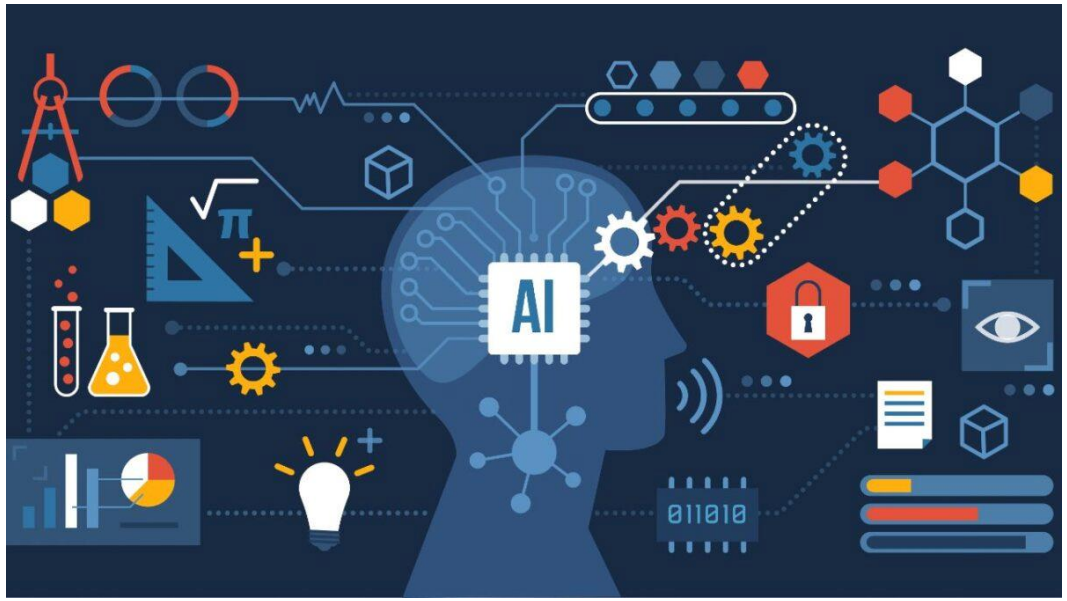
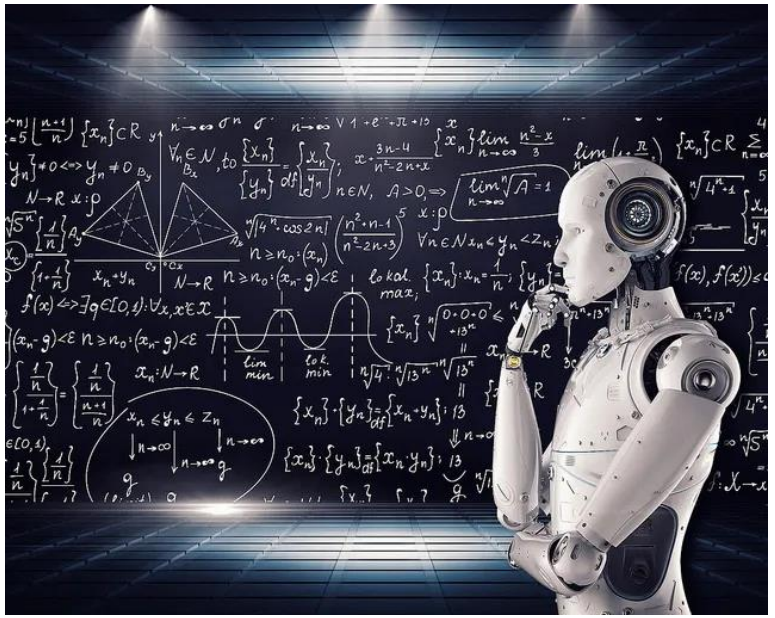
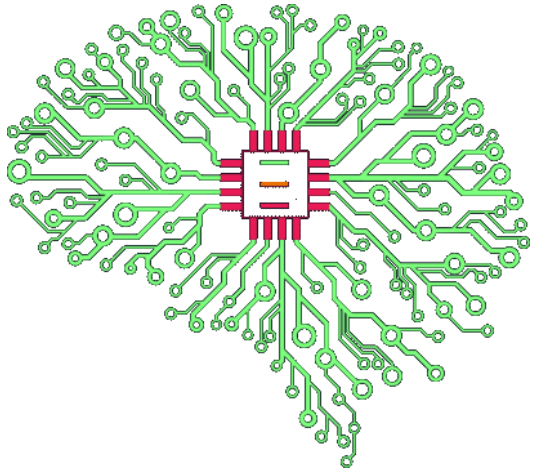
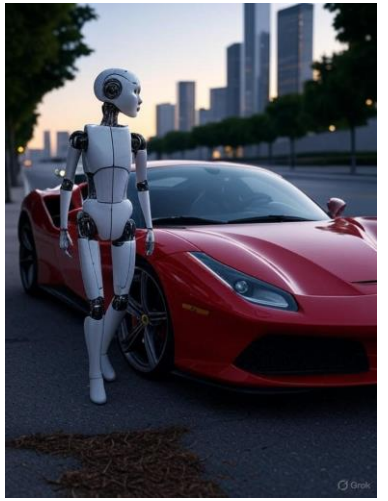
INTELLIGENCE ARTIFICIELLE









INTELLIGENCE ARTIFICIELLE

Compétences attendues :

- ✓ Analyser les principes d'intelligence artificielle. ⇆ *I*
- ✓ Choisir une démarche de résolution d’un problème d'ingénierie numérique ou d'intelligence artificielle. ⇆ ***I***
- ✓ Résoudre un problème en utilisant une solution d'intelligence artificielle. ⇆ ***I***

L'intelligence Artificielle, Qu'est-ce que c'est, à quoi ça sert ?

Définitions

Définition de l'Intelligence Artificielle

L'intelligence artificielle (IA) est « l'ensemble des théories et des techniques mises en œuvre en vue de réaliser des machines capables de simuler l'intelligence ».

En première approche, l'IA peut être assimilée à un ensemble d'algorithmes permettant de réaliser des tris, des classifications…

L'intelligence Artificielle, Qu'est-ce que c'est, à quoi ça sert ?

Définitions

Définition de l'Intelligence Artificielle

Domaines d'applications de l'IA :

- La finance
- La médecine
- Le militaire
- La robotique
- Le contrôle d'accès
- Le journalisme
- Les réseaux sociaux
- Les messageries mails
- La reconnaissance de caractères ou la traduction
- …

L'intelligence Artificielle, Qu'est-ce que c'est, à quoi ça sert ?

Définitions

Exemple de reconnaissance d'objets

L'intelligence Artificielle, Qu'est-ce que c'est, à quoi ça sert ?

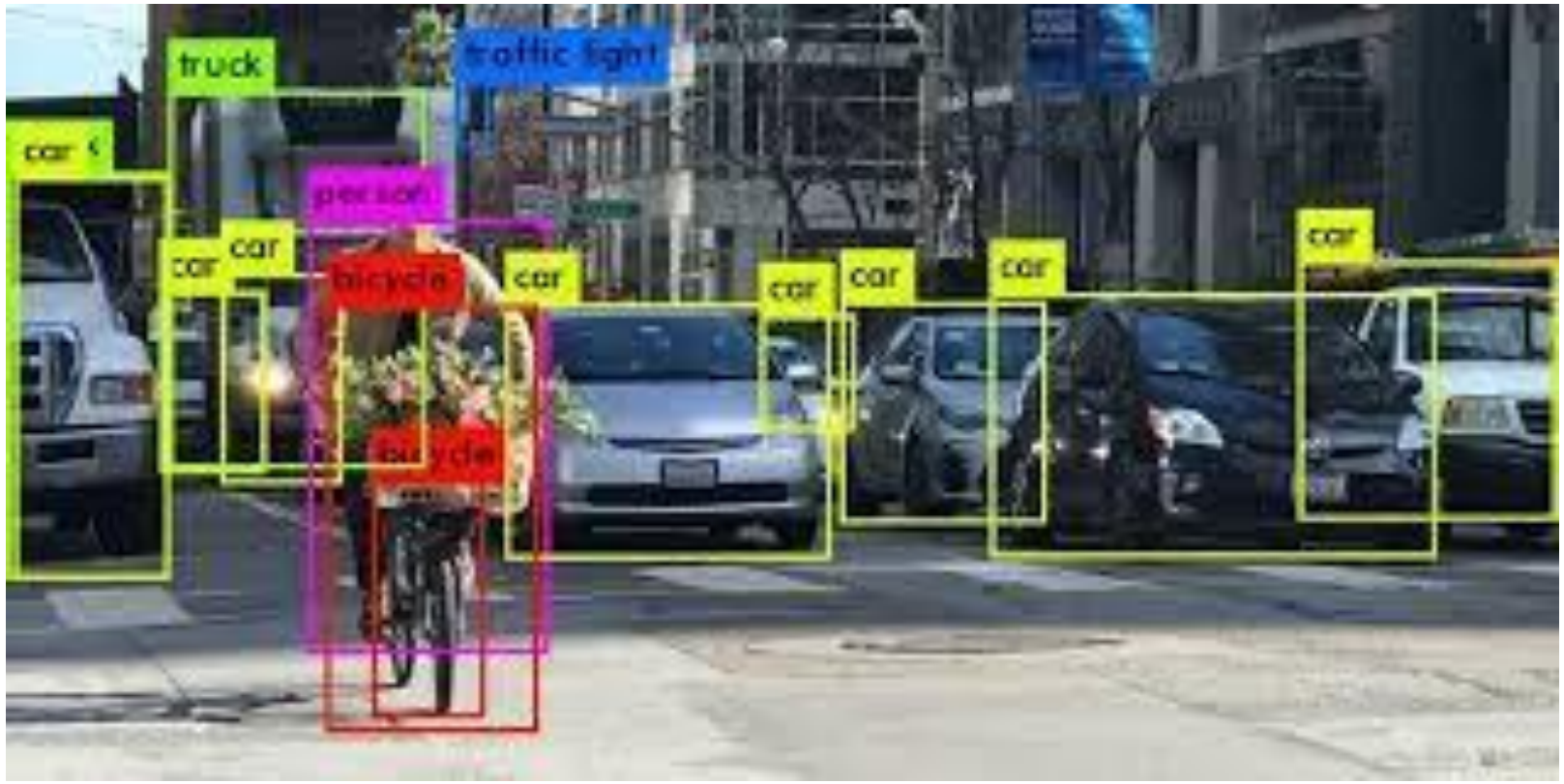
Définitions

Exemple de reconnaissance d'objets

L’intelligence Artificielle, Qu’est-ce que c’est, à quoi ça sert ?

Définitions

Exemple de reconnaissance d’objets



L’intelligence Artificielle, Qu’est-ce que c’est, à quoi ça sert ?

Définitions

Exemple de reconnaissance d’objets

L'intelligence Artificielle, Qu'est-ce que c'est, à quoi ça sert ?

Définitions

Exemple de reconnaissance d'objets

Petite histoire des « Captcha », ces tests d'identification en pleine mutation

Par Morgane Tual

Publié le 09 février 2016 à 15h47, modifié le 10 février 2016 à 15h17

Lecture 6 min.

« **200 millions de Captcha** sont déchiffrés **chaque jour**. Si l'on considère que cela prend **dix secondes** par personne, **cinq cent mille heures** sont ainsi, **chaque jour**, consacrées à cette tâche. »

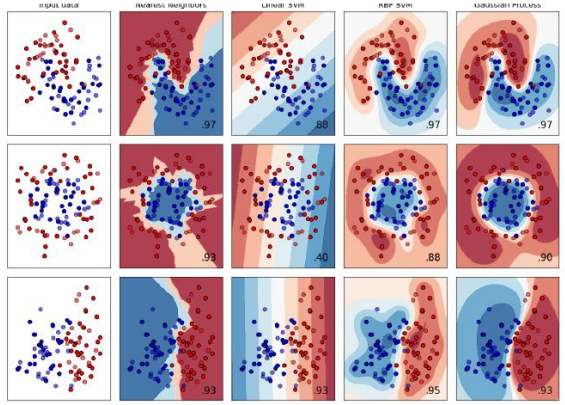
L'intelligence Artificielle, Qu'est-ce que c'est, à quoi ça sert ?

Définitions

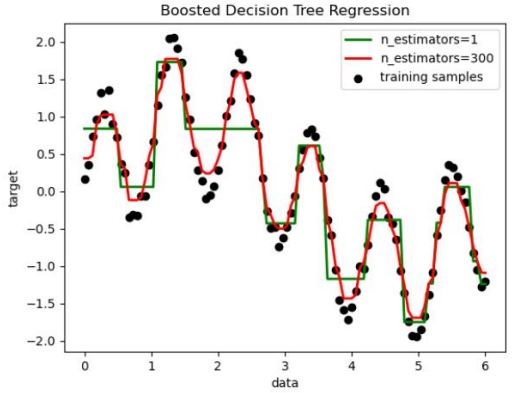
Les problèmes types de l'IA

- **La *classification***

→ Identifier à quelle catégorie un objet appartient *(apprentissage supervisé)*



(a) Classification



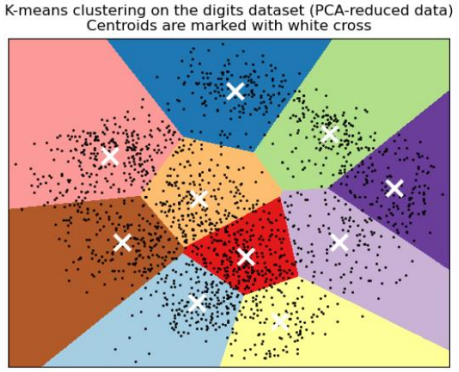
(b) Régression

- **La *régression***

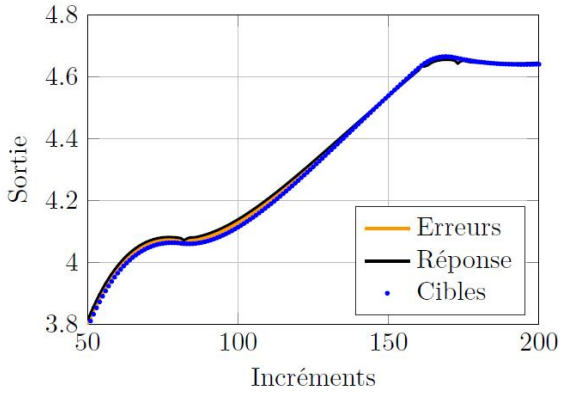
→ Prédire un paramètre à valeur continue associée à un objet.

- **La *segmentation***

→ Regrouper de manière automatique des objets dans des ensembles *(apprentissage non supervisé)*



(c) Clustering



(d) Prédiction

- **La *prédiction***

→ Prédire des données temporelles

L'intelligence Artificielle, Qu'est-ce que c'est, à quoi ça sert ?

Définitions

Définition des données

Les données → algorithme d'IA → différentes formes :

- Des données quantitatives continues (ex: température).
- Des données quantitatives discrètes (ex : nombre de pièces d'un logement).
- Des données qualitatives (ex : couleurs, des notes à un test d'opinion, ...).

Traiter ces données → il faut les organiser :

- Les données structurées (dans une base de données).
- Les données semi-structurées (dans un fichier .csv ou xml).
- Les données non structurées (image, texte ou vidéo).

L'intelligence Artificielle, Qu'est-ce que c'est, à quoi ça sert ?

Définitions

Définition des observations

On appelle observation (ou individu ou objet) une « ligne de donnée » qui va être utilisée par un algorithme d'apprentissage. Une observation est composée de caractéristiques qui varient pour chacune des observations.

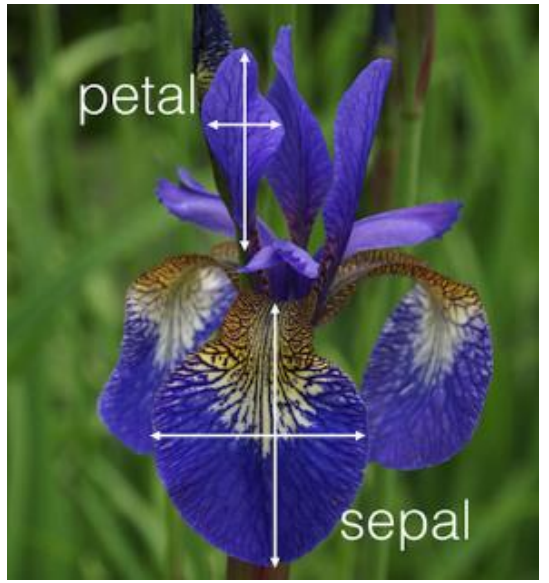
Exemple :

Base de données regroupant des photos d'Iris :
→ 150 observations → 4 caractéristiques : longueur et largeur du sépale ainsi que longueur et largeur du pétale.


Iris de l'Alaska (*Iris Setosa*)


Iris versicolore (*Iris versicolor*)


Iris de Virginie (*Iris virginica*)



L'intelligence Artificielle, Qu'est-ce que c'est, à quoi ça sert ?

Définitions

Définition de Big Data

Mégadonnées ou données massives.

Big data → ressources d'informations → caractéristiques (volume, vélocité et variété)

→ technologies et méthodes analytiques particulières pour générer de la valeur (traitements parallélisés)

Que fait-on avec ces données ?

→ Chercher un lien entre ces données.

→ *lien connu* → *apprentissage supervisé*.

→ *lien non connu* → *apprentissage non supervisé* ou *clustering*

L'intelligence Artificielle, Qu'est-ce que c'est, à quoi ça sert ?

Apprentissage Automatisé – Machine Learning

Définition de l'apprentissage automatisé ou Machine Learning

L'apprentissage automatique est un champ de l'intelligence artificielle dont l'objectif est d'analyser un grand volume de données afin de déterminer des motifs et de réaliser un modèle prédictif.

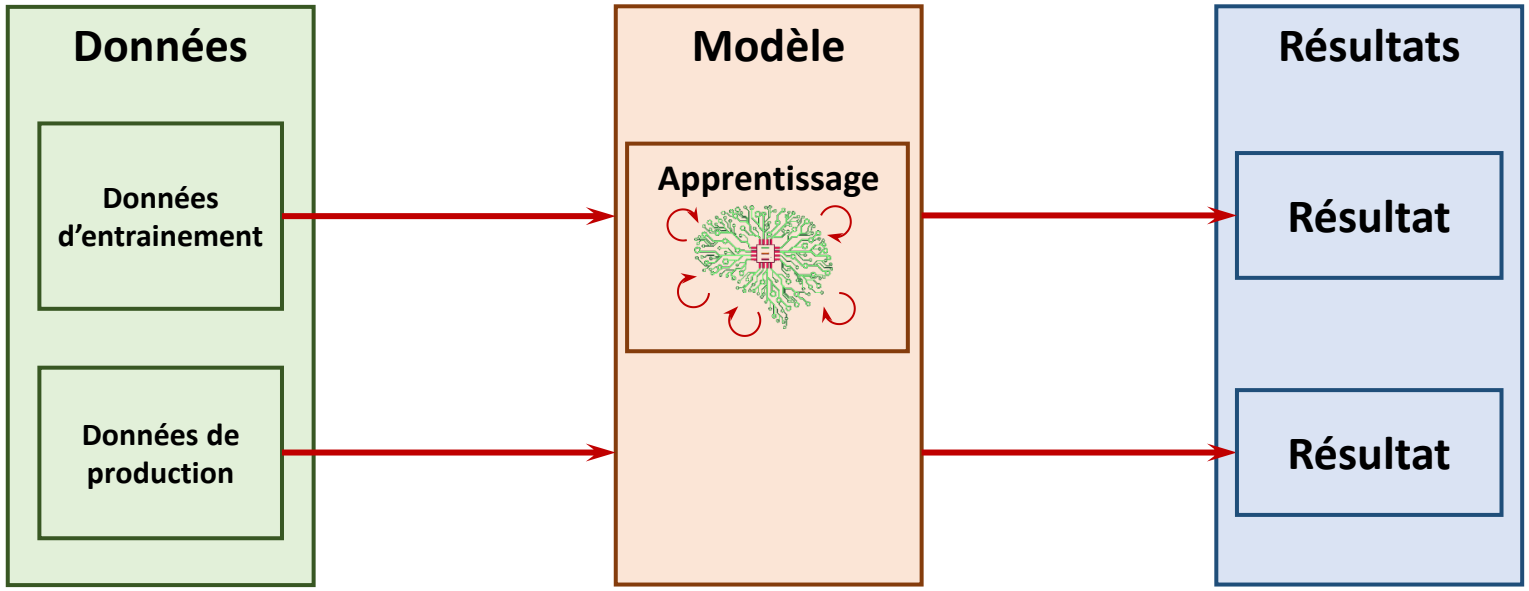
L'apprentissage → 2 phases :

- Entraînement (ou apprentissage) → estimation du modèle à partir de données d'observations.
- Mise en production du modèle (inférence) → nouvelles données → traitées → obtenir le résultat souhaité.

L'intelligence Artificielle, Qu'est-ce que c'est, à quoi ça sert ?

Apprentissage Automatisé – Machine Learning

Définition de l'apprentissage automatisé ou Machine Learning

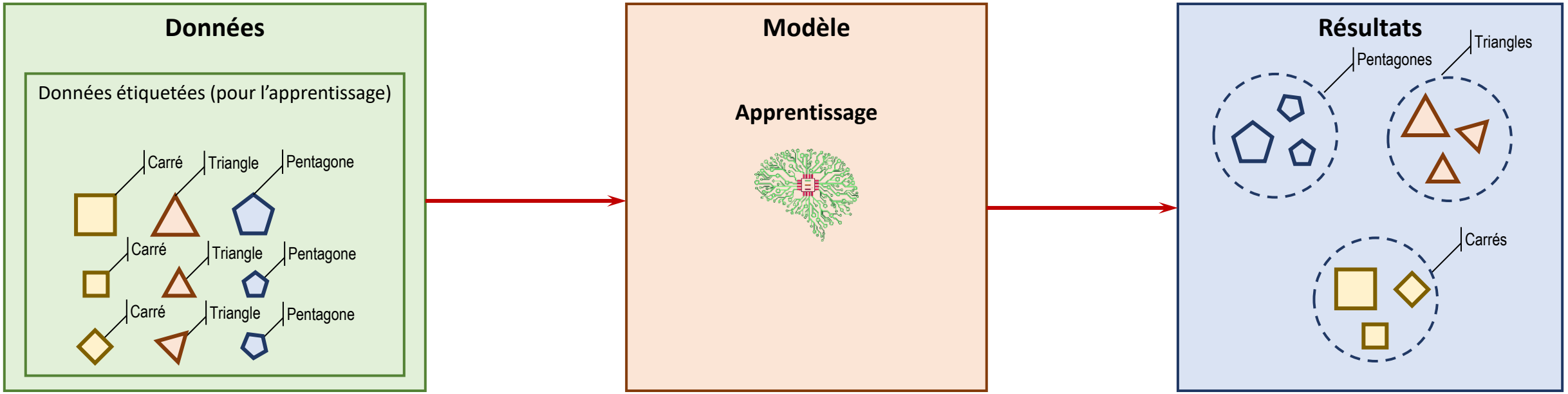


L'intelligence Artificielle, Qu'est-ce que c'est, à quoi ça sert ?

Apprentissage Automatisé – Machine Learning

Définition de l'apprentissage supervisé - Apprentissage

Tâche d'apprentissage au cours de laquelle l'algorithme (ou fonction de prédiction) va, à partir d'un ensemble de données **étiquetées**, déterminer un lien entre un ensemble les données et les étiquettes.



L'intelligence Artificielle, Qu'est-ce que c'est, à quoi ça sert ?

Apprentissage Automatisé – Machine Learning

Définition de l'apprentissage supervisé - Inférence

Tâche au cours de laquelle l'algorithme (ou fonction de prédiction) va, à partir d'un ensemble de données **non étiquetées**, prédire l'étiquette.

Exemple :

Soit un ensemble d'images en noir et blanc représentant des chiffres de 0 à 9.

L'algorithme doit dans un premier temps apprendre le lien entre l'image et le chiffre.

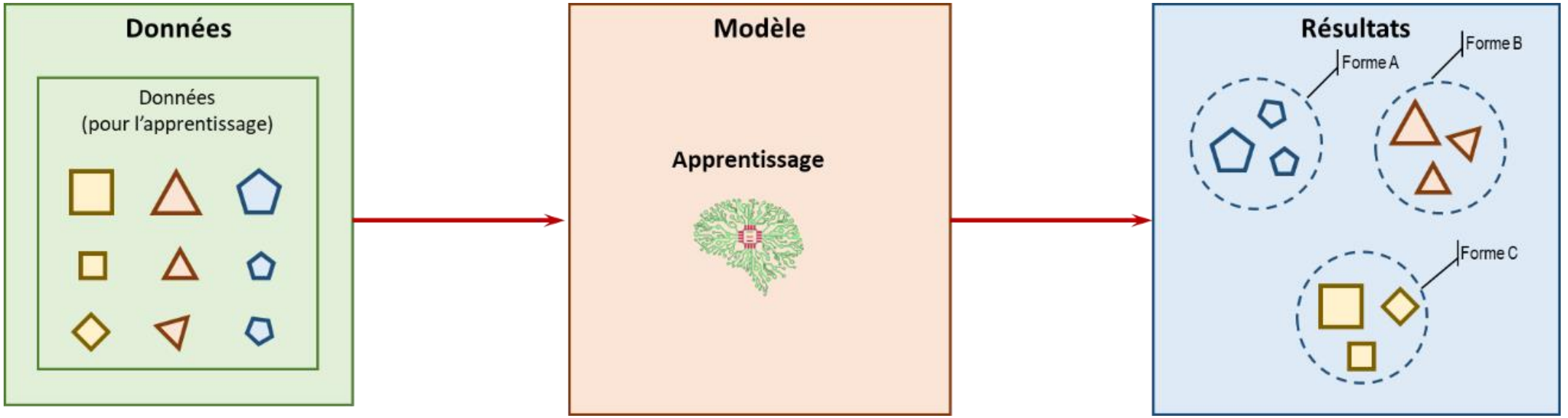
Dans un second temps, l'algorithme devra déterminer le chiffre en fonction d'une image seule.

L'intelligence Artificielle, Qu'est-ce que c'est, à quoi ça sert ?

Apprentissage Automatisé – Machine Learning

Définition de l'apprentissage non supervisé – Clustering

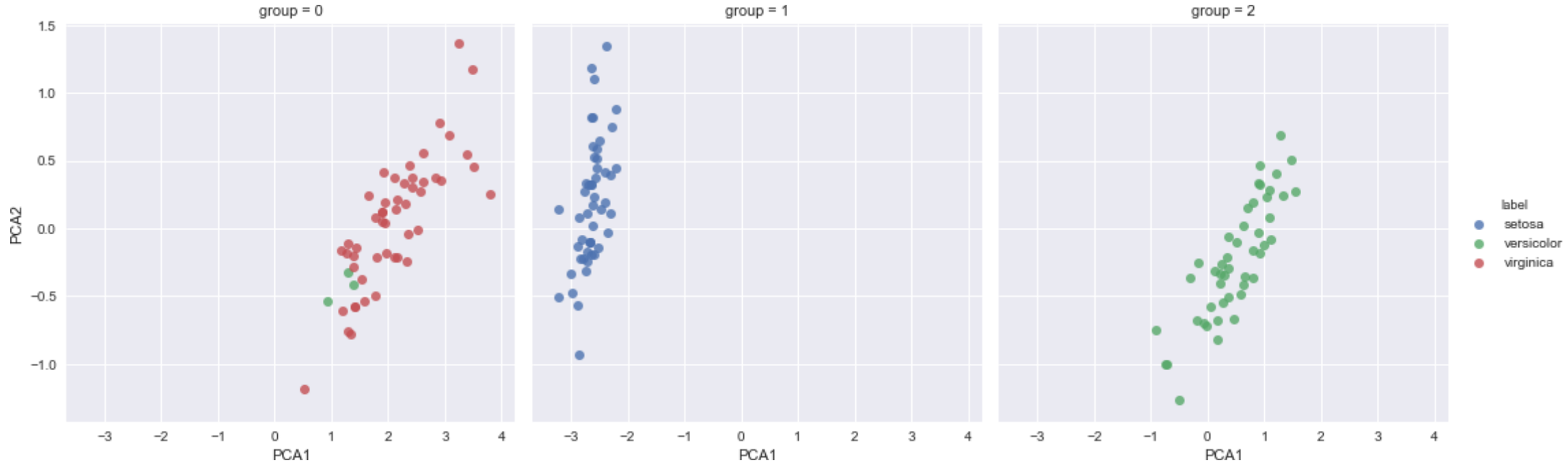
Tâche d'apprentissage au cours de laquelle l'algorithme (ou fonction de prédiction) va, à partir d'un ensemble de données **non étiquetées**, déterminer un lien entre les données (et les regrouper).



L'intelligence Artificielle, Qu'est-ce que c'est, à quoi ça sert ?

Apprentissage Automatisé – Machine Learning

Classification d'Iris en utilisant un algorithme d'apprentissage non supervisé



On remarque que sur 150 images d'Iris, uniquement avec les mesures des pétales, l'algorithme a réussi à retrouver les 3 différentes espèces, sans les connaître, à 3 erreurs près.

L’intelligence Artificielle, Qu’est-ce que c’est, à quoi ça sert ?

Apprentissage Automatisé – Machine Learning

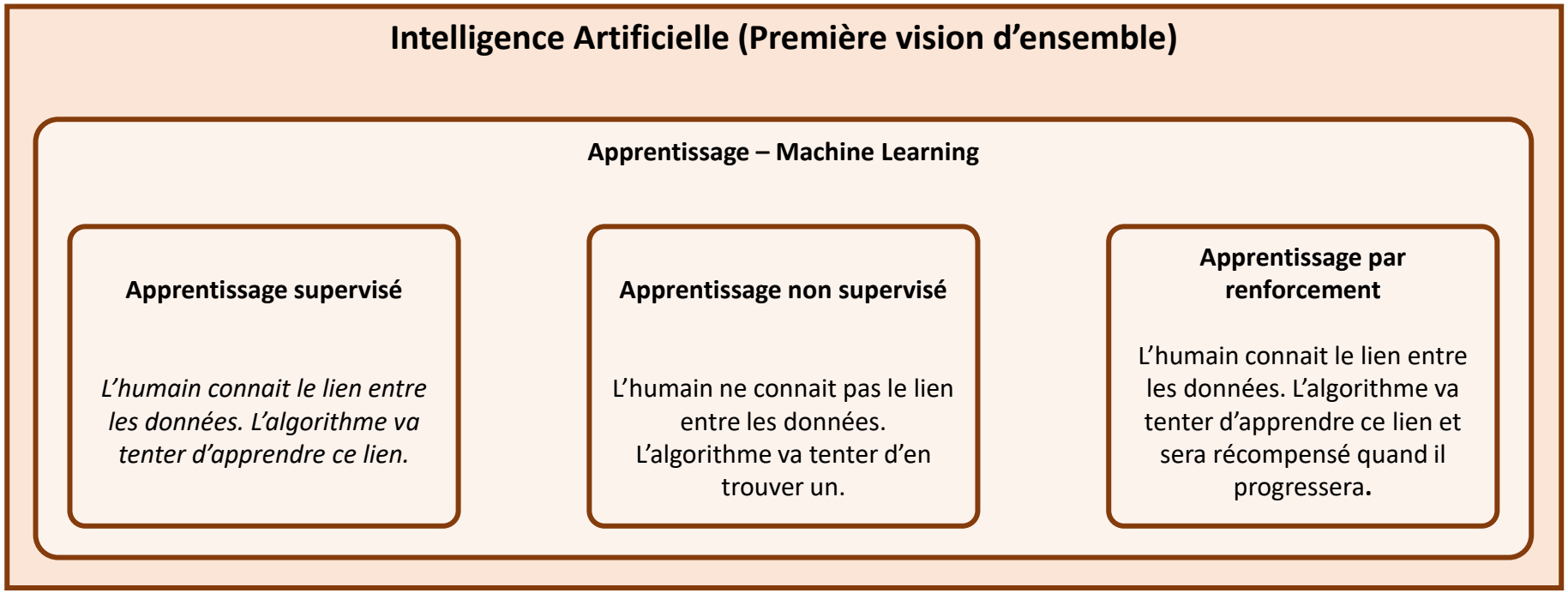
Définition de l’apprentissage par renforcement

Si au cours de l’apprentissage supervisé, un mécanisme de récompense est mis en œuvre pour améliorer les performances du modèle, on parle d’apprentissage par renforcement.

L'intelligence Artificielle, Qu'est-ce que c'est, à quoi ça sert ?

Apprentissage Automatisé – Machine Learning

Synthèse des différents types d'apprentissage

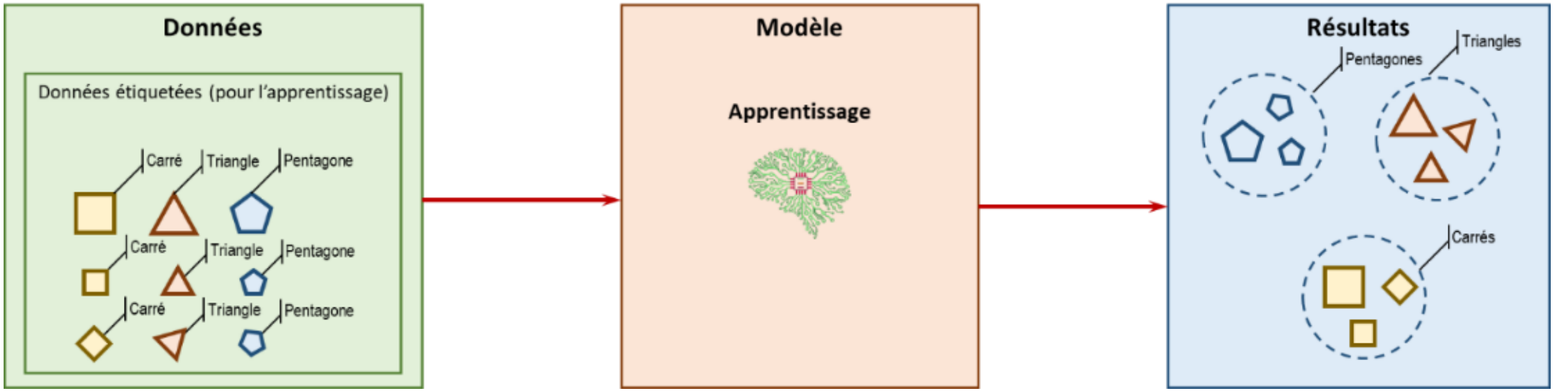


L'intelligence Artificielle, Qu'est-ce que c'est, à quoi ça sert ?

Autres définitions

Définition de la classification

En apprentissage automatique : problème de classification → problèmes → étiquettes **discrètes**



étiquettes discrètes (formes)

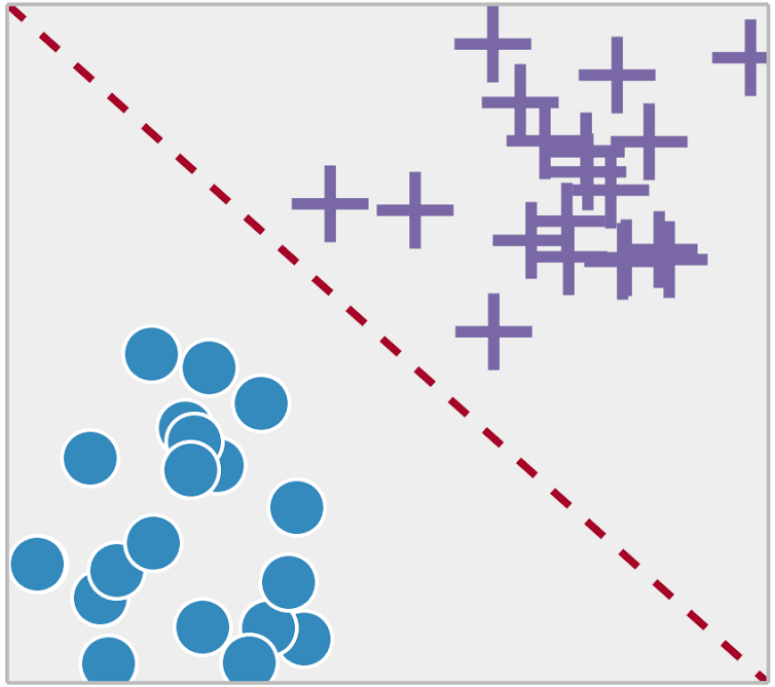
L'intelligence Artificielle, Qu'est-ce que c'est, à quoi ça sert ?

Autres définitions

Définition de la classification

Objectif de la classification : **Prédire une catégorie parmi un ensemble fini**

Remarque : Outre les réseaux de neurones, la méthode des k **plus proches voisins** est le seul algorithme de classification au programme en SII en PSI



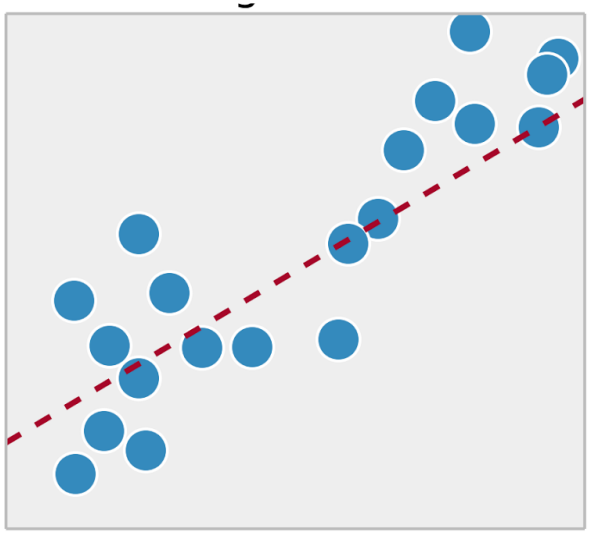
L'intelligence Artificielle, Qu'est-ce que c'est, à quoi ça sert ?

Autres définitions

Définition d'une régression

En apprentissage automatique : Régression → problèmes → étiquettes **continues**

Objectif de la régression : **Approcher une variable à partir d'autres qui lui sont liées**

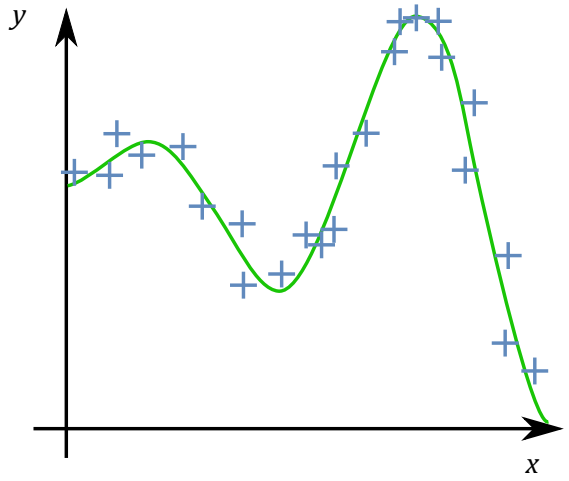


Remarque : Outre les **réseaux de neurones**, la **régression linéaire simple** et la **régression linéaire multiple** sont les seuls algorithmes de régression abordés en SII en PSI

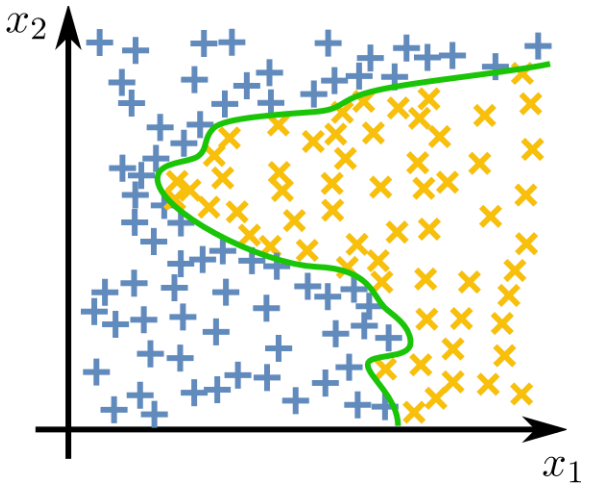
L'intelligence Artificielle, Qu'est-ce que c'est, à quoi ça sert ?

Autres définitions

Différence régression/classification



Régression



Classification

Mécanismes d'apprentissages

Classifications des algorithmes d'apprentissage

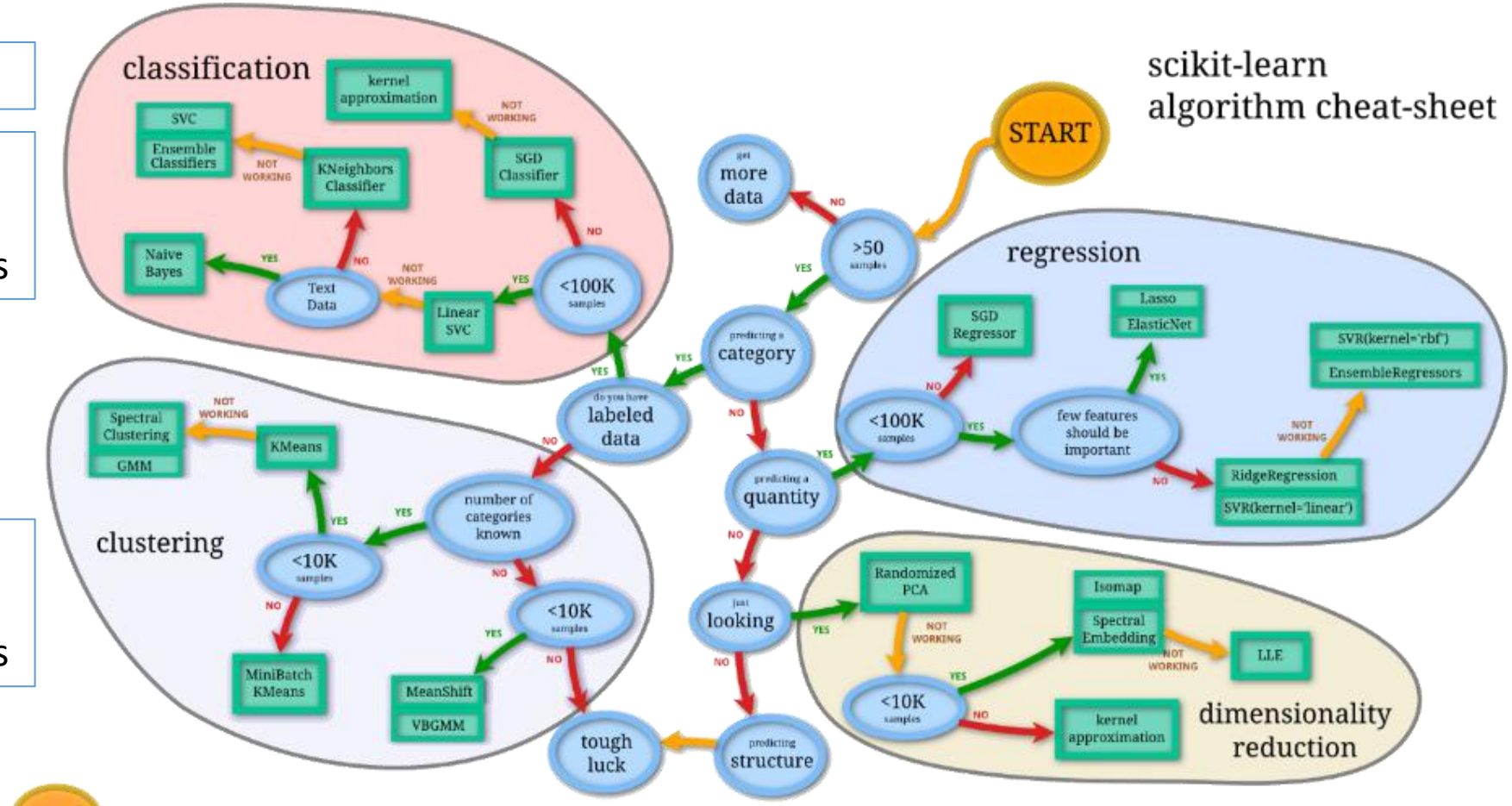
KNN

Réseau de neurones

Réseau de neurones

Réseau de neurones

Régression linéaire



Mécanismes d'apprentissages

Validation du modèle

Définition de la valeur prédictive positive

Quels critères et outils vont nous permettre de considérer que notre apprentissage est « bon » ?

Valeur prédictive positive :

$$Justesse\ (ou\ exactitude) = \frac{Nombre\ de\ prédictions\ vraies}{Nombre\ de\ prédictions\ totales}$$

Mécanismes d'apprentissages

Validation du modèle

Définition des matrices de confusion

Matrice de confusion → Déterminer la qualité d'une classification

Abscisses → valeurs prédites

Ordonnées → valeurs réelles

Pour problème de classification binaire, comportant deux catégories :

		Catégorie prédite		
		1	0	
Catégorie réelle	1	Vrais positifs (VP)	Faux négatifs (FN)	$rappel\ ou\ sensitivité = \frac{VP}{VP+FN}$
	0	Faux positifs (FP)	Vrais négatifs (VN)	$spécificité = \frac{VN}{FP+VN}$
		$précision = \frac{VP}{VP+FP}$	$précision = \frac{VN}{FN+VN}$	$exactitude = \frac{VP+VN}{VP+FN+FP+VN}$

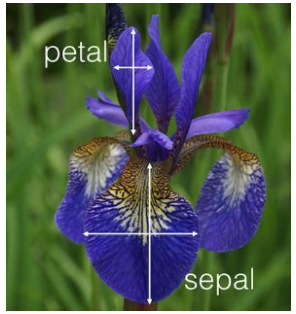
Si étiquettes ~~identiques~~ → **matrices de confusions normalisées** (on divise alors chaque terme de la matrice par la somme des éléments de la même ligne)

Python → module *sklearn.metrics* → fonction *confusion_matrix(y_true,y_pred)*

Mécanismes d'apprentissages

Validation du modèle

Définition des matrices de confusion



Exemple : Classification des Iris selon 3 familles :

Diagonale → Iris → espèce → correctement prédite

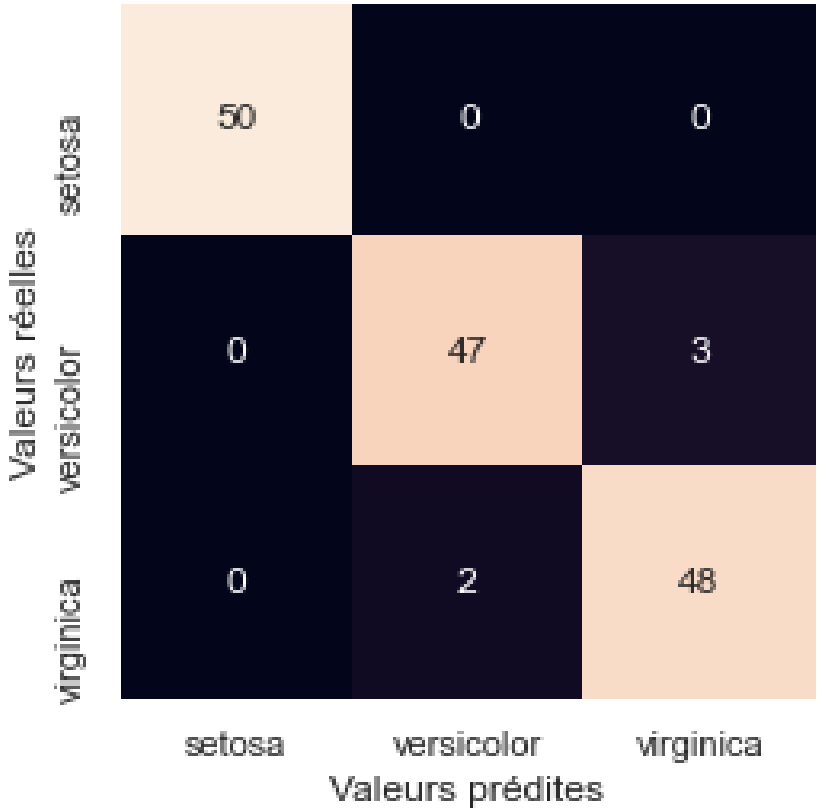
3 versicolor → classées parmi les virginica

2 virginica → classifiées dans les versicolor


Iris de l'Alaska (*Iris Setosa*)


Iris versicolore (*Iris versicolor*)


Iris de Virginie (*Iris virginica*)



Mécanismes d'apprentissages

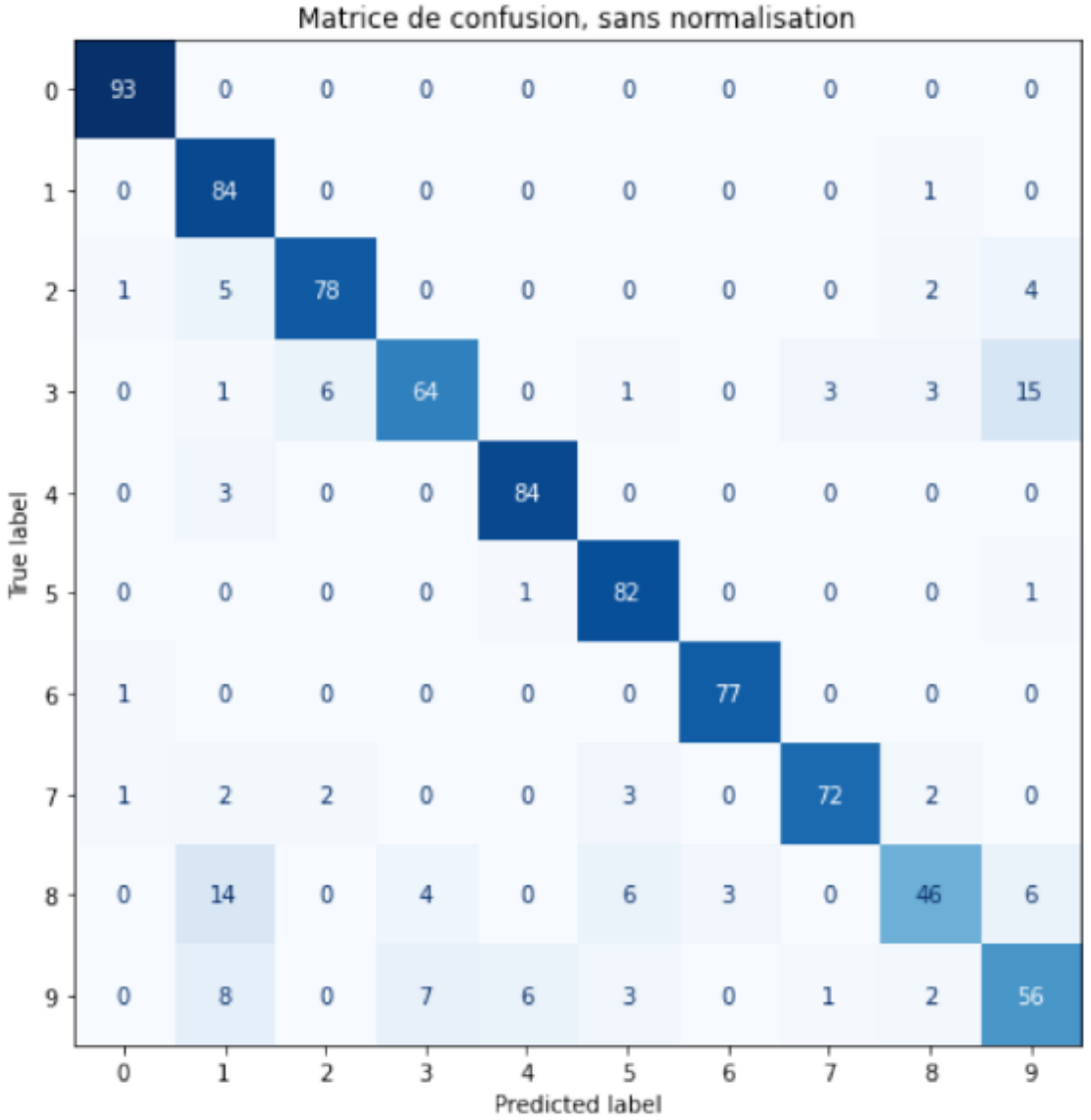
Validation du modèle

Définition des matrices de confusion

Exemple : Matrice de confusion → Reconnaissance d'un chiffre manuscrit

→ *Erreurs de prédiction → classes 3, 8 et 9*

→ *Les plus fréquentes → couples 9-3, 3-9, 9-1 et 8-1*



Mécanismes d'apprentissages

Critères de validation des problèmes de régression

Définition de l'erreur moyenne quadratique (mean squared error) :

Erreur moyenne quadratique → moyenne d'écart au carré

$$msq = \frac{1}{N} . \sum_{i=1}^{N} \|Yi - \hat{Y}i\|^2$$

Yi → valeur réelle (étiquetée)

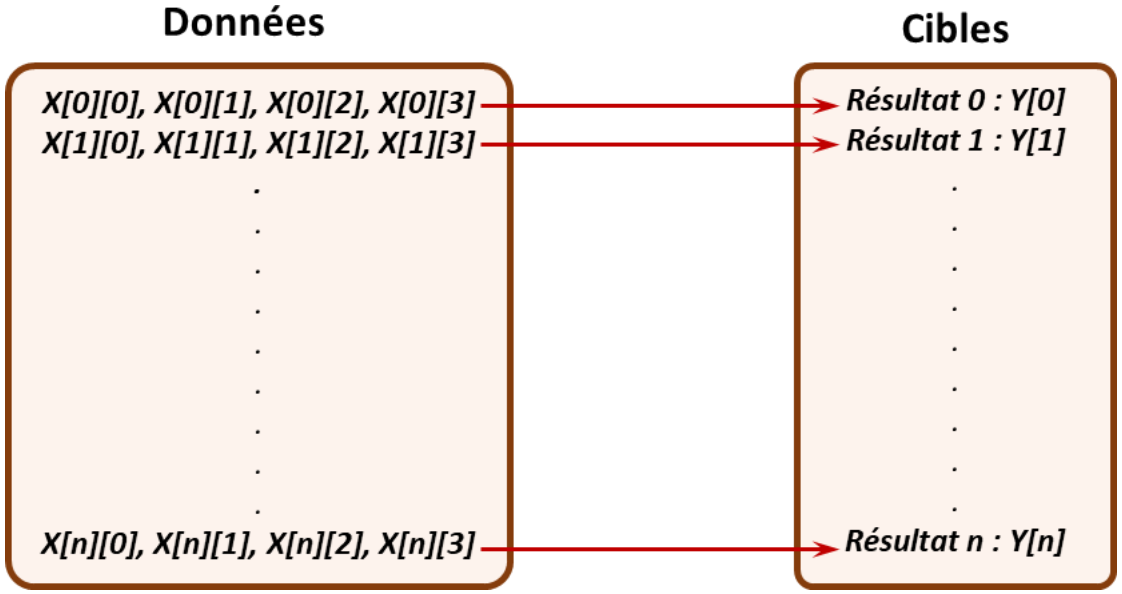
$\hat{Y}i$ → valeur estimée par l'algorithme

Mécanismes d'apprentissages

Séparation des données / Gestion des données ?

Types de données – Apprentissage supervisé

En apprentissage supervisé → nécessaire de connaître → données d'entrées + les sorties correspondantes (cibles)



Remarque :

- Si les sorties y_i sont des **scalaires réels** → problème de **régression simple (ou monovariable).**
- Si les sorties y_i sont des **vecteurs de réels** → problème de **régression multiple (ou multivariable).**

Mécanismes d'apprentissages

Séparation des données / Gestion des données ?

Types de données – Apprentissage non supervisé

L'apprentissage non-supervisé → absence de sortie y fournie lors de l'apprentissage du modèle prédictif.

Exemple : Prédire la partie manquante d'une donnée : prédire les données futures pour une série temporelle.

Mécanismes d'apprentissages

Séparation des données / Gestion des données ?

Séparation des données

Démarrage d'un apprentissage supervisé → séparer les données (entrées + cibles) en deux parties :

- Les données d'entraînement → Entraîner le modèle (*entre 60 et 80% des données de base*).
- Les données de test → Valider l'apprentissage (ou le modèle) (*entre 20 et 40% des données de base*).

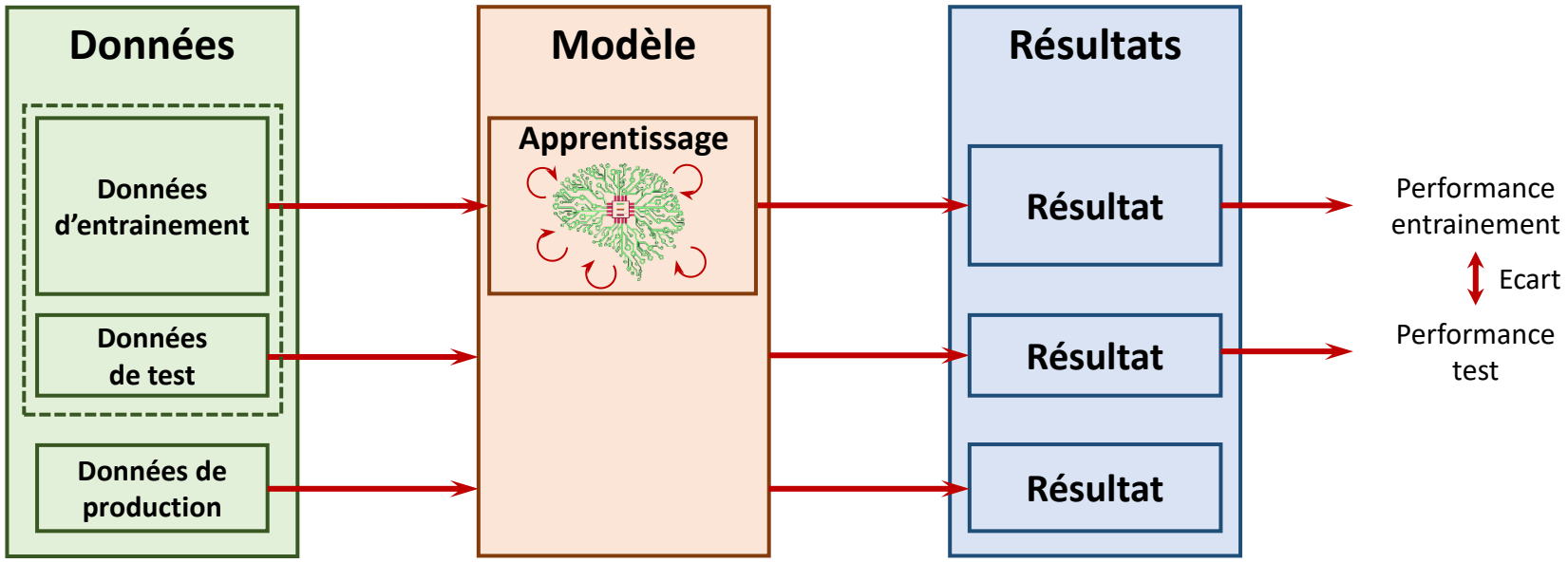
→ Déduire une performance du modèle sur le traitement des données d'entraînement.

→ Déduire une performance du modèle sur le traitement des données de test.

Mécanismes d'apprentissages

Séparation des données / Gestion des données ?

Séparation des données



Algorithmes d'apprentissage

Algorithme des k plus proches voisins (KNN)

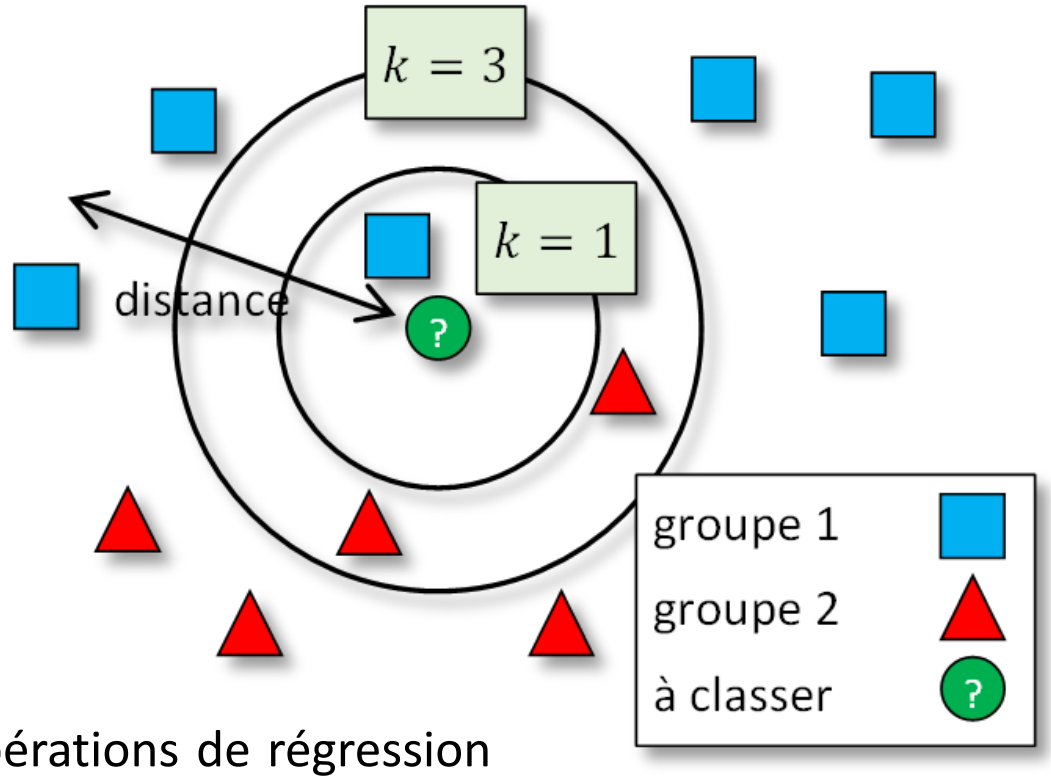
Présentation

Méthode → Estimer la classe d'une nouvelle donnée

→ norme entre cette nouvelle donnée et l'ensemble des données du jeu d'entraînement

→ On cherche les *k* plus proches voisins → classe majoritaire

→ Attribution de cette classe à la nouvelle donnée



Algorithme des *k* plus proches voisins (*kNN*) → Réaliser des opérations de régression et de classification sur un ensemble de données

Algorithmes d'apprentissage

Algorithme des k plus proches voisin

Présentation

Exemple :

Candidats 1 et 2 → deux Iris dont on connaît les caractéristiques mais pas l'espèce.

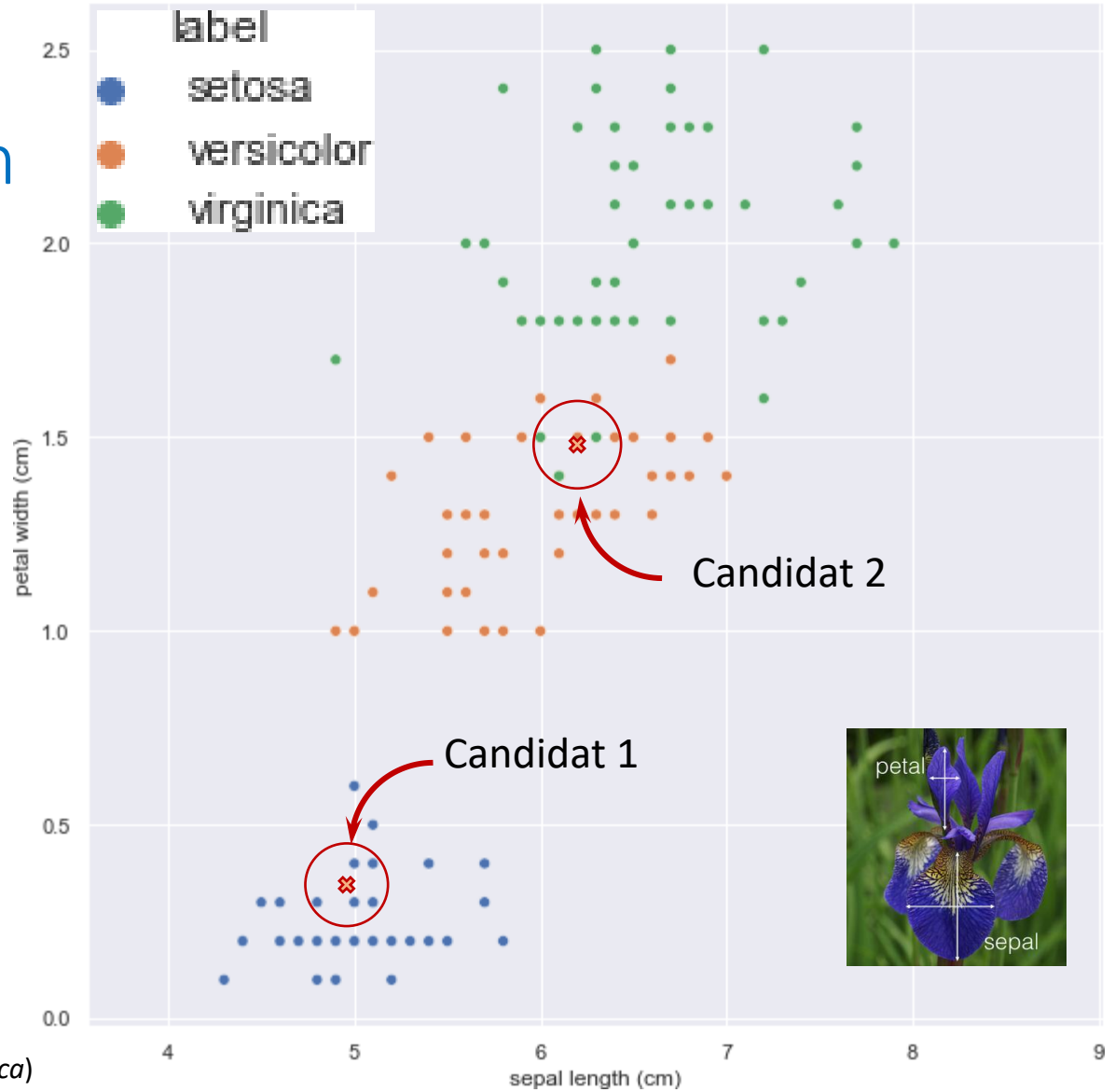
Candidat 1 → 5 plus proches voisins → setosa
→ Probable → setosa

Candidat 2 → 5 plus proches voisins → 3 virginica + 2 versicolor
→ Probable → virginica

Iris de l'Alaska (*Iris Setosa*)

Iris versicolore (*Iris versicolor*)

Iris de Virginie (*Iris virginica*)



Algorithmes d'apprentissage

Algorithme des k plus proches voisins (KNN)

L'algorithme

Algorithme $\boldsymbol{k}$ **plus proches voisins** :

- **d'apprentissage supervisé** (avec jeu de données d'apprentissage)
- **non paramétrique**
- **classification**

Algorithmes d'apprentissage

Algorithme des k plus proches voisins (KNN)

L'algorithme

Première étape : Calculer la distance :

Distance euclidienne : Pour deux vecteurs x_1 et x_2 de taille *N*, $\boldsymbol{d = \sqrt{\sum_{i=1}^{N}(x_{1,i} - x_{2,i})^2}}$

```
import math as m
def distance(x1:list, x2:list) -> float :
    distance = 0.
    for i in range(len(x1)):
        distance += (x1[i]-x2[i])**2
    return m.sqrt(distance)
```

Algorithmes d'apprentissage

Algorithme des k plus proches voisins (KNN)

L'algorithme

Deuxième étape : Détermination des plus proches voisins :

Données pour l'entraînement : *data_train*

Donnée à tester : *data_test*

→ Calculer la distance entre la donnée à tester et les données d'entraînement

→ Trier les données

→ Retourner les *k* lignes les plus proches de *data_test*

```
def get_voisins(data_train:list, data_test, k:int) -> list :
    distances =[]
    for ligne in data_train :
        d = distance(ligne,data_test)
        distances.append([ligne,d])
    # Tri de la liste suivant la seconde colonne (distances)
    distances.sort(key=lambda t : t[1])
    voisins = []
    for i in range(k) :
        voisins.append(distance[i][0])
    return voisins
```

Algorithmes d'apprentissage

Algorithme des k plus proches voisins (KNN)

L'algorithme

Troisième étape : Prédiction :

→ Regarder quelle est la classe majoritaire parmi ces voisins

```
def prediction(data_train:list, data_test, k:int) -> list :
    voisins = get_voisins(data_train, data_test, k)
    sorties = [ligne[-1] for ligne in voisins]
    predict = max(set(sorties), key=sorties.count)
    return predict
```

Remarque : Python → bibliothèque *sklearn* → module *neighbors*

Algorithmes d'apprentissage

Algorithme des k plus proches voisins (KNN)

Applications et limites

Algorithme → robuste, si suffisamment d'exemples sont fournis dans les données d'apprentissage

Choix → **hyperparamètre** k → forte influence sur la prédiction

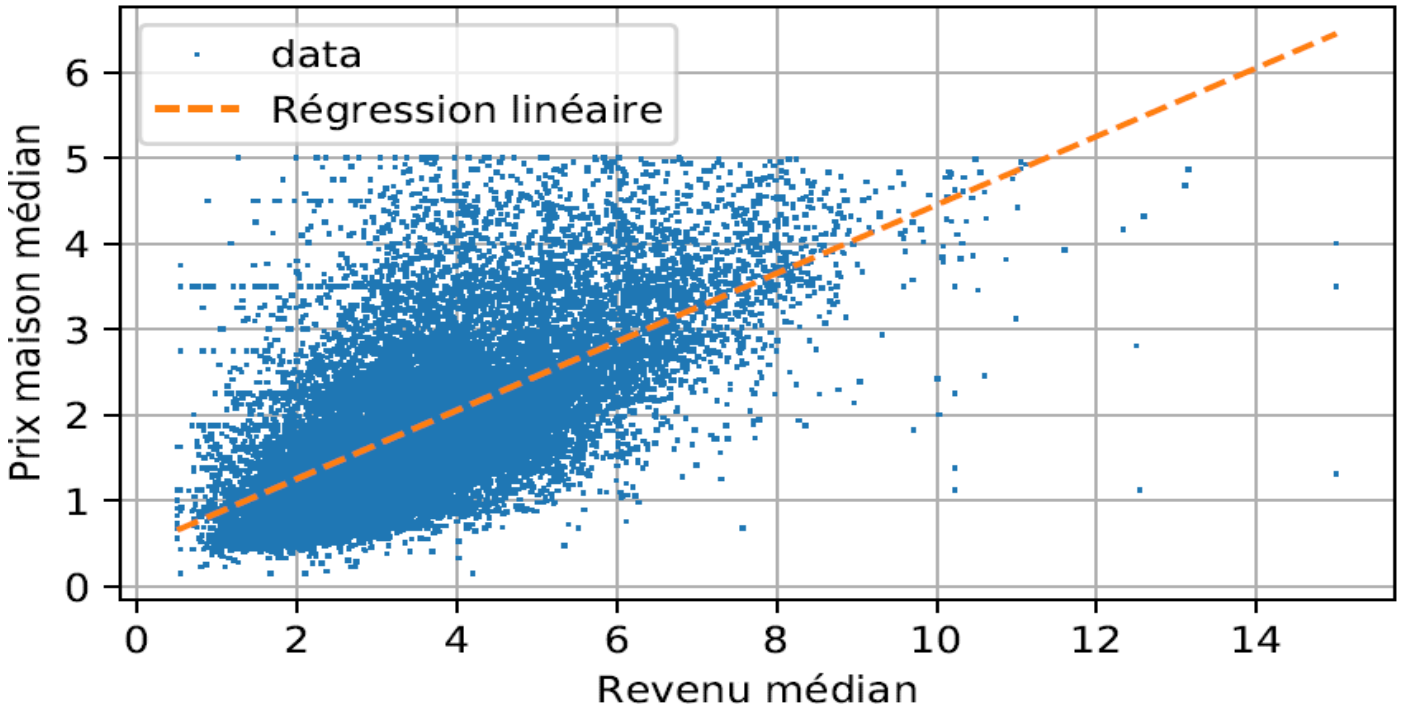
- si k est trop petit → éléments sortant de l'ordinaire → influer la prédiction + prob généralisation du modèle pour de nouveaux éléments → moins bonne
  → **sur-apprentissage**
- si k est trop grand → catégorie majoritaire dans le jeu de données → prédite trop souvent
  → **sous-apprentissage**

Algorithmes d'apprentissage

Régressions

Régression mono variable – Régression linéaire simple

Régression linéaire → algorithme d'apprentissage supervisé



Algorithmes d'apprentissage

Régressions

Régression mono variable – Régression linéaire simple

Régression linéaire → choisir des fonctions de prédiction f :

$$\forall x \in \mathbb{R} \; f(x) = a.x \qquad a \text{ = constante réelle}$$

Fonction de prédiction optimale → minimise l'erreur sur la base de données d'entraînement :

$$\hat{f} = \min_f \frac{1}{n} \sum_{i=1}^{n} (f(x_i) - y_i)^2$$

Algorithmes d'apprentissage

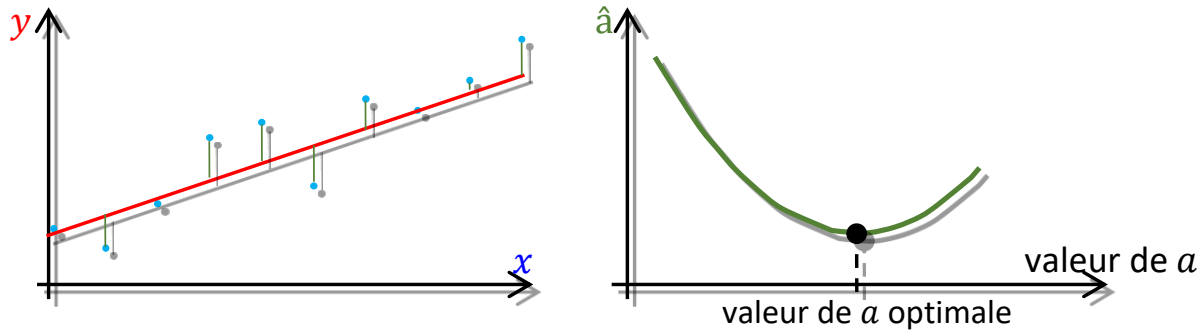
Régressions

Régression mono variable – Régression linéaire simple

Hypothèse d'une solution → fonction linéaire : $\forall x \in \mathbb{R}\ f(x) = a.x$ constante $\hat{a}$ → résolution du problème de minimisation :

$$\hat{a} = \min_{a} \frac{1}{n} \sum_{i=1}^{n} (a.x_i - y_i)^2$$

$\hat{a}$ = **coût → minimiser la somme des écarts au carré entre les prédictions et les valeurs attendues**



<u>Remarque :</u> Python → fonction *linregress()* du module *scipy.stats* + module *linear_model* de la bibliothèque *sklearn*

Algorithmes d'apprentissage

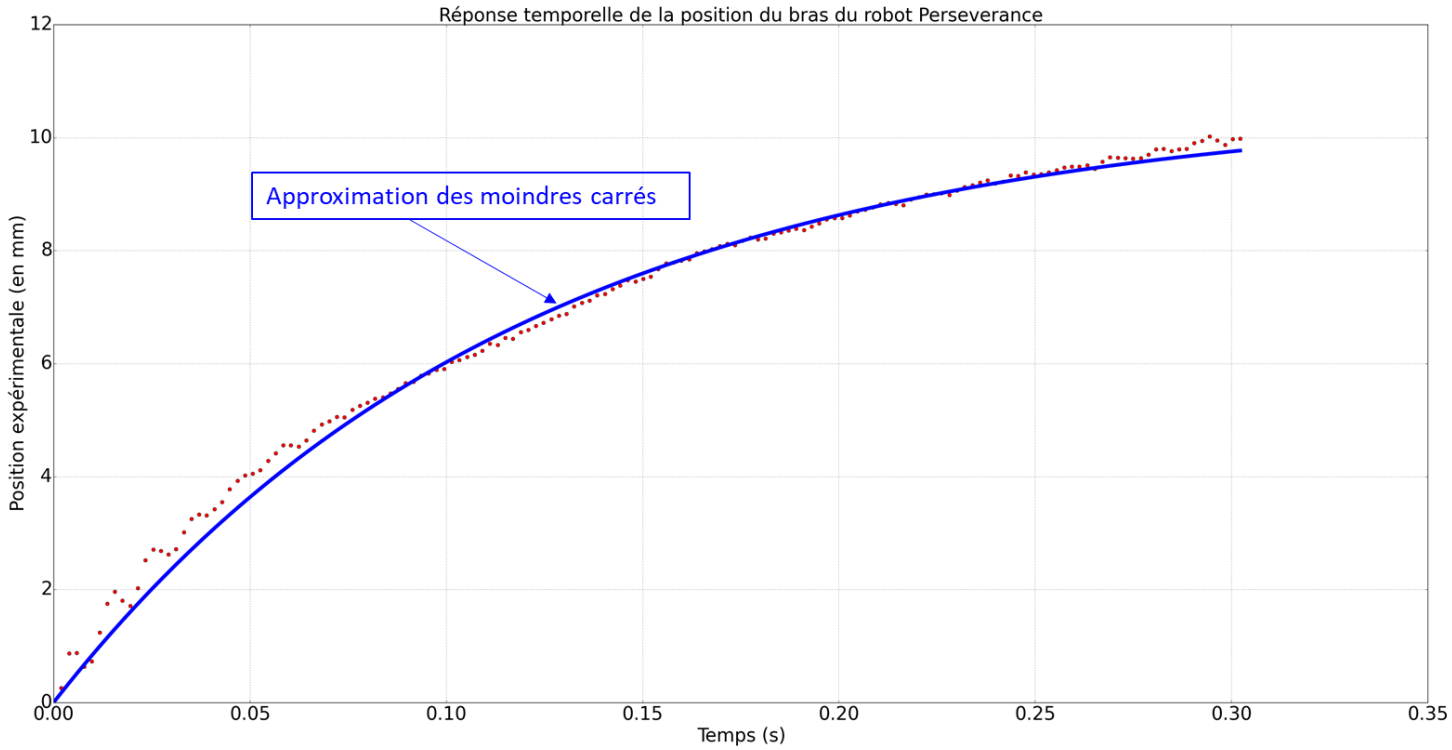
Régressions

Régression mono variable – Régression linéaire simple

Remarque :

TD 4 AUTOMATIQUE (PCSI) : Robot Perseverance

→ Régression linéaire aux moindres carrés pour retrouver les paramètres caractéristiques du système du premier ordre.

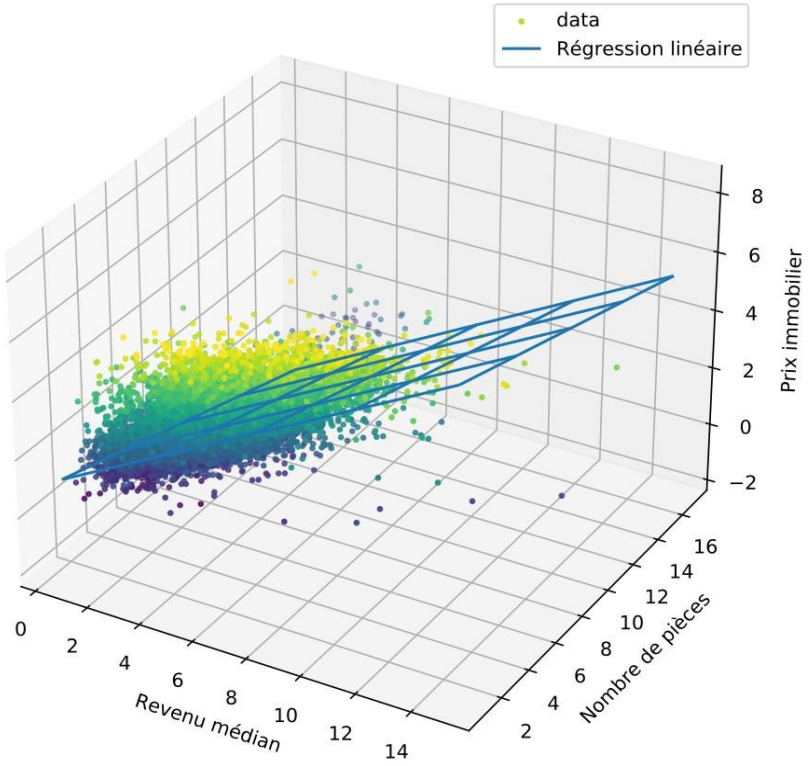


Algorithmes d'apprentissage

Régressions

Régression multiple ou multi-variables

Régression linéaire multiple, ou multivariable → généralisation de la méthode précédente (dimension >1)



Algorithmes d'apprentissage

Régressions

Régression multiple ou multi-variables

Entrées x_i → vecteurs de $\mathbb{R}^d$ Sorties y_i → vecteurs de $\mathbb{R}^p$

Modèle linéaire liant → entrées x et sorties y → existence d'un vecteur de paramètres $A \in \mathbb{R}^{d.p}$

$$y = x^T Ay = x^T A = f(\mathrm{A})$$

→ Minimiser l'erreur de prédiction quadratique sur la base de données d'entraînement

→ Calculer la matrice $\hat{A}$ → modèle reproduisant les observations le plus fidèlement :

$$\hat{A} = \min_{A \in \mathbb{R}^{d.p}} \sum_{i=1}^{n} \frac{1}{n} \left\| y_i - x_i^T A \right\|^2$$

où $\min_{A \in \mathbb{R}^{d.p}} f(A)$ = valeur prise par A lorsque $f(A)$ atteint son minimum

Algorithmes d'apprentissage

Régression multi-variables

Soient $Y = \begin{pmatrix} y_1^T \\ y_2^T \\ \cdots \\ y_n^T \end{pmatrix} \in \mathbb{R}^{n.p}$ et X$= \begin{pmatrix} x_1^T \\ x_2^T \\ \cdots \\ x_n^T \end{pmatrix} \in \mathbb{R}^{n.d}$

$$\hat{A} = \min_{A \in \mathbb{R}^{d.p}} \frac{1}{n} \|Y - XA\|^2$$

Solution du problème : $\hat{A} = \left(X^T X\right)^{-1} X^T Y$

<u>Remarque :</u> Python → Tableaux du module *numpy*, *numpy.array* → calcul matriciel

→ Module *linear_model* de la bibliothèque *sklearn* → régression linéaire multiple

Algorithmes d'apprentissage

Applications et limites

Modèles de régression linéaire ou multivariables → facilement et rapidement implémentables

<u>Limites :</u>

- Domaine d'applications restreint → linéarité du modèle
  Régressions polynomiales → élargie le champ d'application + complexifie le modèle
  → problèmes de **sur-apprentissage** et **sous-apprentissage**

- Grand nombre de données (phase d'apprentissage)

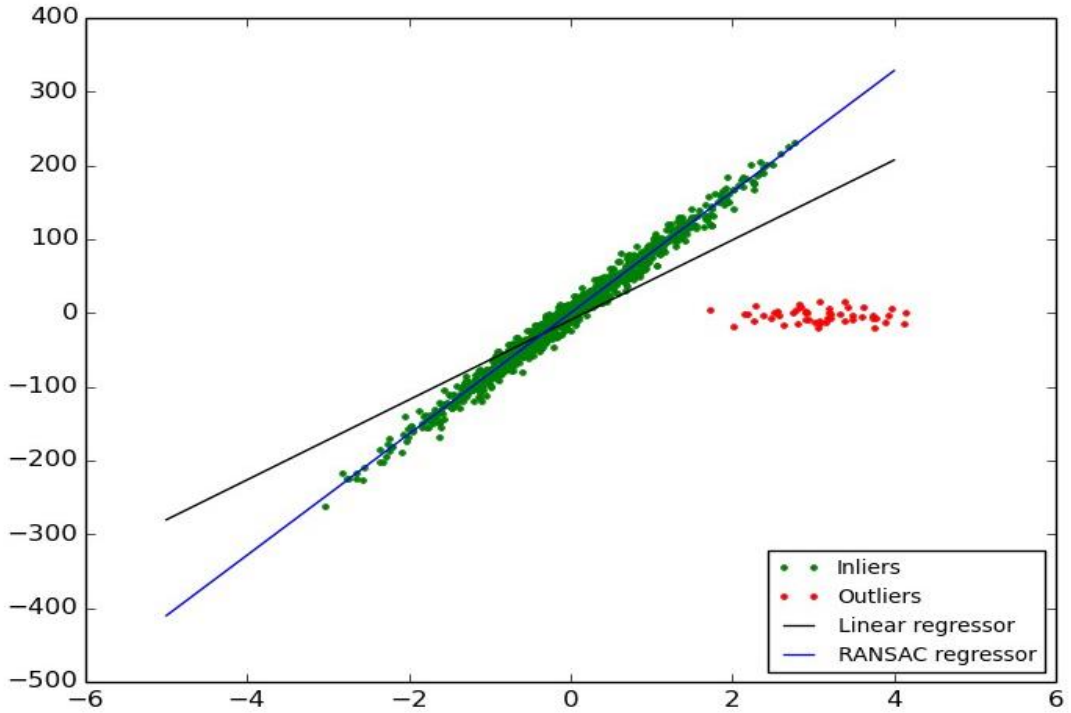
- Estimation sensible au bruit de mesure

Algorithmes d'apprentissage

Applications et limites

Limites :

- Outil d'analyse très sensible à la présence de données aberrantes (*outliers*)



RANSAC = RANdom SAmple Consensus
= Méthode d'estimation mathématique et itérative
→ valeurs aberrantes (outliers).

Algorithmes d'apprentissage

Limites de l'apprentissage automatique

- **Probabilité d'erreur non nulle** (précision des prédictions ~~100 %~~)
- **Algorithmes opaques**
- **Erreurs difficiles à détecter et corriger** (opacité des algorithmes)
- Algorithmes de **complexité exponentielle**
- **Forte dépendance entre les performances du modèle prédictif et les données d'apprentissage**
- « Fléau de la dimension » : nombre de données d'apprentissage nécessaire au bon fonctionnement du modèle prédictif croît fortement avec la dimension du problème
- Forts impacts environnementaux (consommation énergétique ou empreinte carbone)

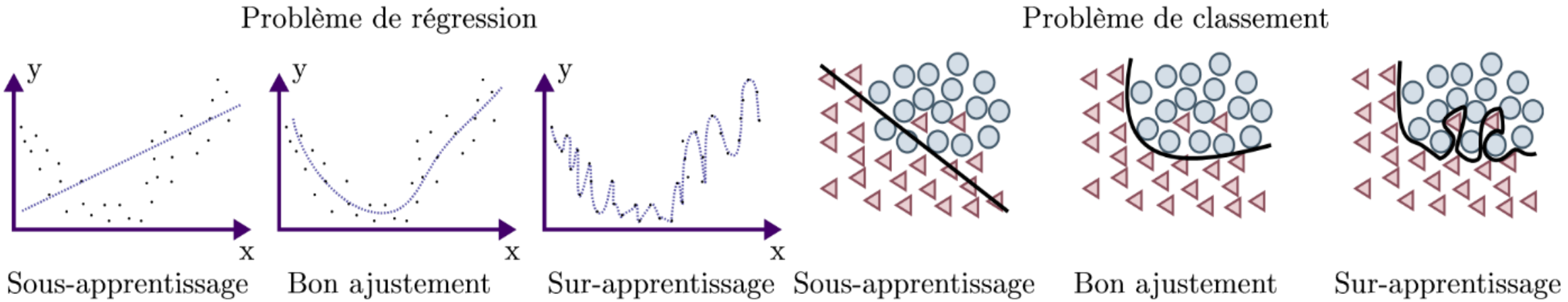
Algorithmes d'apprentissage

Le danger du surapprentissage – Dilemme biais-variance

Risque de surapprentissage (*overfitting*) = modèle colle trop aux données d'apprentissage

→ Surévalue la confiance que l'on apporte aux données (~~considère~~ incertitudes et bruits de mesures)

→Dilemme (ou compromis) biais-variance



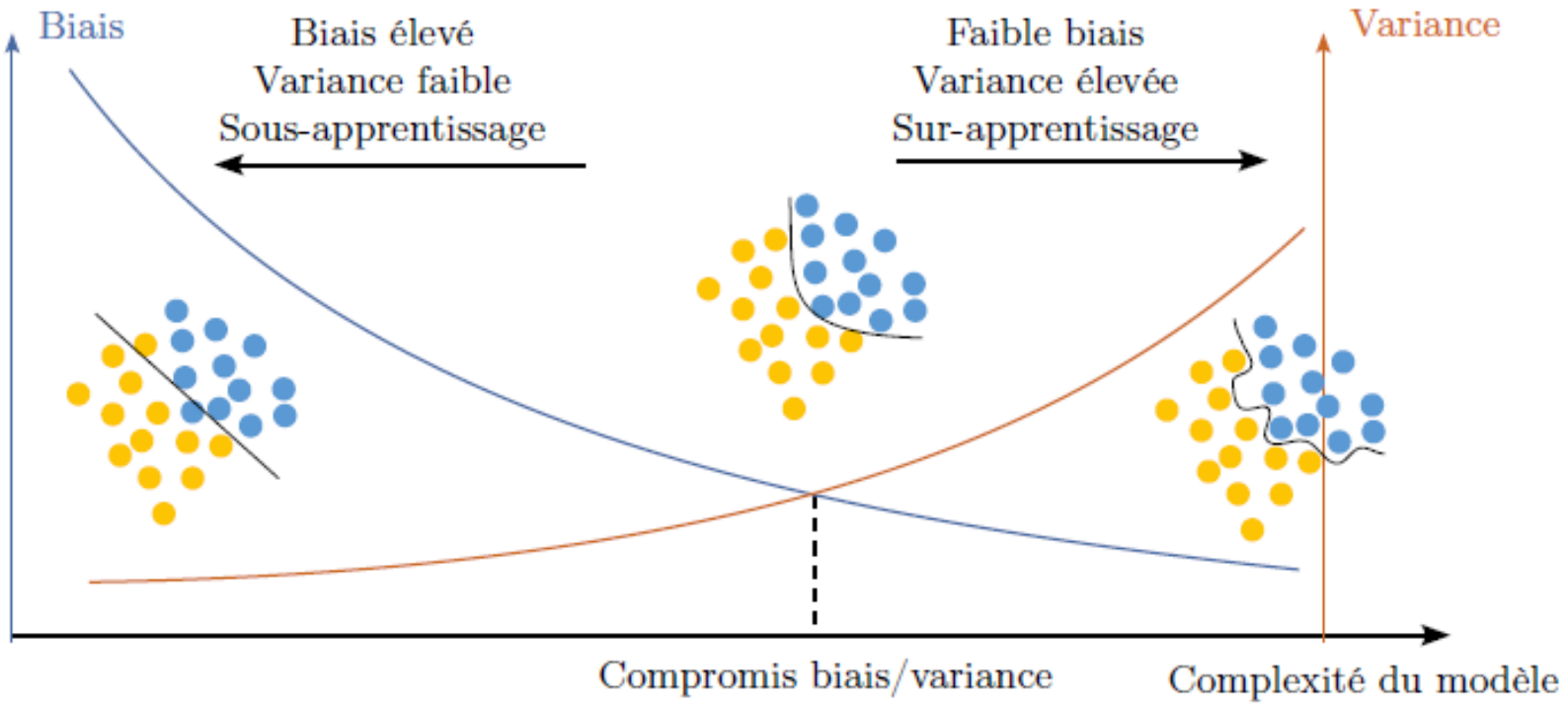
Algorithmes d'apprentissage

Le danger du surapprentissage – Dilemme biais-variance

Plus un modèle est complexe, plus il va diminuer le biais, mais plus il va augmenter la variance

Biais = erreur provenant
d'hypothèses erronées
dans l'algorithme d'apprentissage

Variance = erreur due à la sensibilité
aux petites fluctuations
de l'échantillon d'apprentissage

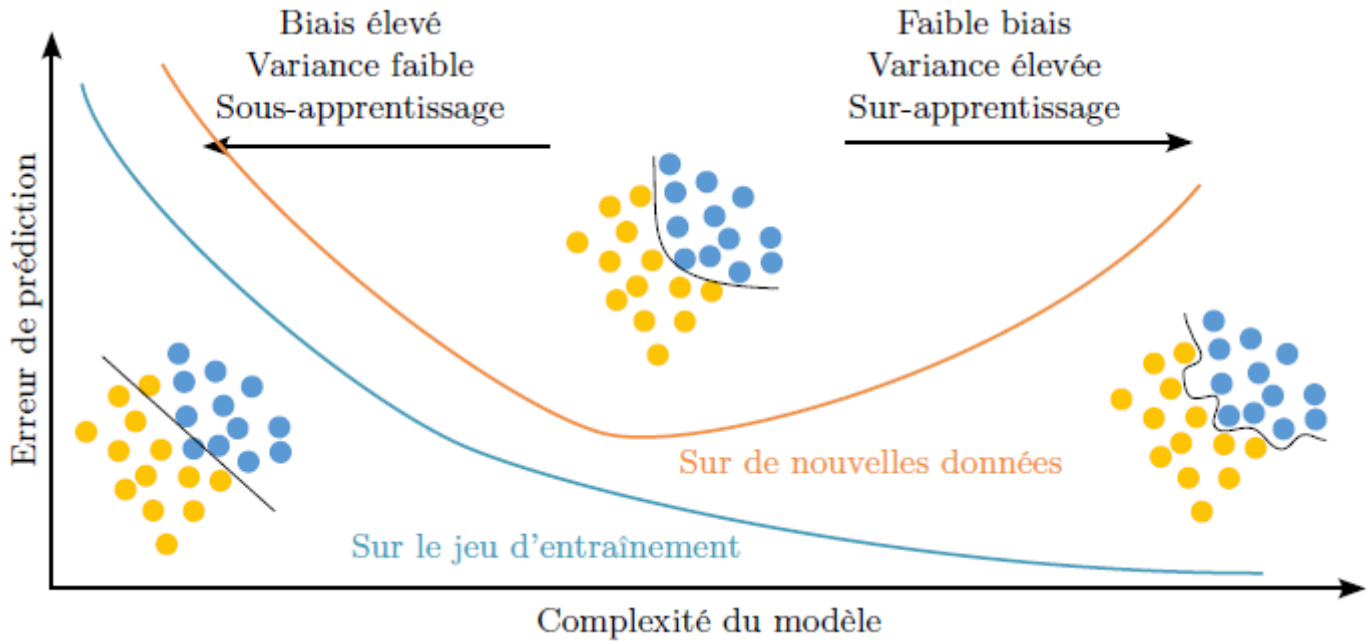


Algorithmes d'apprentissage

Le danger du surapprentissage – Dilemme biais-variance

Modèle complexe → très bon sur les enregistrements utilisés pour l'apprentissage

→ performances médiocres sur de nouvelles données



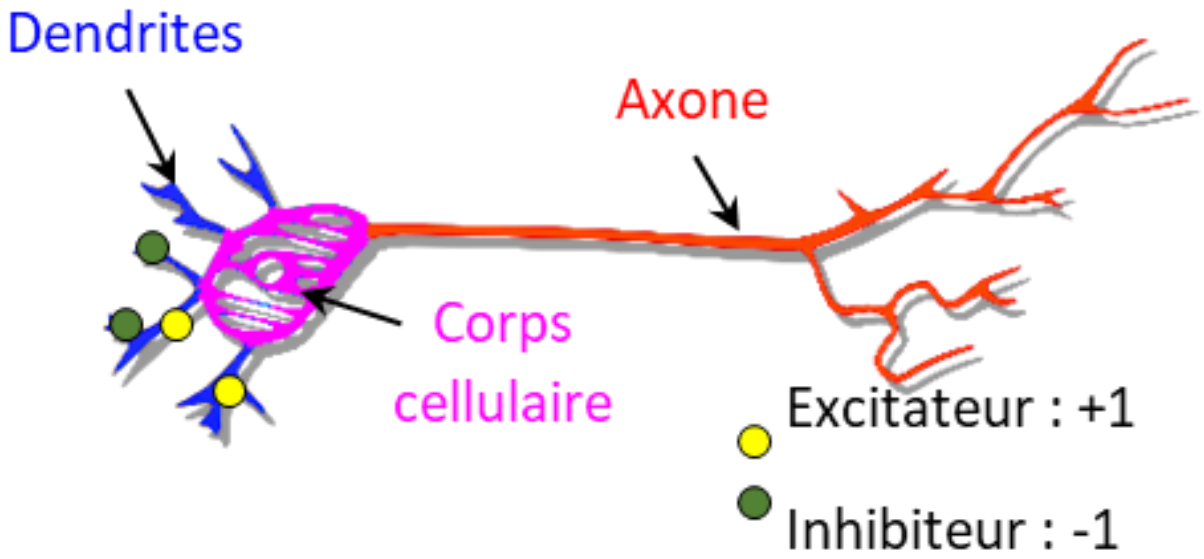
Les réseaux de neurones

Exemples d'applications des réseaux de neurones

- **Cartes de crédit :** détection des fraudes.
- **Finance :** analyse d'investissements et de fluctuations des taux de change.
- **Assurance :** couverture assurantielle et estimation des réserves.
- **Marketing :** ciblage des prospections, mesures et comparaisons des campagnes.
- **Archéologie :** identification et datation de fossiles et d'ossements.
- **Défense :** identification de cibles.
- **Production :** contrôles qualité.
- **Médecine :** diagnostics médicaux.
- **Energies :** estimations des réserves, prévisions de prix.
- **Pharmacie :** efficacité de nouveaux médicaments.
- **Psychologie :** prévisions comportementales.
- **Immobilier :** études de marchés.
- **Recherche scientifique :** identification de spécimens, séquençages de protéines.
- **Télécommunication :** détection des pannes de réseaux.
- **Transport :** maintenance des voies.
- **…**

Modèle de neurone

Neurone biologique



Si le seuil d'activation est dépassé, un signal est envoyé aux autres neurones par le biais de l'axone

Modèle de neurone

Définition d'un neurone (ou perceptron)

- **X** : vecteur d'entrée et x_i les données de la couche d'entrée.
- ω_i : poids (poids synaptiques).
- b : biais.
- z_0 : somme pondérée des entrées.
- f : fonction d'activation.
- $\tilde{y}_0$: valeur de sortie du neurone.

$$z_0 = b + \sum_{i=0}^{n} \omega_i x_i$$

Modèle de neurone

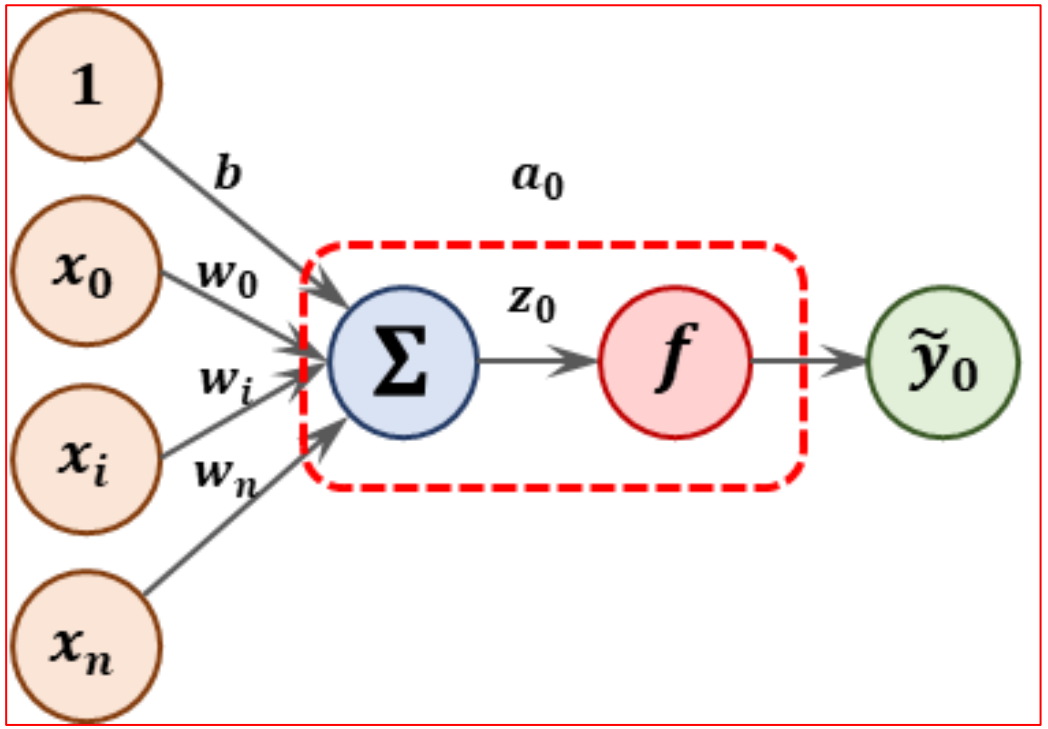
Définition d'un neurone (ou perceptron)

Sortie du neurone :

$$\tilde{y}_0 = f(z_0) = f(b + \sum_{i=0}^{n} \omega_i x_i)$$

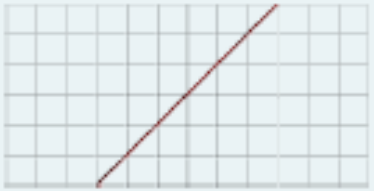
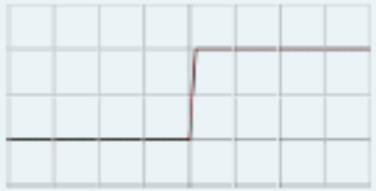
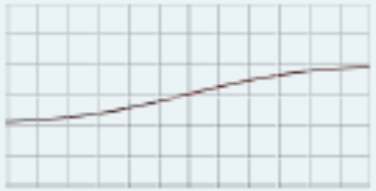
- **X** : vecteur d'entrée et x_i les données de la couche d'entrée.
- ω_i : poids (poids synaptiques).
- b : biais.
- z_0 : somme pondérée des entrées. $z_0 = b + \sum_{i=0}^{n} \omega_i x_i$
- f : fonction d'activation.
- $\tilde{y}_0$: valeur de sortie du neurone.

Remarque : Notation tilde ($\tilde{y}_0$) → valeur de sortie d'un neurone → valeur estimée



Modèle de neurone

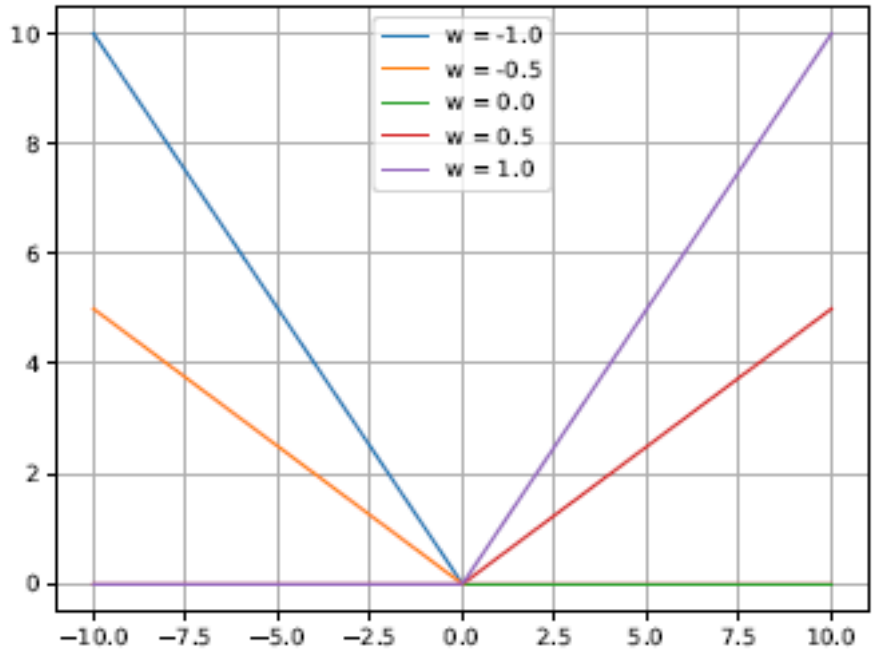
Définition d'une fonction d'activation :

Identité	Heaviside	Logistique (sigmoïde)	Unité de rectification linéaire (ReLU)
			
$f(x)=x$	$f(x)=\begin{cases} 0 \text{ si } x<0 \\ 1 \text{ si } x\geq 0 \end{cases}$	$f(x)=\dfrac{1}{1+e^{-x}}$	$f(x)=\begin{cases} 0 \text{ si } x<0 \\ x \text{ si } x\geq 0 \end{cases}$

Modèle de neurone

Définition d'une fonction d'activation :

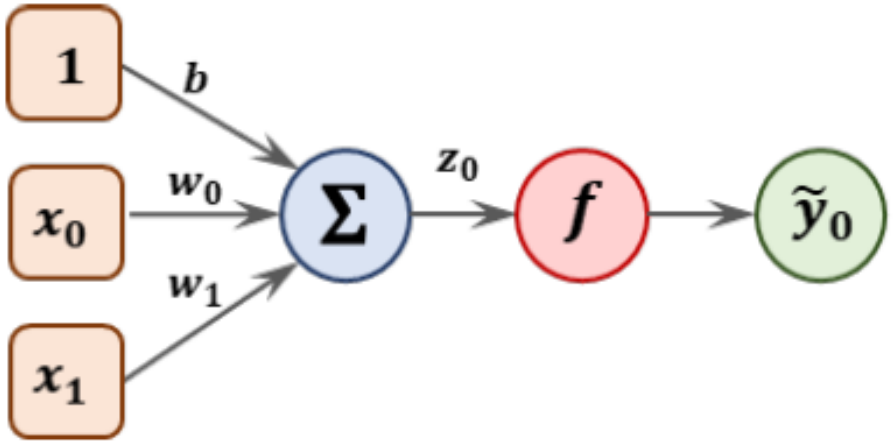
Remarque : Influence des poids et des biais sur la sortie du neurone en utilisant une fonction d'activation *ReLU*.



Modèle de neurone

Exemple de calcul avec un réseau de neurones

Initialisation les poids et le biais avec des valeurs aléatoires : $\omega_0 = -0{,}3 \quad \omega_1 = 0{,}8 \;\; et \; b = 0{,}2$.



x_0	x_1	z	Id.	H.	Sig.	ReLu
0	0	0,2	0,2	1	0.549	0,2
0	1	1	1	1	0.731	1
1	0	-0.1	-0.1	0	0.475	0
1	1	0.7	0.7	1	0.668	0.7

Réseaux de neurones

Modélisation d'un réseau de neurones

Définition des couches

Un réseau de neurones est un ensemble de neurones reliés, par couches, entre eux.

Dans un réseau de neurones **dense** tous les neurones de la couche i seront reliés à tous les neurones de la couche $i+1$.

Réseaux de neurones

Modélisation d'un réseau de neurones

Définition des couches

- **Couche d'entrée** → copie de l'ensemble des données d'entrées.

Nombre de neurones de cette couche correspond aux nombres de données d'entrées.

- **Couche cachée (ou couche intermédiaire)** → utilité intrinsèque au réseau de neurones.

Ajouter des neurones dans cette couche (ou ces couches) → ajouter de nouveaux paramètres.
Pour une couche → même fonction d'activation pour tous les neurones.
Fonction d'activation utilisée → peut être différente pour deux couches différentes.

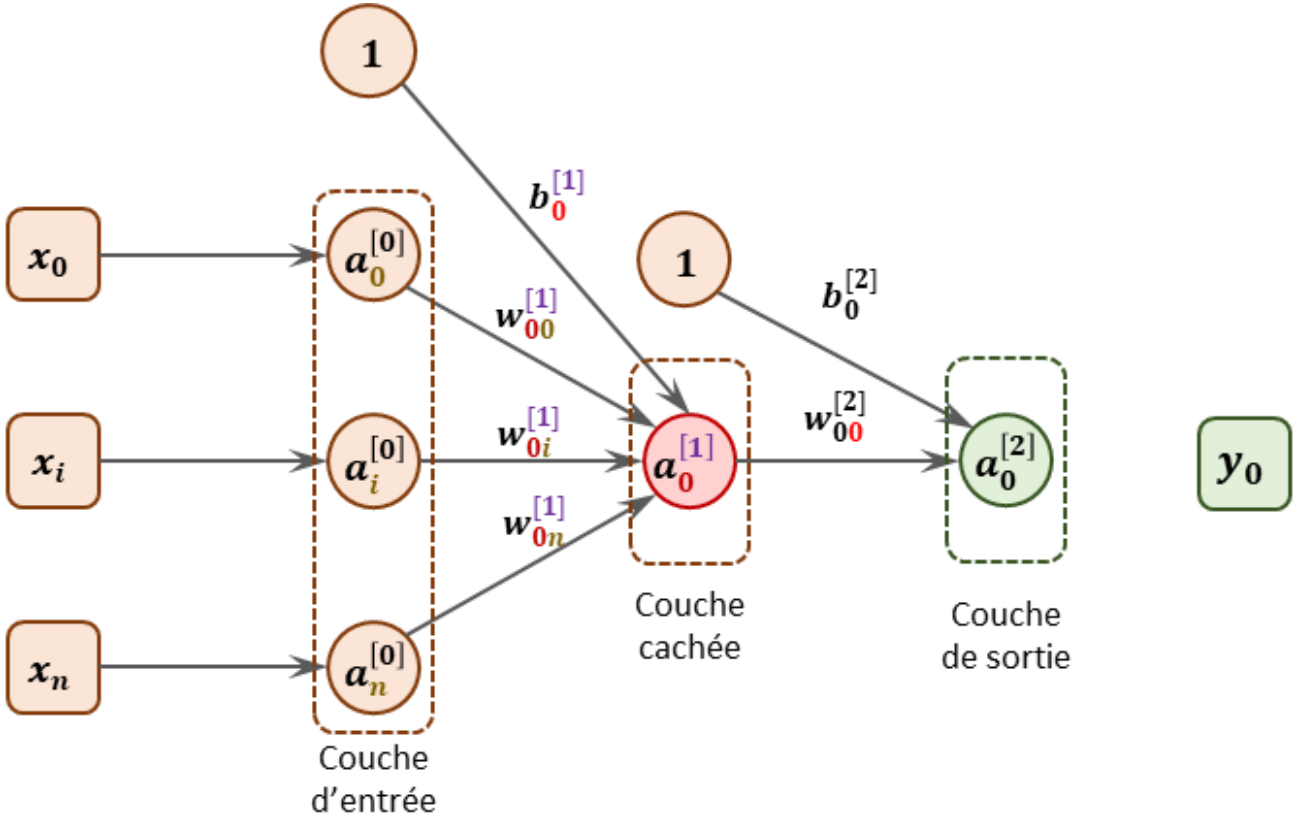
- **Couche de sortie :** Nombre de neurones → nombre de sorties attendues.

Fonction d'activation de la couche de sortie → souvent linéaire.

Réseaux de neurones

Modélisation d'un réseau de neurones

Définition des couches



Remarque : Chaque couche a sa propre matrice de poids, son propre vecteur de biais, un vecteur d'entrée et un vecteur de sortie.

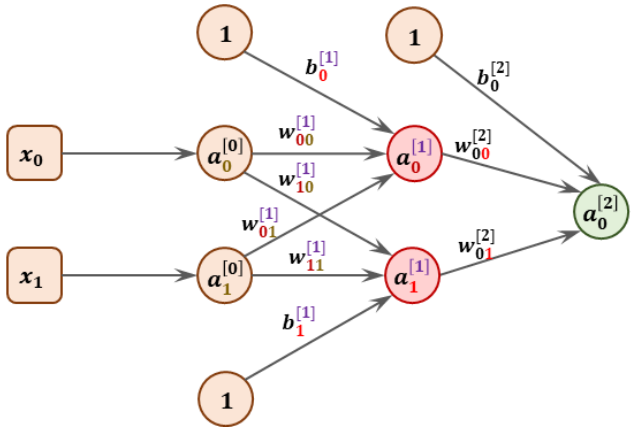
Réseaux de neurones

Modélisation d'un réseau de neurones

Définition de l'équation de propagation

Pour chacun des neurones $a_j^{[l]}$ → équation de propagation :

$$\boldsymbol{a_j^{[l]} = f^{[l]}\left(\sum_{k=0}^{n^{[l-1]}} \left(\omega_{jk}^{[l]} a_k^{[l-1]}\right) + b_j^{[l]}\right) = f^{[l]}(z_j^{[l]})}$$

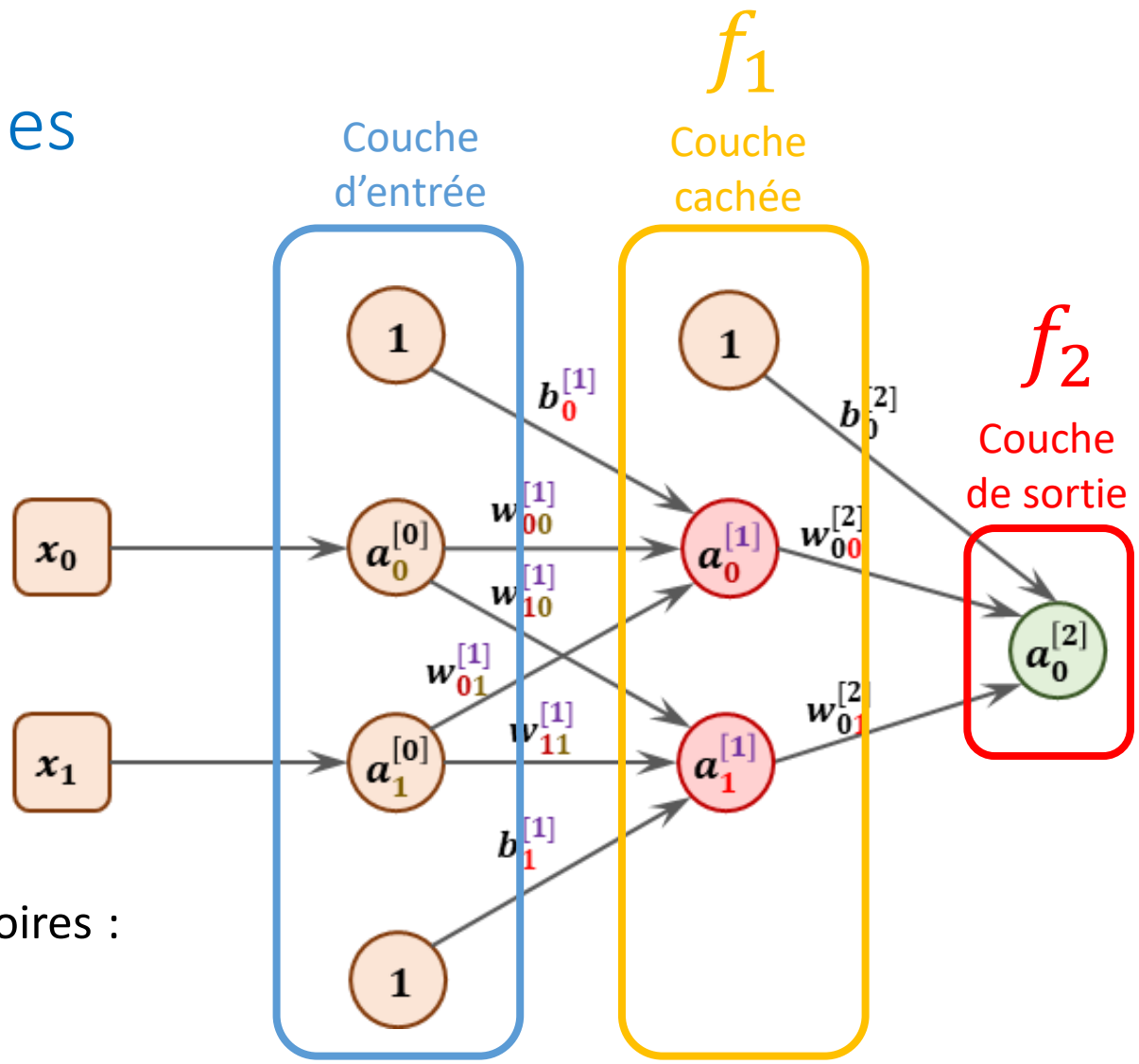


Réseaux de neurones

Modélisation d'un réseau de neurones

Exemple (réseau de neurone à 3 couches)

- 1 couche d'entrée à 2 neurones
- 1 couche cachée à 2 neurones de fonction d'activation f_1
- 1 couche de sortie à 1 neurone de fonction d'activation f_2



Initialisation les poids et le biais avec des valeurs aléatoires :
$\omega_0 = -0{,}3 \quad \omega_1 = 0{,}8 \;\; et \; b = 0{,}2.$

Réseaux de neurones

Modélisation d'un réseau de neurones

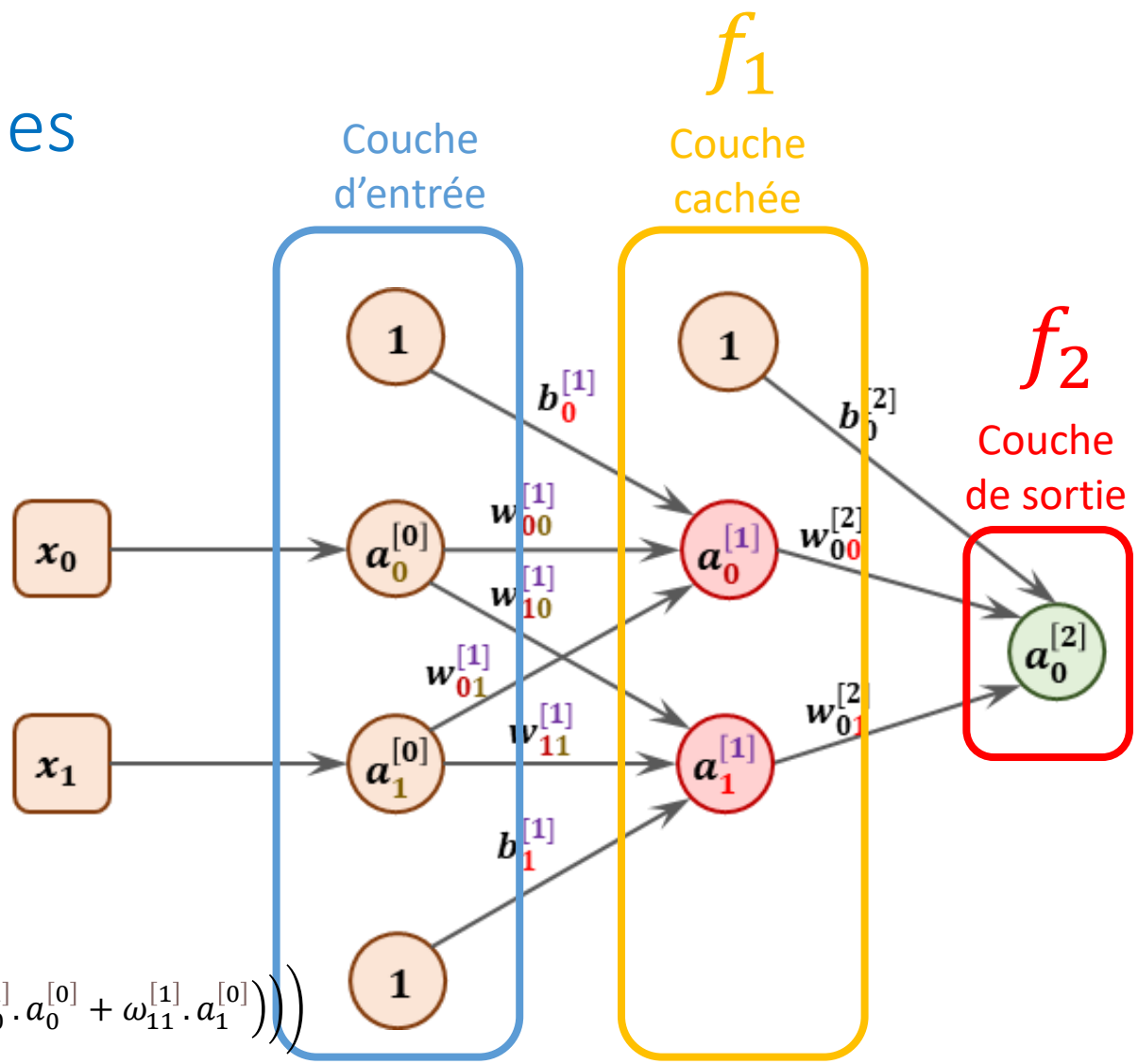
Exemple (réseau de neurone à 3 couches)

$$y_0 = a_0^{[2]}$$

$$y_0 = a_0^{[2]} = f_2\left(b_0^{[2]} + \omega_{00}^{[2]}.\boldsymbol{a_0^{[1]}} + \omega_{01}^{[2]}.\boldsymbol{a_1^{[1]}}\right)$$

$$a_0^{[1]} = f_1(b_0^{[1]} + \omega_{00}^{[1]}.a_0^{[0]} + \omega_{01}^{[1]}.a_1^{[0]})$$

$$a_1^{[1]} = f_1(b_1^{[1]} + \omega_{10}^{[1]}.a_0^{[0]} + \omega_{11}^{[1]}.a_1^{[0]})$$



$$y_0 = a_0^{[2]} = f_2\left(b_0^{[2]} + \omega_{00}^{[2]}.\left(f_1\left(b_0^{[1]} + \omega_{00}^{[1]}.a_0^{[0]} + \omega_{01}^{[1]}.a_1^{[0]}\right)\right) + \omega_{01}^{[2]}.\left(f_1\left(b_1^{[1]} + \omega_{10}^{[1]}.a_0^{[0]} + \omega_{11}^{[1]}.a_1^{[0]}\right)\right)\right)$$

Réseaux de neurones

Modélisation d'un réseau de neurones

Définition des paramètres

Les paramètres du réseau de neurones sont les poids et les biais, autant de valeurs que l'entraînement devra déterminer.

Poids ω_i + biais b pour chaque neurone → déterminés → données d'entrainement
→ **méthode de descente du gradient**

Réseaux de neurones

Modélisation d'un réseau de neurones

Méthode : Calcul du nombre de paramètres

n entrées *p* sorties l couches a_l nombre de neurones de la couche l.

Nombre de poids : $n_\omega = \sum_{i=1}^{l-1}(a_i . a_{i+1})$ **Nombre de bais :** $n_b = \sum_{i=2}^{l}(a_i)$

Nombre total de paramètre à calculer : $N = n_\omega + n_b$.

Objectif de la phase d'apprentissage : Déterminer les valeurs de l'ensemble des poids et des biais de telle sorte que l'écart entre les entrées et le résultat prédit soit minimale.

Réseaux de neurones

Etapes préliminaires à l'entrainement

- Sélection des données.
- Prétraitement des données.
- Choix du type de réseau et de son architecture.

Réseaux de neurones

Etapes préliminaires à l'entrainement

Sélection des données

Qualité du réseau ←→ Qualité des données utilisées pour l'entraîner.

Les données relatives à l'entrainement doivent couvrir l'ensemble de l'espace des entrées d'utilisation du réseau.

« *Avons-nous suffisamment de données ?* »

→ Quantité de données requises ←→ Complexité de la fonction que nous essayons d'approximer.

→ Processus d'entrainement du réseau neuronal → itératif (analyse après chaque entrainement des performances du réseau)

Réseaux de neurones

Etapes préliminaires à l'entrainement

Prétraitement des données

Prétraitement des données → faciliter l'entrainement au réseau.

Prétraitement des données → normalisation / transformations non linéaires / extraction de caractéristiques / traitement des données manquantes / …

Réseaux de neurones

Etapes préliminaires à l'entrainement

Choix du réseau

Type de base de l'architecture de réseau ←→ Type de problème que nous souhaitons résoudre.

Une fois que l'architecture de base est choisie → décider → nombre de neurones / nombre de couches / nombre de sorties / fonction de performance pour l'entrainement / …

Réseaux de neurones

Entrainement

Après la préparation des données et la mise en place de l'architecture du réseau sélectionnés, nous sommes prêts à entrainer le réseau.

Initialiser les poids et les biais (généralement fixés à de petites valeurs aléatoires, par exemple, répartis uniformément entre -0,5 et 0,5).

Entrainement des réseaux de neurones → Processus itératif

Même après convergence de l'algorithme → plusieurs cycles d'entrainement

Réseaux de neurones

Fonction de coût (cost function ou de perte, loss function)

Définition de la fonction de coût

Objectif : Minimiser l'écart entre la sortie du réseau de neurones et la valeur réelle de la sortie

n_b : nombre de données dans la base d'entraînement

Fonction de coût (moyenne des erreurs quadratique) :

$$C = \frac{1}{n_b} \sum_{i=1}^{n_b} (\tilde{Y}_i - Y_i)^2$$

Objectif : Déterminer les poids et les biais qui minimisent la fonction coût.

Réseaux de neurones

Fin d'apprentissage

Définition de l'epoch

Cycle d'apprentissage où tous les poids et tous les biais ont été mis à jour en faisant passer toutes les données du jeu d'entraînement dans les algorithmes de propagation et de rétropropagation.

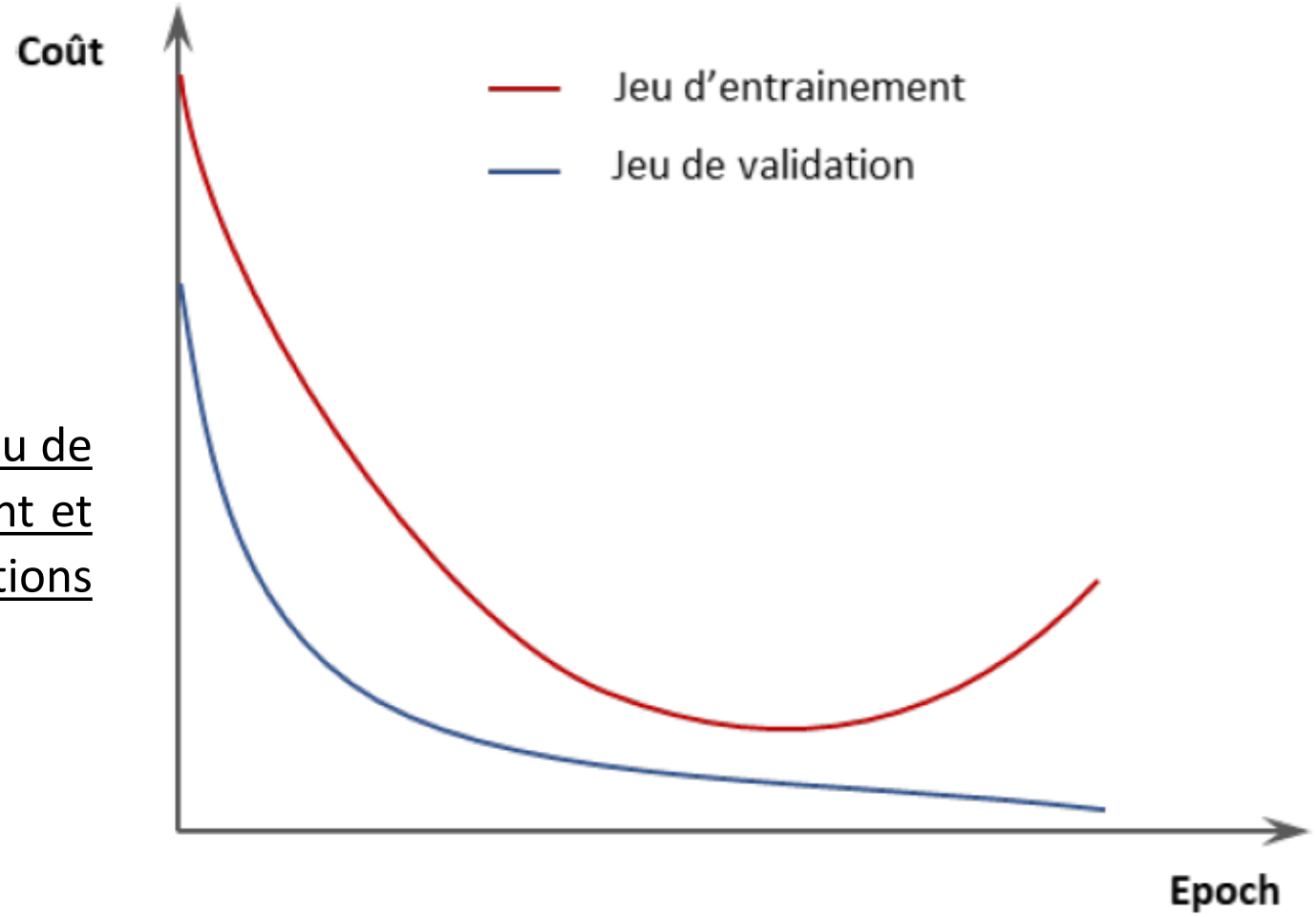
Réseaux de neurones

Fin d'apprentissage

Définition de l'epoch

Il existe en fait un stade à partir duquel le réseau de neurones se spécialise sur le jeu d'entraînement et devient donc incapable de réaliser des prédictions fiables sur un nouveau jeu de données.

On parle de surentraînement (ou *overfitting*).



Réseaux de neurones

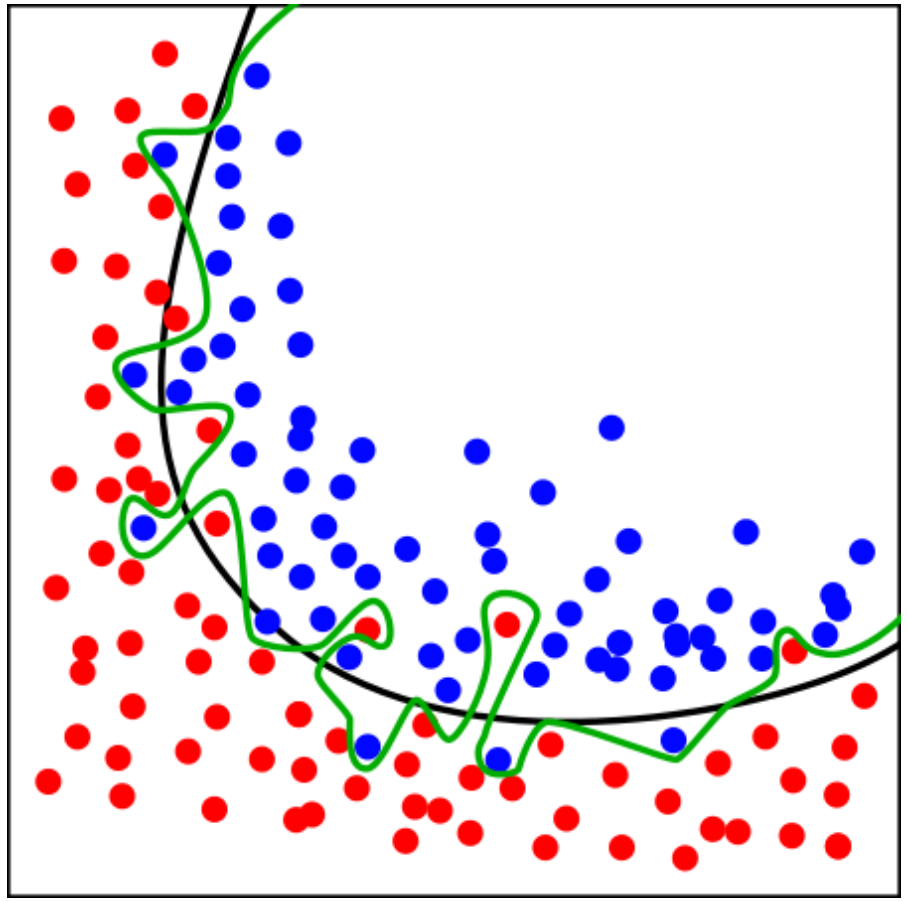
Fin d'apprentissage

Définition de l'epoch

Exemple :

Ligne verte → modèle sur-appris

Ligne noire → modèle régulier



Ligne verte classifie trop parfaitement les données d'entraînement, elle généralise mal et donnera de mauvaises prévisions futures avec de nouvelles données.

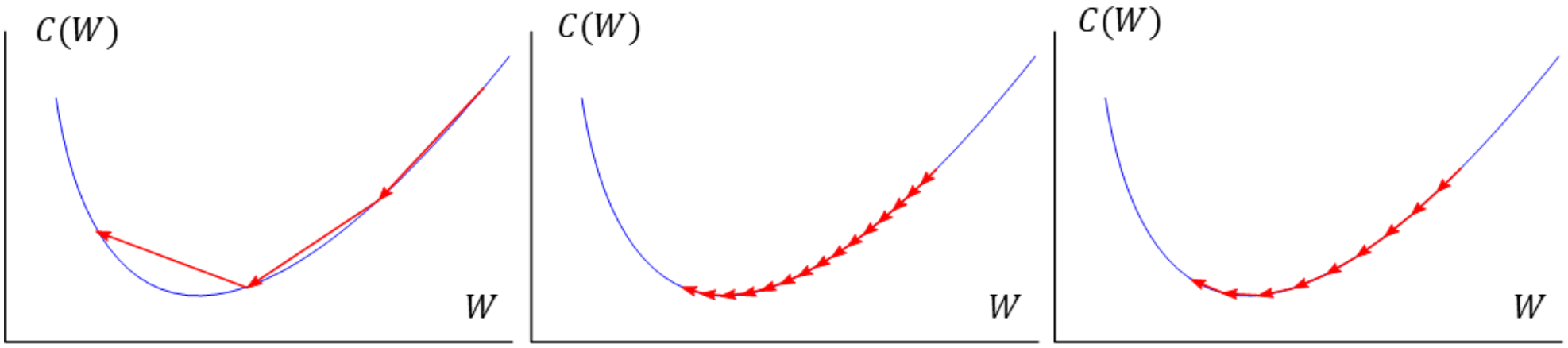
Modèle vert → moins bon → Modèle noir

Minimiser la fonction le coût (cas général) : La méthode de descente de Gradient

Etude variation de la fonction coût à chaque couche du modèle + minimiser la valeur globale du coût
→ **Rétropropagation → Méthode de descente de gradient**

Fonction de coût $C(W) = \sum_i^n (f(x_i) - y_i)^2$ et $W = \{w_1, w_2, \dots, w_i\}$, w_i paramètres de $f(x)$

→ Se déplacer dans la direction opposée au gradient de la fonction = méthode de « descente » la plus rapide



Minimiser la fonction le coût (cas général) : La méthode de descente de Gradient

Faire varier en suivant la plus grande pente pour arriver le plus vite possible au minimum

$$\boldsymbol{W_{p+1} = W_p - \alpha \nabla C(W)}$$

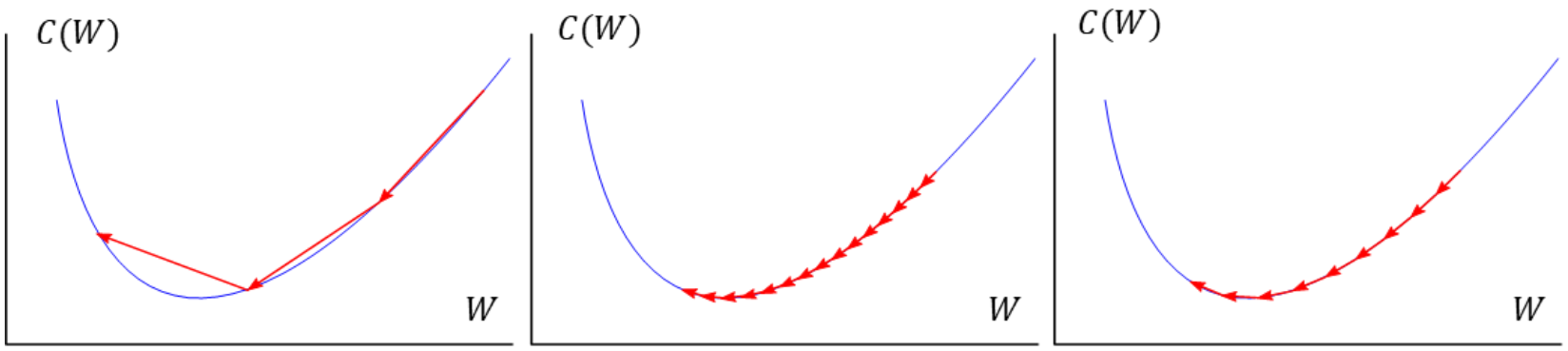
$\nabla C(W)$ = direction du gradient en W

Signe (–) = direction opposée au gradient

α = facteur d'apprentissage (*learning rate*)

Si α est trop grand → rater le point le plus bas

Si α est trop petit → descente trop lente



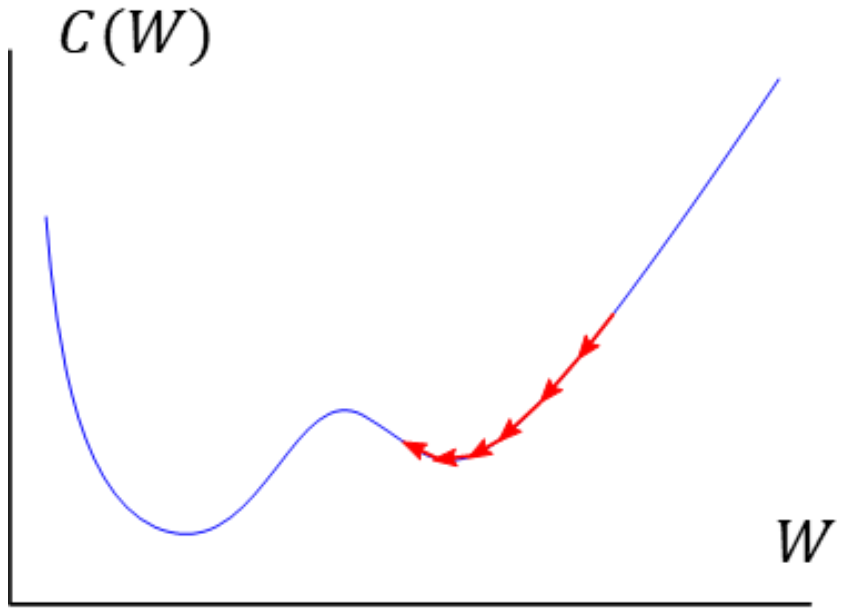
Minimiser la fonction le coût (cas général) : La méthode de descente de Gradient

$\alpha \in [0; 1[$ variable

Stop → certain nombre d'itération maxi atteint (différence entre 2 valeurs de pentes successives < ϵ fixé)

→ ou erreur absolue entre 2 itérations < seuil fixé

Problème → minimum local



Estimation → sensible au bruit de mesure (données d'apprentissage)
donnée aberrante → influera → estimations des paramètres

Minimiser la fonction le coût (cas général) : La méthode de descente de Gradient

Exemple d'application :

$X_I = (x_1, x_2, x_3, x_4)$ vecteur de la combinaison des entrées

Poids W et biais B → MAJ → Méthode de la descente du gradient.

→ Première couche du réseau de neurones → Entrée vecteur X + Sortie vecteur d'activations A = $f(W_1X + B_1)$
→ Fonctionnement des autres couches → identique

Notation : W = W1, B = B1 et A = f(Y)= f(WX + B)

Minimiser la fonction le coût (cas général) : La méthode de descente de Gradient

Exemple d'application :

W et B → MAJ → $\begin{cases} W = W - \alpha \frac{\partial C}{\partial W} \\ B = B - \alpha \frac{\partial C}{\partial B} \end{cases}$ α = taux d'apprentissage C = fonction de coût

Notation : $\frac{\partial C}{\partial W} = \begin{bmatrix} \frac{\partial C}{\partial w_{11}} & \cdots & \frac{\partial C}{\partial w_{14}} \\ \cdots & \cdots & \cdots \\ \frac{\partial C}{\partial w_{41}} & \cdots & \frac{\partial C}{\partial w_{44}} \end{bmatrix}$ $\frac{\partial C}{\partial X} = \begin{bmatrix} \frac{\partial C}{\partial x_1} \\ \cdots \\ \frac{\partial C}{\partial x_4} \end{bmatrix}$ $\frac{\partial C}{\partial A} = \begin{bmatrix} \frac{\partial C}{\partial a_1} \\ \cdots \\ \frac{\partial C}{\partial a_4} \end{bmatrix}$

$$\frac{\partial C}{\partial w_{ij}} = \frac{\partial C}{\partial a_1}\frac{\partial a_1}{\partial w_{ij}} + \cdots + \frac{\partial C}{\partial a_i}\frac{\partial a_i}{\partial w_{ij}} + \cdots + \frac{\partial C}{\partial a_4}\frac{\partial a_4}{\partial w_{ij}} \text{ et } a_i = f(\sum_j w_{ij}.x_j + bi)$$

Minimiser la fonction le coût (cas général) : La méthode de descente de Gradient

Exemple d'application :

Question : Montrer que : $\frac{\partial C}{\partial w_{ij}} = \frac{\partial C}{\partial a_i} . f'(y_i) . x_j$

$$\frac{\partial C}{\partial w_{ij}} = \frac{\partial C}{\partial a_1}\frac{\partial a_1}{\partial w_{ij}} + \ldots + \frac{\partial C}{\partial a_i}\frac{\partial a_i}{\partial w_{ij}} + \ldots + \frac{\partial C}{\partial a_4}\frac{\partial a_4}{\partial w_{ij}}$$

Avec $a_i = f(w_{i1}x_1 + w_{i2}x_2 + w_{i3}x_3 + w_{i4}x_4 + b_i)$,

On obtient $\frac{\partial a_i}{\partial w_{ij}} = x_j f'(w_{i1}x_1 + w_{i2}x_2 + w_{i3}x_3 + w_{i4}x_4 + b_i) = x_j f'(y_i)$

Remarque : $\frac{\partial a_{k\neq i}}{\partial w_{ij}} = 0$ car indépendant de w_{ij}

On a bien $\frac{\partial C}{\partial w_{ij}} = \frac{\partial C}{\partial a_i} f'(y_i) x_j$

Minimiser la fonction le coût (cas général) : La méthode de descente de Gradient

Question : En utilisant la relation précédente, donner la relation matricielle entre $\frac{\partial C}{\partial W}$, $\frac{\partial C}{\partial A}$, $f'(Y)$ et X, puis entre $\frac{\partial C}{\partial W}$, $\frac{\partial C}{\partial A}$, *W*, X et *B*.

On a besoin du produit $\frac{\partial C}{\partial A} . f'(Y)$ dont le résultat est un vecteur colonne $\frac{\partial \mathrm{C}}{\partial \mathrm{W}} = \left(\frac{\partial \mathrm{C}}{\partial \mathrm{A}} . \mathrm{f}'(\mathrm{Y})\right) \mathrm{X}^{\mathrm{T}}$

$$\frac{\partial \mathbf{C}}{\partial \mathbf{W}} = \left(\frac{\partial \mathbf{C}}{\partial \mathbf{A}} . \mathbf{f}'(\mathbf{WX} + \mathbf{B})\right) \mathbf{X}^{\mathbf{T}}$$

Relation → erreur en entrée en fonction de l'erreur en sortie : $\frac{\partial \mathbf{C}}{\partial \mathbf{X}} = \frac{\partial \mathbf{C}}{\partial \mathbf{A}} (\mathbf{W} . \mathbf{f}'(\mathbf{WX} + \mathbf{B}))$

Choix d'une solution pour un problème de prédiction

	Modèle linéaire (régression ou classification)	k plus proches voisins	Réseaux de neurones (éventuellement profonds)
Temps de calcul à l'apprentissage	+ Inversion de matrice et calcul de $X^T X$ $\mathcal{O}(d^3 + nd^2)$ mais existence de méthodes numériques plus efficaces	++ Stockage des données de la base de donnée d'entraînement	- - Procédure longue et coûteuse notamment sur de grandes bases de données
Temps de calcul à la prédiction	++ 1 Produit matrice vecteur $\mathcal{O}(d \times p)$	- Tri d'un tableau de distances $\mathcal{O}(n \log(n))$	+ Séquence de produits matrice vecteur
Performances avec peu de données d'entrainement	++ Peu de paramètres à apprendre $\rightarrow$ bonnes performances avec peu de données	+ Performances correctes avec peu de voisins	+ Peu de paramètres par couche conduisent à un modèle proche du linéaire
Performances avec données abondantes	- Peu de paramètres apprenables	+ Réglage du nombre de voisins	++ Grande flexibilité de la fonction de prédiction
Performance avec données complexes (image / texte)	- Représentation des entrées inadaptable efficacement	- Difficile de choisir une distance correcte	++ Possibilité d'adapter les couches cachées aux propriétés des entrées

Modules pythons pour l'IA

Python For Data Science *Cheat Sheet*

Scikit-Learn

Learn Python for data science Interactively at www.DataCamp.com

Scikit-learn

Scikit-learn is an open source Python library that implements a range of machine learning, preprocessing, cross-validation and visualization algorithms using a unified interface.

A Basic Example

```
>>> from sklearn import neighbors, datasets, preprocessing
>>> from sklearn.model_selection import train_test_split
>>> from sklearn.metrics import accuracy_score
>>> iris = datasets.load_iris()
>>> X, y = iris.data[:, :2], iris.target
>>> X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=33)
>>> scaler = preprocessing.StandardScaler().fit(X_train)
>>> X_train = scaler.transform(X_train)
>>> X_test = scaler.transform(X_test)
>>> knn = neighbors.KNeighborsClassifier(n_neighbors=5)
>>> knn.fit(X_train, y_train)
>>> y_pred = knn.predict(X_test)
>>> accuracy_score(y_test, y_pred)
```

Loading The Data

Also see NumPy & Pandas

Your data needs to be numeric and stored as NumPy arrays or SciPy sparse matrices. Other types that are convertible to numeric arrays, such as Pandas DataFrame, are also acceptable.

```
>>> import numpy as np
>>> X = np.random.random((10,5))
>>> y = np.array(['M','M','F','F','M','F','M','M','F','F','F'])
>>> X[X < 0.7] = 0
```

Training And Test Data

```
>>> from sklearn.model_selection import train_test_split
>>> X_train, X_test, y_train, y_test = train_test_split(X,
                                                        y,
                                                        random_state=0)
```

Create Your Model

Supervised Learning Estimators

Linear Regression

```
>>> from sklearn.linear_model import LinearRegression
>>> lr = LinearRegression(normalize=True)
```

Support Vector Machines (SVM)

```
>>> from sklearn.svm import SVC
>>> svc = SVC(kernel='linear')
```

Naive Bayes

```
>>> from sklearn.naive_bayes import GaussianNB
>>> gnb = GaussianNB()
```

KNN

```
>>> from sklearn import neighbors
>>> knn = neighbors.KNeighborsClassifier(n_neighbors=5)
```

Unsupervised Learning Estimators

Principal Component Analysis (PCA)

```
>>> from sklearn.decomposition import PCA
>>> pca = PCA(n_components=0.95)
```

K Means

```
>>> from sklearn.cluster import KMeans
>>> k_means = KMeans(n_clusters=3, random_state=0)
```

Model Fitting

Supervised learning	
`>>> lr.fit(X, y)`	Fit the model to the data
`>>> knn.fit(X_train, y_train)`	
`>>> svc.fit(X_train, y_train)`	
Unsupervised Learning	
`>>> k_means.fit(X_train)`	Fit the model to the data
`>>> pca_model = pca.fit_transform(X_train)`	Fit to data, then transform it

Prediction

Supervised Estimators	
`>>> y_pred = svc.predict(np.random.random((2,5)))`	Predict labels
`>>> y_pred = lr.predict(X_test)`	Predict labels
`>>> y_pred = knn.predict_proba(X_test)`	Estimate probability of a label
Unsupervised Estimators	
`>>> y_pred = k_means.predict(X_test)`	Predict labels in clustering algos

Preprocessing The Data

Standardization

```
>>> from sklearn.preprocessing import StandardScaler
>>> scaler = StandardScaler().fit(X_train)
>>> standardized_X = scaler.transform(X_train)
>>> standardized_X_test = scaler.transform(X_test)
```

Normalization

```
>>> from sklearn.preprocessing import Normalizer
>>> scaler = Normalizer().fit(X_train)
>>> normalized_X = scaler.transform(X_train)
>>> normalized_X_test = scaler.transform(X_test)
```

Binarization

```
>>> from sklearn.preprocessing import Binarizer
>>> binarizer = Binarizer(threshold=0.0).fit(X)
>>> binary_X = binarizer.transform(X)
```

Encoding Categorical Features

```
>>> from sklearn.preprocessing import LabelEncoder
>>> enc = LabelEncoder()
>>> y = enc.fit_transform(y)
```

Imputing Missing Values

```
>>> from sklearn.preprocessing import Imputer
>>> imp = Imputer(missing_values=0, strategy='mean', axis=0)
>>> imp.fit_transform(X_train)
```

Generating Polynomial Features

```
>>> from sklearn.preprocessing import PolynomialFeatures
>>> poly = PolynomialFeatures(5)
>>> poly.fit_transform(X)
```

Evaluate Your Model's Performance

Classification Metrics

Accuracy Score	
`>>> knn.score(X_test, y_test)`	Estimator score method
`>>> from sklearn.metrics import accuracy_score`	Metric scoring functions
`>>> accuracy_score(y_test, y_pred)`	
Classification Report	
`>>> from sklearn.metrics import classification_report`	Precision, recall, f1-score and support
`>>> print(classification_report(y_test, y_pred))`	
Confusion Matrix	
`>>> from sklearn.metrics import confusion_matrix`	
`>>> print(confusion_matrix(y_test, y_pred))`	

Regression Metrics

Mean Absolute Error

```
>>> from sklearn.metrics import mean_absolute_error
>>> y_true = [3, -0.5, 2]
>>> mean_absolute_error(y_true, y_pred)
```

Mean Squared Error

```
>>> from sklearn.metrics import mean_squared_error
>>> mean_squared_error(y_test, y_pred)
```

R^2 Score

```
>>> from sklearn.metrics import r2_score
>>> r2_score(y_true, y_pred)
```

Clustering Metrics

Adjusted Rand Index

```
>>> from sklearn.metrics import adjusted_rand_score
>>> adjusted_rand_score(y_true, y_pred)
```

Homogeneity

```
>>> from sklearn.metrics import homogeneity_score
>>> homogeneity_score(y_true, y_pred)
```

V-measure

```
>>> from sklearn.metrics import v_measure_score
>>> metrics.v_measure_score(y_true, y_pred)
```

Cross-Validation

```
>>> from sklearn.cross_validation import cross_val_score
>>> print(cross_val_score(knn, X_train, y_train, cv=4))
>>> print(cross_val_score(lr, X, y, cv=2))
```

Tune Your Model

Grid Search

```
>>> from sklearn.grid_search import GridSearchCV
>>> params = {"n_neighbors": np.arange(1,3),
              "metric": ["euclidean", "cityblock"]}
>>> grid = GridSearchCV(estimator=knn,
                        param_grid=params)
>>> grid.fit(X_train, y_train)
>>> print(grid.best_score_)
>>> print(grid.best_estimator_.n_neighbors)
```

Randomized Parameter Optimization

```
>>> from sklearn.grid_search import RandomizedSearchCV
>>> params = {"n_neighbors": range(1,5),
              "weights": ["uniform", "distance"]}
>>> rsearch = RandomizedSearchCV(estimator=knn,
                                 param_distributions=params,
                                 cv=4,
                                 n_iter=8,
                                 random_state=5)
>>> rsearch.fit(X_train, y_train)
>>> print(rsearch.best_score_)
```

DataCamp
Learn Python for Data Science Interactively