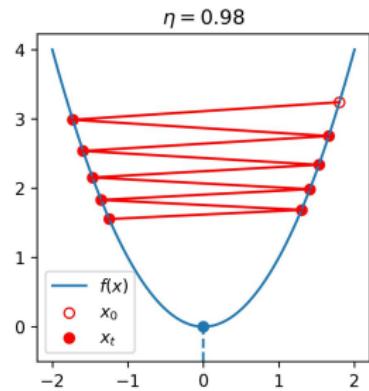
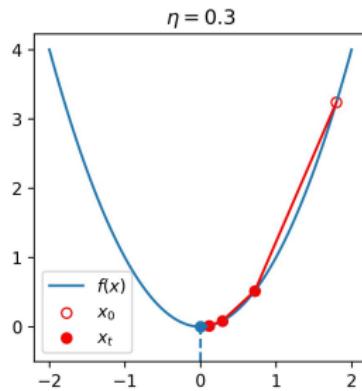
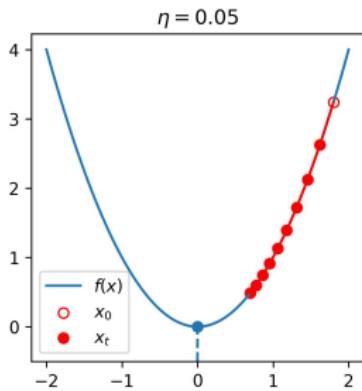
Descente de gradient

1. Initialisation aléatoire des paramètres \mathbf{w}
2. Mise à jour à l'itération t :
$$\mathbf{w}_i^{(t+1)} = \mathbf{w}_i^{(t)} - \eta \frac{\partial \mathcal{L}}{\partial \mathbf{w}_i}$$
3. Répéter l'étape 2 jusqu'à convergence, par ex. $\|\nabla_{\mathbf{w}} \mathcal{L}\|^2 \approx 0$ (le coût cesse de décroître).

Pas d'apprentissage

Équation de mise à jour : $\mathbf{w}_i^{(t+1)} = \mathbf{w}_i^{(t)} - \eta \frac{\partial \mathcal{L}}{\partial \mathbf{w}_i}$ **η pas d'apprentissage**
(*learning rate*)

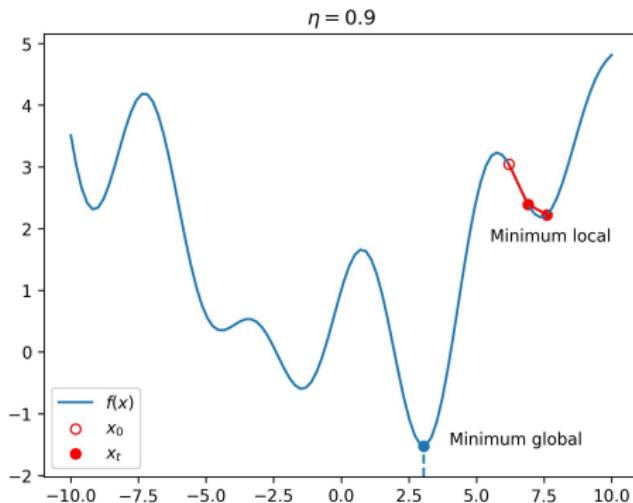
- ▶ Peut-on garantir la convergence de l'algorithme vers un minimum ? \Rightarrow seulement pour une valeur de η « bien choisie ».



Minimum local ou global

Équation de mise à jour : $\mathbf{w}_i^{(t+1)} = \mathbf{w}_i^{(t)} - \eta \frac{\partial \mathcal{L}}{\partial \mathbf{w}_i}$

- ▶ **Le minimum trouvé est-il le meilleur (minimum global) ?**
- ⇒ garanti seulement si la fonction $\mathcal{L}(\mathbf{w})$ à minimiser est **convexe**
- ▶ dans la quasi-totalité des cas, la fonction à minimiser n'est pas convexe par rapport aux paramètres du modèle...



Fonctions de coût pour le perceptron

Perceptron : $\hat{y} = \phi(x_i \mathbf{W} + \mathbf{b})$

Régression

- ▶ L2 (*mean-squared error*) : $\|\hat{\mathbf{y}} - \mathbf{y}\|^2$
- ▶ L1 (*mean-absolute error*) : $|\hat{\mathbf{y}} - \mathbf{y}|$

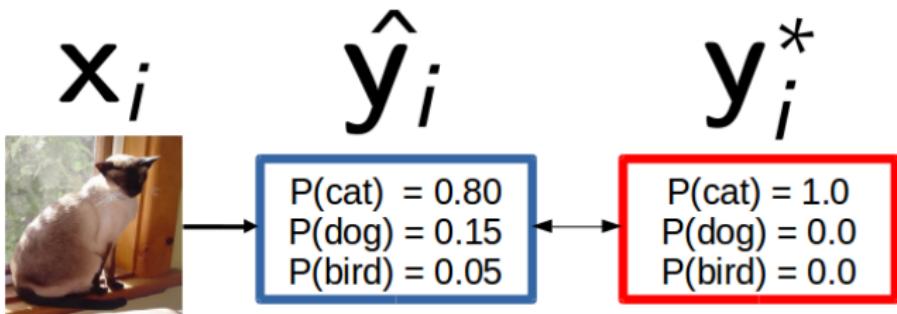
Classification à K classes

Idéalement, on aimerait utiliser une fonction de coût proche de l'*accuracy* :

$$\blacktriangleright \mathcal{L}_{0/1}(\hat{y}_i, y_i^*) = \begin{cases} 1 & \text{si } \hat{y}_i \neq y_i^* \\ 0 & \text{sinon} \end{cases} : \text{perte 0/1}$$

⇒ mais cette fonction de coût est non-différentiable !

Classification : encodage des variables



- ▶ Vérité terrain (étiquettes de supervision) $y_i^* \in \{1, 2, \dots, K\}$
- ▶ Encodage *one hot* pour chaque étiquette :

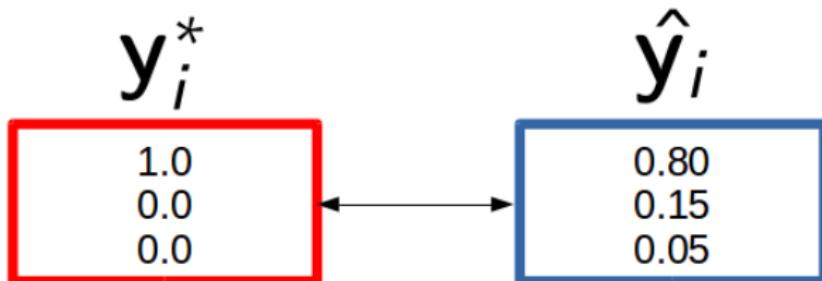
$$y_{c,i}^* = \begin{cases} 1 & \text{si } c \text{ est la classe de } x_i \\ 0 & \text{sinon} \end{cases}$$

$$\mathbf{y}_i^* = \left(0, 0, \dots, \underbrace{1}_{c^{\text{e}} \text{ composante}}, \dots, 0 \right)$$

Entropie croisée

- ▶ \mathcal{L}_{CE} mesure la différence entre la distribution de probabilité y_i^* de la vérité terrain et la distribution prédite \hat{y}_i

$$\mathcal{L}_{CE}(y_i^*, \hat{y}_i) = - \sum_{c=1}^K \underbrace{y_{c,i}^*}_{= 0 \text{ sauf pour } c^*} \log(\hat{y}_{c,i}) = - \log(\hat{y}_{c^*,i})$$



$$KL(y_i^*, \hat{y}_i) = -\log(\hat{y}_{c^*,i}) = -\log(0.8) \approx 0.22$$

Entraînement de la régression logistique

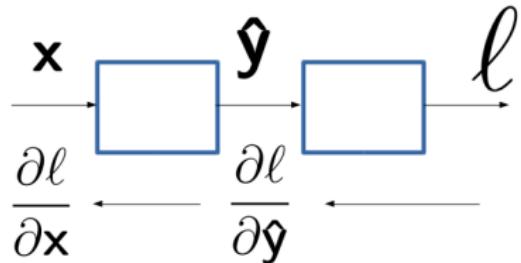
- ▶ $\mathcal{L}_{CE}(\mathbf{W}, \mathbf{b}) = \frac{1}{N} \sum_{i=1}^N \mathcal{L}_{CE}(\hat{y}_i, y_i^*) = -\frac{1}{N} \sum_{i=1}^N \log(\hat{y}_{c^*,i})$
- ▶ \mathcal{L}_{CE} est une borne supérieure de la perte $\mathcal{L}_{0/1}$
 - ▶ minimiser $\mathcal{L}_{CE} \Rightarrow$ minimiser la perte $\mathcal{L}_{0/1}$
- ▶ \mathcal{L}_{CE} est différentiable \Rightarrow **descente de gradient !**
 - ▶ \mathcal{L}_{CE} est convexe mais seulement par rapport à \mathbf{y} (pas par rapport à \mathbf{w})

Calcul du gradient

Descente de gradient : $\mathbf{W}^{(t+1)} = \mathbf{W}^{(t)} - \eta \frac{\partial \mathcal{L}_{CE}}{\partial \mathbf{W}}$ et $\mathbf{b}^{(t+1)} = \mathbf{b}^{(t)} - \eta \frac{\partial \mathcal{L}_{CE}}{\partial \mathbf{b}}$

- ▶ Principal problème : calculer $\frac{\partial \mathcal{L}_{CE}}{\partial \mathbf{W}}$
- ▶ Propriété centrale : *chain rule* (dérivation des fonctions composées)

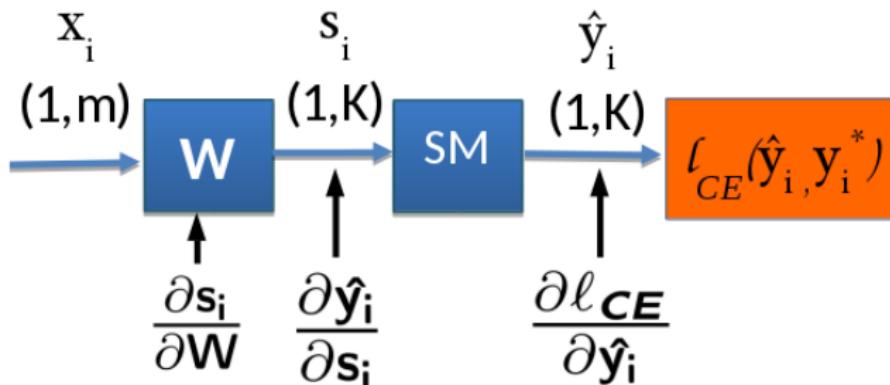
$$\frac{\partial \mathbf{x}}{\partial \mathbf{z}} = \frac{\partial \mathbf{x}}{\partial \mathbf{y}} \frac{\partial \mathbf{y}}{\partial \mathbf{z}}$$



$$\frac{\partial \mathcal{L}}{\partial \mathbf{x}} = \frac{\partial \mathcal{L}}{\partial \hat{\mathbf{y}}} \frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{x}}$$

► Régression logistique :

$$\frac{\partial \mathcal{L}_{CE}}{\partial \mathbf{W}} = \frac{\partial \mathcal{L}_{CE}}{\partial \hat{\mathbf{y}}_i} \frac{\partial \hat{\mathbf{y}}_i}{\partial \mathbf{s}_i} \frac{\partial \mathbf{s}_i}{\partial \mathbf{W}}$$



Régression logistique : calcul des gradients (1/2)

$$\frac{\partial \mathcal{L}_{CE}}{\partial \mathbf{W}} = \frac{\partial \mathcal{L}_{CE}}{\partial \hat{\mathbf{y}}_i} \frac{\partial \hat{\mathbf{y}}_i}{\partial \mathbf{s}_i} \frac{\partial \mathbf{s}_i}{\partial \mathbf{W}}, \quad \mathcal{L}_{CE}(\hat{\mathbf{y}}_i, \mathbf{y}_i^*) = -\log(\hat{y}_{c^*, i})$$

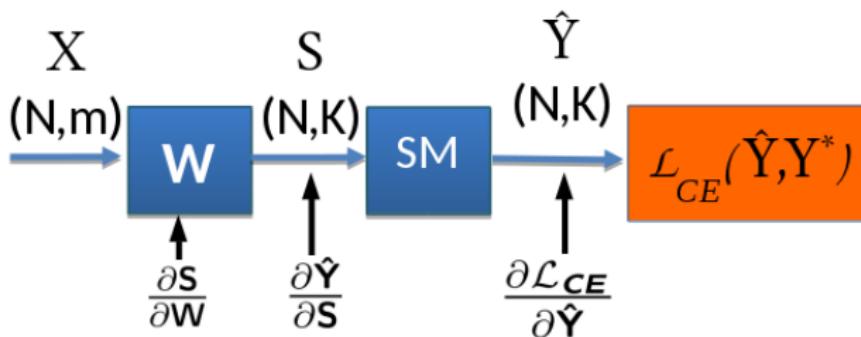
Mise à jour pour 1 exemple :

- ▶ $\frac{\partial \mathcal{L}_{CE}}{\partial \hat{\mathbf{y}}_i} = \frac{-1}{\hat{y}_{c^*, i}} = \frac{-1}{\hat{\mathbf{y}}_i} \odot \delta_{c, c^*}$ avec $\delta_{c, c^*} = (0, \dots, \underbrace{1}_{c^*}, \dots, 0)$
- ▶ $\frac{\partial \mathcal{L}_{CE}}{\partial \mathbf{s}_i} = \hat{\mathbf{y}}_i - \mathbf{y}_i^* = \delta_i^y$
- ▶ $\frac{\partial \mathcal{L}_{CE}}{\partial \mathbf{W}} = \mathbf{x}_i^T \delta_i^y$

Régression logistique : calcul des gradients (2/2)

Pour l'ensemble du jeu de données \mathbf{X} ($N \times m$), étiquettes \mathbf{Y}^* et prédictions $\hat{\mathbf{Y}}$ ($N \times K$) :

- ▶ $\mathcal{L}_{CE}(\mathbf{W}, \mathbf{b}) = -\frac{1}{N} \sum_{i=1}^N \log(\hat{y}_{c^*,i}), \quad \frac{\partial \mathcal{L}_{CE}}{\partial \mathbf{W}} = \frac{\partial \mathcal{L}_{CE}}{\partial \hat{\mathbf{Y}}} \frac{\partial \hat{\mathbf{Y}}}{\partial \mathbf{S}} \frac{\partial \mathbf{S}}{\partial \mathbf{W}}$
- ▶ $\frac{\partial \mathcal{L}_{CE}}{\partial \mathbf{S}} = \hat{\mathbf{Y}} - \mathbf{Y}^* = \Delta^y$
- ▶ $\frac{\partial \mathcal{L}_{CE}}{\partial \mathbf{W}} = \mathbf{X}^T \Delta^y$



- ▶ La fonction de coût est calculée sur tout le jeu de données d'entraînement :

$$\mathcal{L} = \frac{1}{N} \sum_{i=1}^N \mathcal{L}(\hat{\mathbf{y}}_i, \mathbf{y}_i^*; \mathbf{w}, \mathbf{b})$$

- ▶ Optimisation par descente de gradient :

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta \frac{\partial \mathcal{L}}{\partial \mathbf{w}} \left(\mathbf{w}^{(t)} \right)$$

- ▶ La complexité du calcul du gradient $\nabla_{\mathbf{w}}^{(t)} = \frac{1}{N} \sum_{i=1}^N \frac{\partial \mathcal{L}(\hat{\mathbf{y}}_i, \mathbf{y}_i^*)}{\partial \mathbf{w}} (\mathbf{w}^{(t)})$ croît linéairement avec :
 - ▶ la dimensionnalité de \mathbf{w} (nombre de paramètres du modèle),
 - ▶ N , la taille du jeu de données (nombre d'exemples d'apprentissage).

⇒ **coûteux, même pour des modèles de dimensionnalité modérée et des jeux de données de taille moyenne !**

Descente de gradient stochastique

- ▶ **Solution** : approximation du vrai gradient

$\nabla_{\mathbf{w}}^{(t)} = \frac{1}{N} \sum_{i=1}^N \frac{\partial \mathcal{L}(\hat{\mathbf{y}}_i, \mathbf{y}_i^*)}{\partial \mathbf{w}} (\mathbf{w}^{(t)})$ sur un échantillon d'exemples (un *batch* ou « lot »)

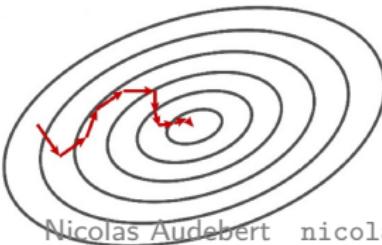
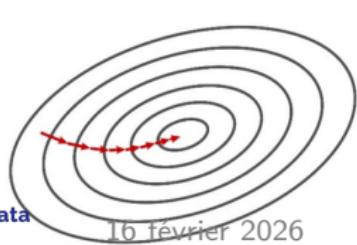
⇒ Stochastic Gradient Descent (SGD)

- ▶ Version *online* : mise à jour à partir d'un seul exemple

$$\nabla_{\mathbf{w}}^{(t)} \approx \frac{\partial \mathcal{L}(\hat{\mathbf{y}}_i, \mathbf{y}_i^*)}{\partial \mathbf{w}} (\mathbf{w}^{(t)})$$

- ▶ Version par mini-batch : mise à jour sur $B \ll N$ exemples :

$$\nabla_{\mathbf{w}}^{(t)} \approx \frac{1}{B} \sum_{i=1}^B \frac{\partial \mathcal{L}(\hat{\mathbf{y}}_i, \mathbf{y}_i^*)}{\partial \mathbf{w}} (\mathbf{w}^{(t)})$$

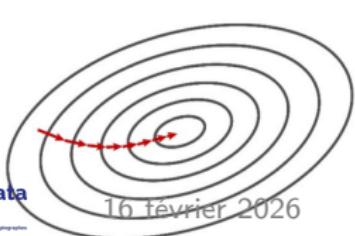


Avantages et inconvénients

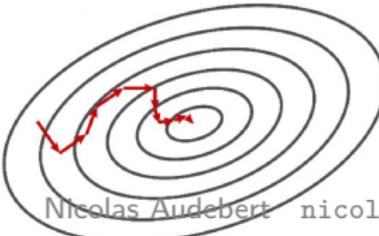
La descente de gradient stochastique produit une approximation du véritable gradient ∇_w .

- estimation bruitée, pouvant envoyer une direction incorrecte (pente forte pour un exemple mais pas pour les autres),
- sensibilité aux *outliers* pour la version en ligne,
- + mises à jour des poids plus nombreuses car itérations moins coûteuses :
 $\times N$ mises à jour (online), $\times \frac{N}{B}$ mises à jour (mini-batch)
- + à nombre de calculs égal, convergence plus rapide

La SGD est incontournable pour la convergence des modèles profonds sur des jeux de données massifs.

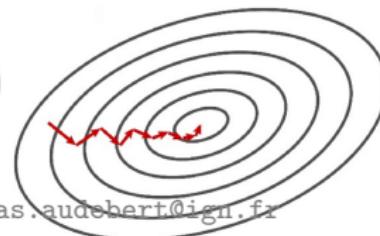


16 février 2026



Nicolas Audebert

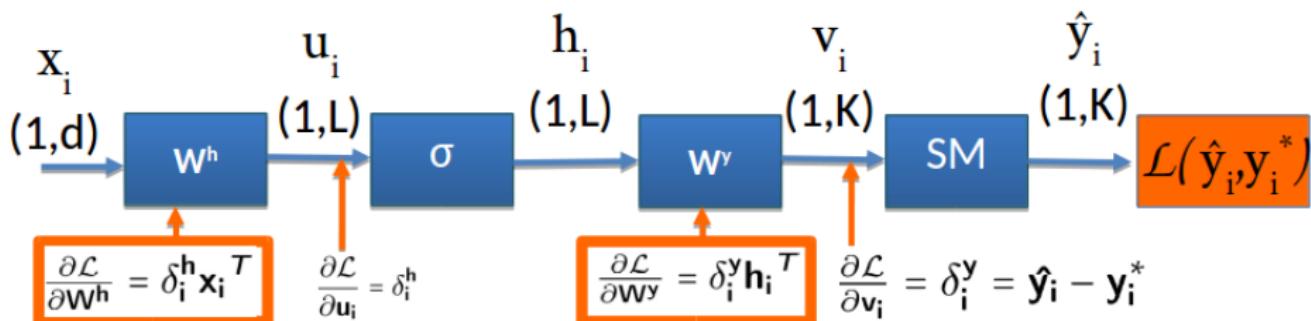
nicolas.audebert@ign.fr



Entraînement du perceptron : rétropropagation

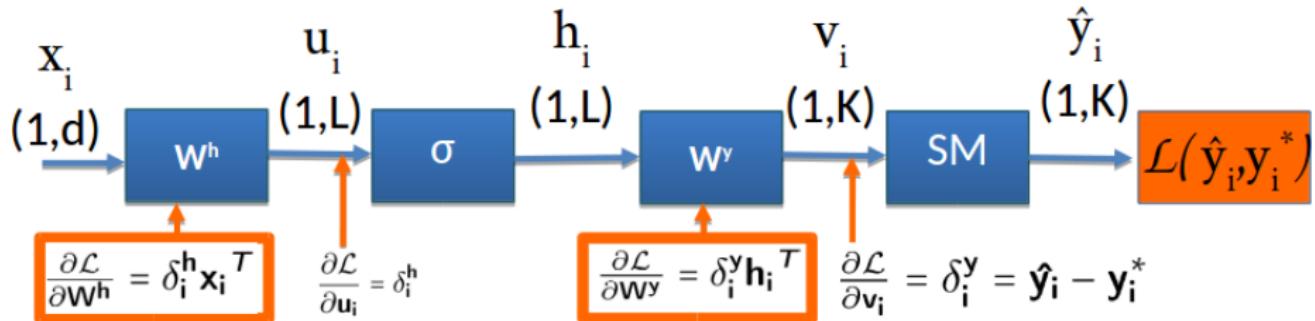
- Passer de la régression logistique au perceptron multi-couche implique l'ajout d'une couche cachée (+ sigmoïde)
- Entraînement : apprendre les paramètres \mathbf{W}^y et \mathbf{W}^h (+ bias) par **rétropropagation**

$$\frac{\partial \mathcal{L}_{CE}}{\partial \mathbf{W}^y} = \frac{1}{N} \sum_{i=1}^N \frac{\partial \mathcal{L}_{CE}}{\partial \mathbf{W}^y}$$
 et
$$\frac{\partial \mathcal{L}_{CE}}{\partial \mathbf{W}^h} = \frac{1}{N} \sum_{i=1}^N \frac{\partial \mathcal{L}_{CE}}{\partial \mathbf{W}^h}$$



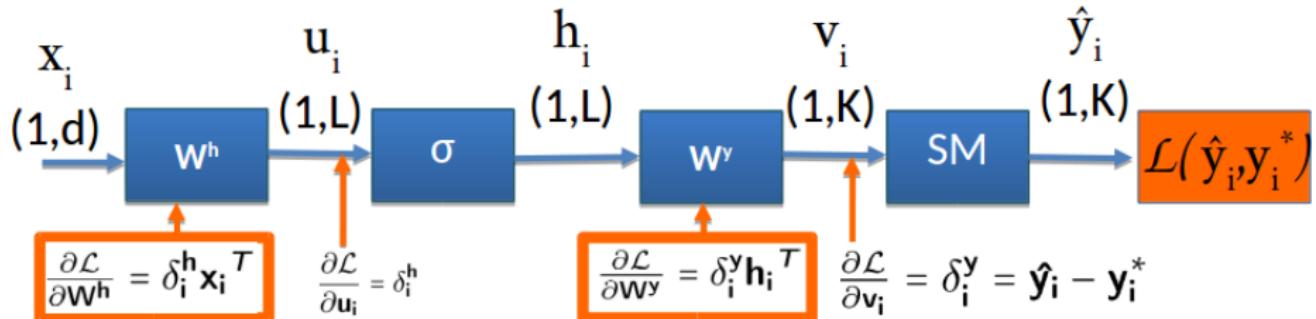
- La dernière couche est équivalente à une régression logistique.
- Couche cachée : $\frac{\partial \mathcal{L}_{CE}}{\partial \mathbf{W}^h} = \mathbf{x}_i^T \frac{\partial \mathcal{L}_{CE}}{\partial u_i} \Rightarrow$ **calcul de** $\frac{\partial \mathcal{L}_{CE}}{\partial u_i} = \delta_i^h$

Rétropropagation du gradient



- ▶ Coût pour un seul exemple : $\mathcal{L}_W(\hat{y}_i, y_i^*) = -\log(\hat{y}_{c^*,i})$
- ▶ $\frac{\partial \mathcal{L}}{\partial v_i} = \delta_i^y = \hat{y}_i - y_i^*$ et $\frac{\partial \mathcal{L}}{\partial w^y} = \delta_i^y h_i^T$
- ▶ **Pour revenir en arrière** : on calcule $\frac{\partial \mathcal{L}}{\partial u_i} = \delta_i^h$
- ▶ Si on connaît $\delta_i^h \Rightarrow \frac{\partial \mathcal{L}}{\partial w^h} = \delta_i^h x_i^T \sim$ régression logistique

Rétropropagation dans le perceptron



- Le calcul de $\frac{\partial \mathcal{L}_{CE}}{\partial u_i} = \delta_i^h \Rightarrow$ s'obtient par *chain rule* :

$$\frac{\partial \mathcal{L}_{CE}}{\partial u_i} = \frac{\partial \mathcal{L}_{CE}}{\partial v_i} \frac{\partial v_i}{\partial h_i} \frac{\partial h_i}{\partial u_i}$$

- ce qui permet d'obtenir :

$$\frac{\partial \mathcal{L}_{CE}}{\partial u_i} = \delta_i^h = \delta_i^y {}^T W^y \odot \sigma'(h_i) = \delta_i^y {}^T W^y \odot (h_i \odot (1 - h_i))$$

Rétropropagation dans les réseaux profonds

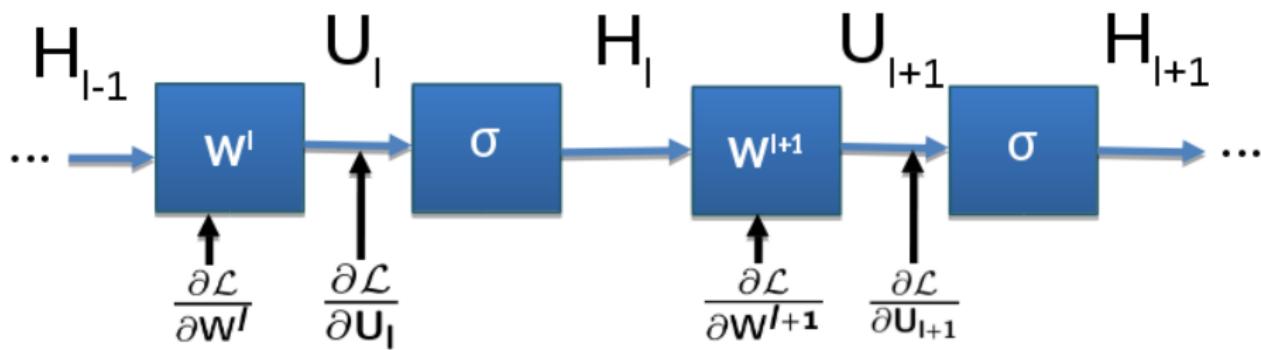
- ▶ Perceptron multi-couche : ajout d'encore plus de couches
- ▶ Rétropropagation ~ Perceptron : **en supposant que l'on connaisse**

$$\frac{\partial \mathcal{L}}{\partial \mathbf{U}_{l+1}} = \Delta^{l+1}$$

▶ $\frac{\partial \mathcal{L}}{\partial \mathbf{W}^{l+1}} = \mathbf{H}_l^T \Delta^{l+1}$

▶ Calcul de $\frac{\partial \mathcal{L}}{\partial \mathbf{U}_l} = \Delta^l$ ($= \Delta^{l+1 T} \mathbf{W}^{l+1} \odot \mathbf{H}_l \odot (1 - \mathbf{H}_l)$ pour la sigmoïde)

▶ $\frac{\partial \mathcal{L}}{\partial \mathbf{W}^l} = \mathbf{H}_{l-1}^T \Delta^h_l$



- ▶ **Apprentissage** : minimisation d'une fonction de coût \mathcal{L} sur un jeu de données \implies risque empirique
 - ▶ Jeu d'apprentissage : ensemble d'échantillons représentatif de la distribution des données **et** des étiquettes
 - ▶ Objectif : apprendre une fonction de prédiction avec une erreur faible **sur la distribution (inconnue) des données réelles** \implies risque espéré

Optimisation \neq apprentissage ! \implies sur-apprentissage \neq généralisation

Régularisation

Réduire la capacité du modèle pour réduire l'écart entre performances entraînement/test.

- ▶ Avec un modèle de suffisamment grande capacité, améliore la généralisation et donc les performances en test.
- ▶ Régularisation structurelle : ajout d'une contrainte sur les poids pour les forcer à suivre un a priori

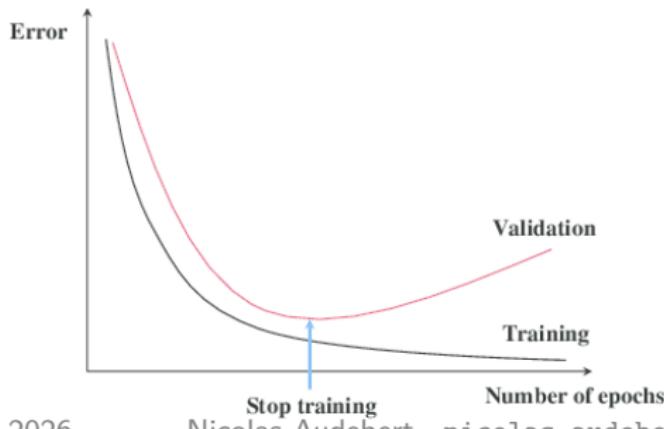
$$\mathcal{L}_r(\mathbf{w}) = \mathcal{L}(\mathbf{y}; \mathbf{y}^*; \mathbf{w}) + \alpha R(\mathbf{w})$$

- ▶ *Weight decay* : régularisation L_2 pour pénaliser les poids de norme élevée :

$$R(\mathbf{w}) = \|\mathbf{w}\|^2$$

Régularisation et hyperparamètres

- ▶ Hyperparamètres pour les réseaux de neurones :
 - ▶ Hyperparamètres d'optimisation : pas d'apprentissage (et évolution), nombre d'itérations, régularisation, etc.
 - ▶ Hyperparamètres d'architecture : nombre de couches, nombre de neurones, choix de la non-linéarité, etc.
- ▶ Le réglage des hyperparamètres permet d'ajuster la capacité du modèle et d'améliorer la généralisation :
 - ▶ L'utilisation d'un jeu de validation est cruciale !

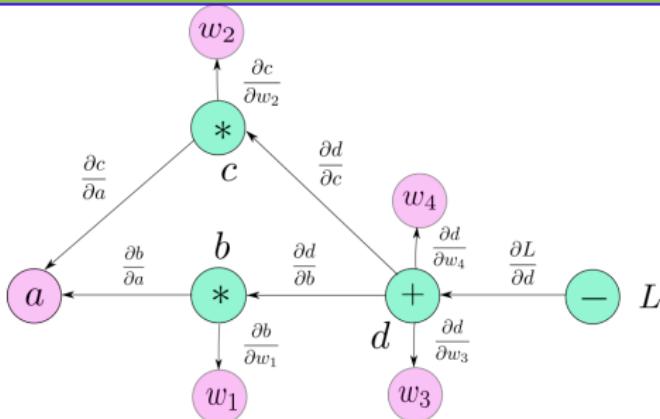


- ▶ On écrit $f_\theta(x)$ comme une suite d'opérations élémentaires (multiplications matricielles, convolutions, fonctions de transfert) *différentiables* paramétrées par θ
- ▶ Sous réserve que la fonction de coût \mathcal{L} soit différentiable par rapport aux paramètres θ , optimisation par descente de gradient :

$$\theta_{t+1} \leftarrow \theta_t + \nabla_{\theta_t} \mathcal{L}(y_i, f_{\theta_t}(x_i))$$

- ▶ Coûteux en calcul mais très grande capacité d'approximation !

Réseaux de neurones : graphe de calcul différentiable



$$b = w_1 \cdot a$$

$$c = w_2 \cdot a$$

$$d = w_3 \cdot b + w_4 \cdot c$$

$$L = 10 - d$$

Les bibliothèques d'apprentissage profond implémentent automatiquement la **rétropropagation**, c'est-à-dire le calcul des dérivées partielles de l'erreur par rapport aux paramètres du modèle grâce à la dérivation des fonctions composées :

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial w}$$

Conclusion

- ▶ Ai-je besoin de l'apprentissage automatique ?
- ▶ Mon problème se prête-t-il à l'apprentissage automatique ?
- ▶ Ai-je des données ?
- ▶ De quelle tâche s'agit-il ? Classification, régression ? Autre chose ?
- ▶ Quelle fonction de perte puis-je utiliser ? Quelle métrique d'évaluation ?
Quelles familles de modèles ?
- ▶ Entraîner les modèles
- ▶ Évaluer les modèles sur un ensemble de test séparé pour évaluer
l'erreur de généralisation, idéalement par validation croisée

Quelques références bibliographiques

- ▶ C.-A. Azencott, *Introduction au Machine Learning*, Éditions Eyrolles, 2018.
- ▶ A. Géron, *Machine Learning avec scikit-learn ; Deep Learning avec Keras*, Éditions Dunod, 2021.
- ▶ I. Goodfellow, Y. Bengio, A. Courville. *Deep Learning*. MIT Press, 2016. <http://www.deeplearningbook.org>. **goodfellow16deep**
- ▶ A. Amor, L. Estève, O. Grisel, G. Lemaître, G. Varoquaux, T. Schmitt, MOOC Machine learning with scikit-learn, France Université Numérique, <https://www.fun-mooc.fr/fr/cours/machine-learning-python-scikit-learn/>.