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***

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

1.1. 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…

Les domaines d'applications de l'IA sont de plus en plus nombreux, avec certains très controversés. Parmi les plus courants, on trouve notamment :

- La finance, avec une aide à la décision quant à l'acceptation éventuelle d'un prêt, la gestion d'un fond d'investissement
- La médecine, avec une aide au diagnostic de maladie et de préconisations médicamenteuses
- Le militaire, avec des systèmes d'aide à la décision et de commandement
- La robotique, avec des systèmes d'aide à la décision
- Le contrôle d'accès, avec des systèmes de reconnaissance faciale par exemple
- Le journalisme, avec des systèmes de reconnaissances de *fakenews*
- Les réseaux sociaux, avec des systèmes de suivi de la popularité d'une personne, ou l'identification de diffusion de message à caractère haineux
- Les messageries *mail* avec des propositions de réponses automatiques
- La reconnaissance de caractères ou la traduction

Exemple : Identification d'objets, en temps réel, dans une vidéo.

On remarquera sur l'exemple de gauche que l'algorithme parvient à identifier sur l'image des zones puis à identifier à quoi elles correspondent. On remarquera aussi que la flamme est identifiée comme étant un donut...

Les problèmes types de l'IA :

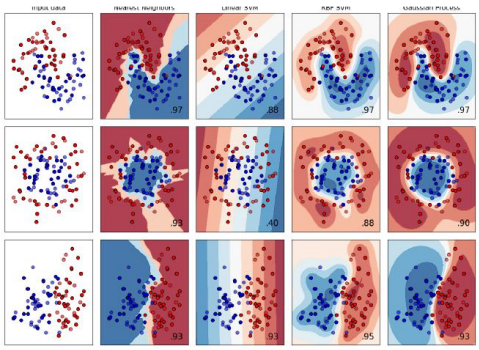
Les thèmes d'utilisation de l'IA (ou plutôt du M*achine Learning*) sont :

- **La *classification*** (idem en anglais) **(a)** : Il s'agit d'identifier à quelle catégorie un objet appartient (apprentissage supervisé).
  Applications : Détection de spam, reconnaissance d'images.

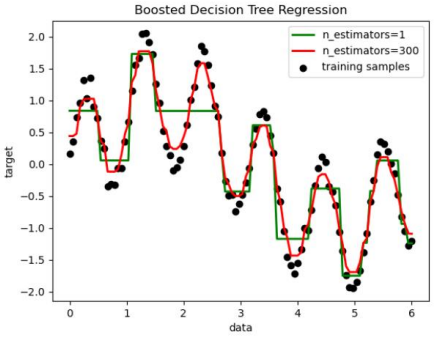
- **La *régression*** (idem en anglais) (b) : Le but est de prédire un paramètre à valeur continue associée à un objet.
  Applications : Les réponses aux médicaments, le cours de la bourse.

- **La *segmentation*** *(*ou *clustering)* **(c)** *:* Regrouper de manière automatique des objets dans des ensembles (apprentissage non supervisé).
  Applications : Segmentation marketing, regroupement des résultats des expériences.

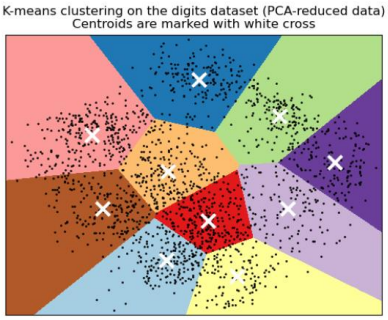
- **La *prédiction*** (idem en anglais) **(d)** : Prédire des données temporelles.
  Applications : Reconnaissance vocale, la météo ou encore la correction d'un système.



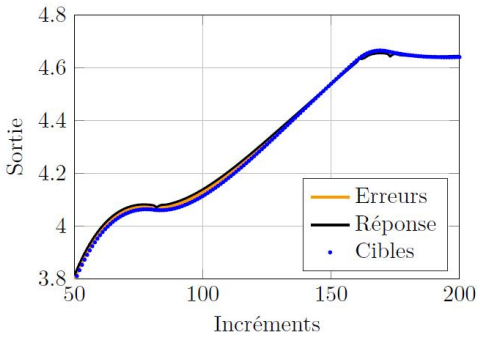
(a) Classification



(b) Régression



(c) Clustering



(d) Prédiction

Pour utiliser un algorithme d'IA il est nécessaire de disposer de données... en général beaucoup de données.

Définition des données :

Les données utilisées par un algorithme d'IA peuvent être sous différentes formes :

- Des données quantitatives continues qui peuvent prendre n'importe quelles valeurs dans un ensemble de valeurs (par exemple la température).
- Des données quantitatives discrètes qui peuvent prendre un nombre limité de valeurs dans un ensemble de valeurs (par exemple nombre de pièces d'un logement).
- Des données qualitatives (ordinales ou nominales suivant qu'on puisse les classer ou non, par exemple des couleurs, des notes à un test d'opinion, ...).

Pour pouvoir traiter ces données, il peut être nécessaire qu'elles soient organisées sous une certaine forme. On peut par exemple identifier :

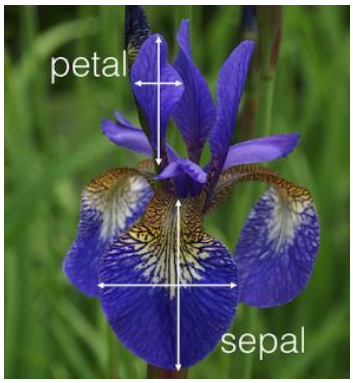
- Les données structurées (dans une base de données).
- Les données semi-structurées (dans un fichier .csv ou xml).
- Les données non structurées (image, texte ou vidéo).

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 chacun des observations.

Exemple :

Dans le cas d'une base de données regroupant des photos d'Iris, les données sont composées de 150 observations, chacune composée de 4 caractéristiques : longueur et largeur du sépale ainsi que longueur et largeur du pétale.



Définition de Big Data :

Mégadonnées ou données massives. Le big data désigne les ressources d'informations dont les caractéristiques en termes de volume, de vélocité et de variété imposent l'utilisation de technologies et de méthodes analytiques particulières pour générer de la valeur, et qui dépassent en général les capacités d'une seule et unique machine et nécessitent des traitements parallélisés.

Que fait-on avec ces données ?

En général les algorithmes vont chercher un lien entre ces données. Dans le cas où il existe un *lien connu* par le *Data Scientist* on parle *d'apprentissage supervisé*. Si ce *lien n'est pas connu*, on parle d'*apprentissage non supervisé* ou de *clustering*.

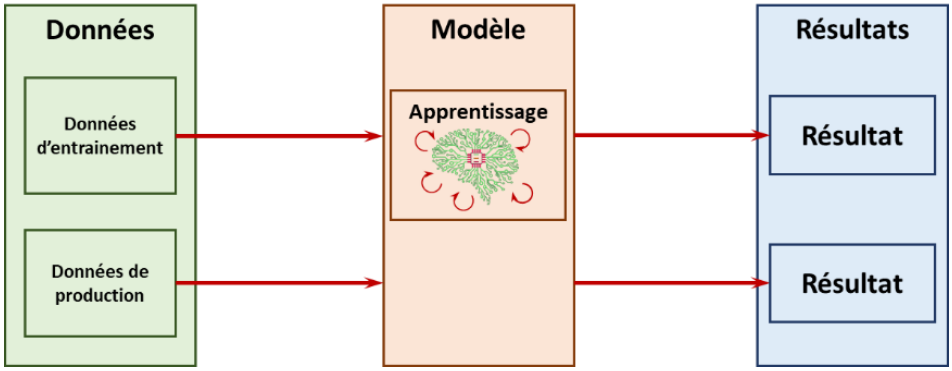
1.2. Apprentissage Automatisé – Machine Learning

Définition de l'apprentissage automatique (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.

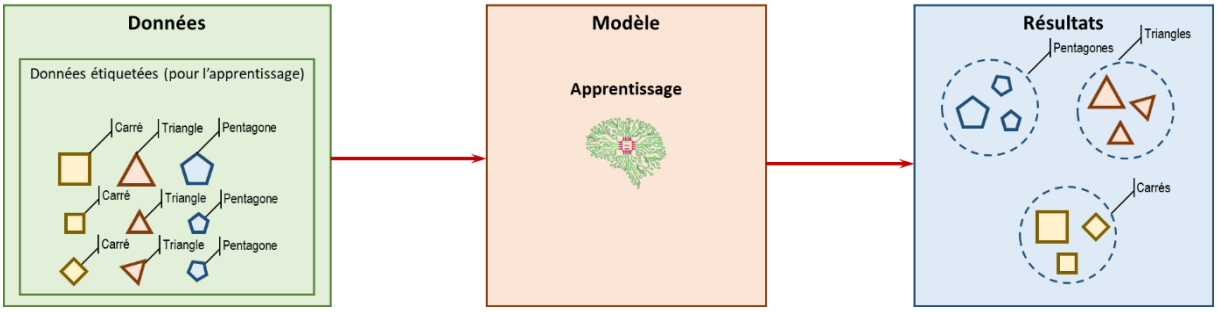
L'apprentissage comprend deux phases :

- L'entraînement (ou apprentissage) est une phase d'estimation du modèle à partir de données d'observations.
- La mise en production du modèle (inférence) est une phase pendant laquelle de nouvelles données sont traitées dans le but d'obtenir le résultat souhaité.



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.



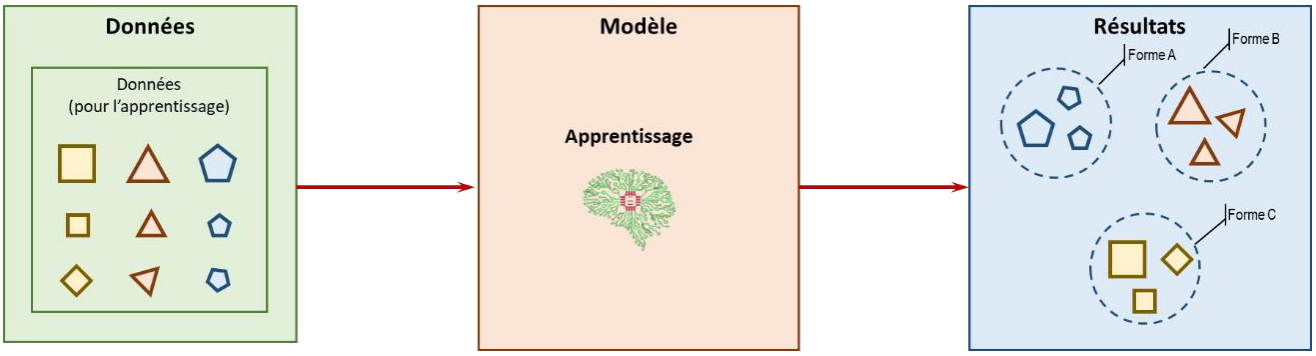
Définition de l'apprentissage supervisé - Inférence :

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

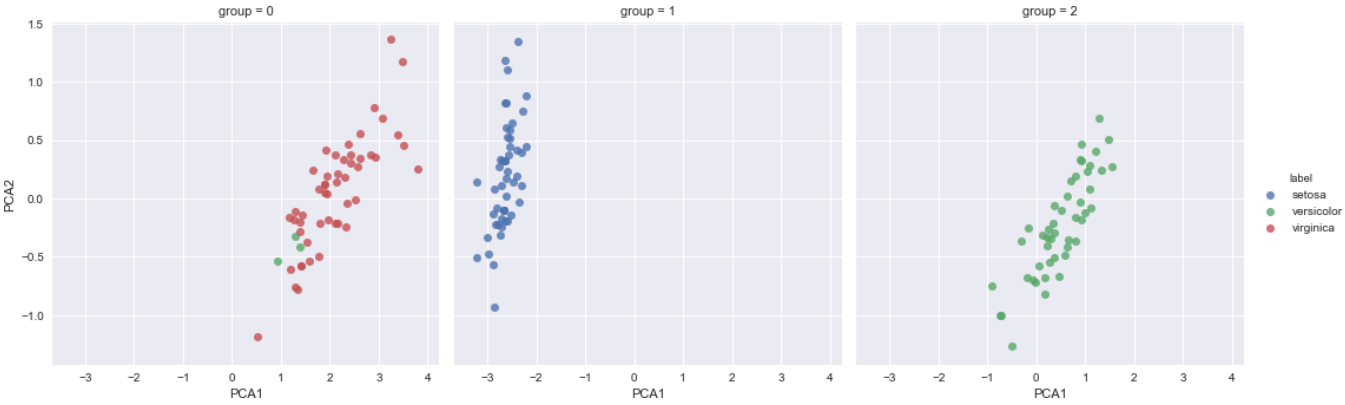
Exemple : Soit un ensemble d'images en noir et blanc représentant des chiffres de 0 à 9. L'algorithme doit dans un premier temps *apprendre* le lien entre l'image et le chiffre. Dans un second temps, l'algorithme devra déterminer le chiffre en fonction d'une image seule.

Définition de l'apprentissage non supervisé – Clustering :

Tâche d'apprentissage au cours duquel 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).



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

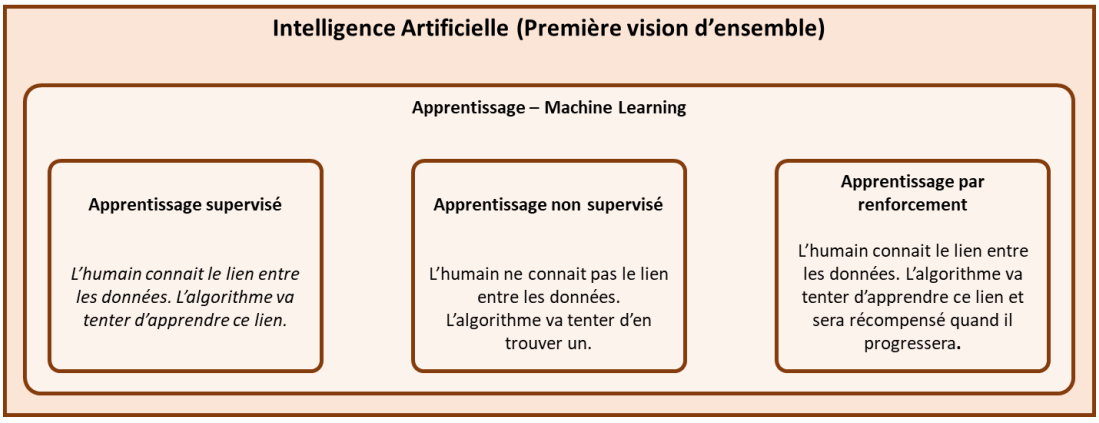


On remarque que sur 150 images d'Iris, uniquement avec les mesures des pétales, l'algorithme a réussi à retrouver les 3 différentes espèces, sans les connaître, à 3 erreurs près.

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.

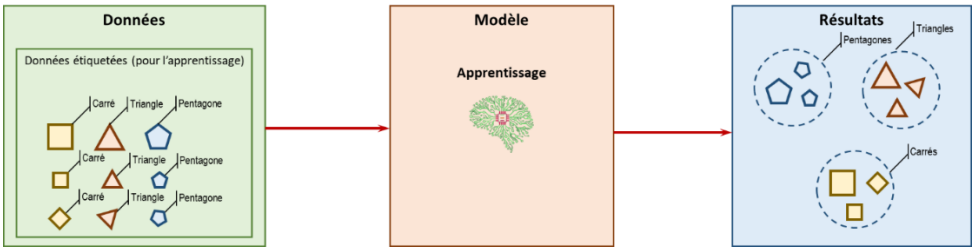
On peut donc faire une synthèse des méthodes d'apprentissage dont nous avons pris connaissance (il en existe d'autres).



1.3. Autres définitions

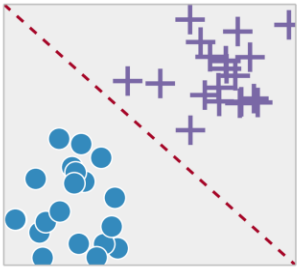
Définition de la classification :

En apprentissage automatique, on appelle problème de classification les problèmes où les étiquettes sont **discrètes**.



Dans le cas où nous souhaitons regrouper par forme, il s'agit d'un problème de classification. Les classes discrètes sont les formes (triangles, carrés, pentagones ...). Il serait aussi possible d'opérer un regroupement par couleur. Les classes seraient donc différentes.

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

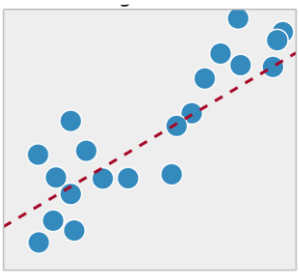


Remarque : Outre les réseaux de neurones, la méthode des k **plus proches voisins** est le seul algorithme de classification au programme en SII en PSI.

Définition d'une régression :

En apprentissage automatique, on appelle problème de régression les problèmes où les étiquettes sont **continues**.

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

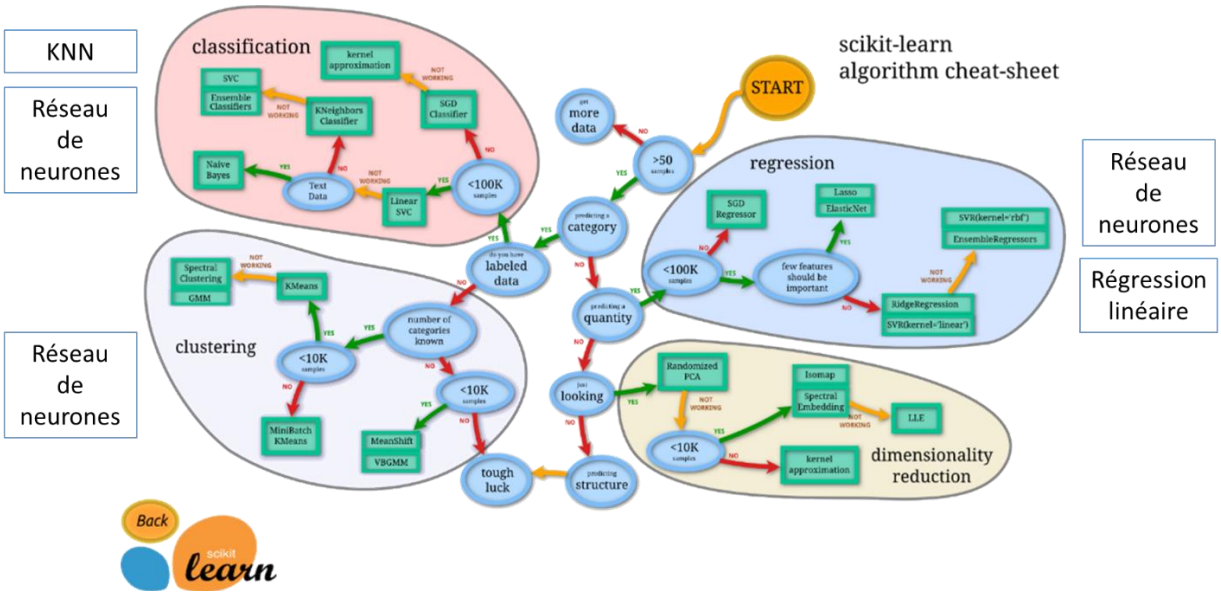


Remarque : Outre les **réseaux de neurones**, la **régression linéaire simple** et la **régression linéaire multiple** sont les seuls algorithmes de régression abordés en SII en PSI.

2. Mécanismes d'apprentissages

2.1. Classifications des algorithmes d'apprentissage

La bibliothèque *scikit-learn* propose une classification de divers algorithmes en fonction des apprentissages automatiques (hors réseaux de neurones).



2.2. Validation du modèle

Une fois un algorithme d'apprentissage choisi, on se pose la question de la validation du modèle.

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

Lors d'un problème de classification, il est par exemple possible de déterminer les écarts entre les valeurs prédites par l'algorithme et les valeurs cibles.

2.2.1. Critères de validation des problèmes de classification

Définition de la valeur prédictive positive :

Valeur prédictive positive (*accuracy classification score*) :

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

Définition des matrices de confusion :

La **matrice de confusion** est un indicateur de performance utilisé pour les problèmes de **classification**. Elle compare les données réelles à celles prédites par un modèle. En abscisses sont indiquées les valeurs prédites, et en ordonnées les valeurs réelles. Cet outil, très visuel, permet de distinguer rapidement les erreurs du modèle.

Pour problème de classification binaire, comportant deux catégories :

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

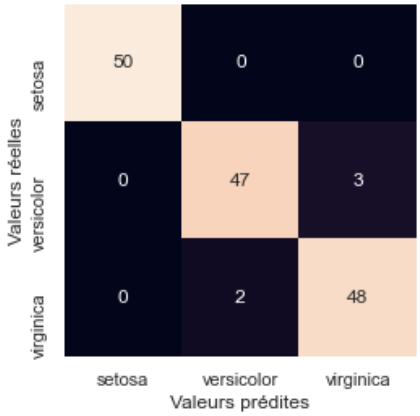
Lorsque le nombre de catégories est supérieur à deux, le calcul est fait pour chaque catégorie. Si la répartition des étiquettes n'est pas uniforme (en fonction des classes), il est probable qu'il y ait davantage d'erreurs pour les classes ayant beaucoup d'étiquettes. Pour pallier ce problème on peut utiliser des **matrices de confusions normalisées** (on divise alors chaque terme de la matrice par la somme des éléments de la même ligne).

En Python, le module *sklearn.metrics* dispose d'une fonction *confusion_matrix(y_true,y_pred).*

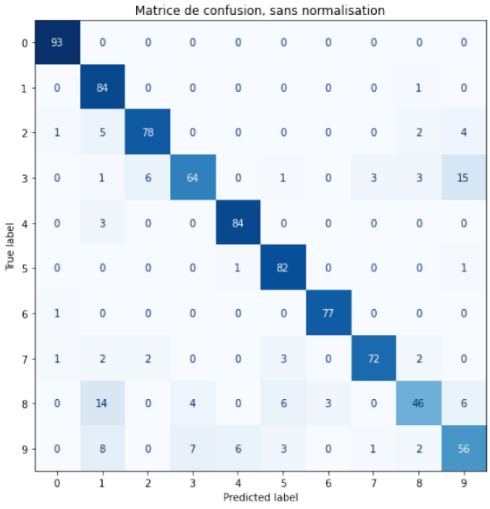
Exemple : Dans l'exemple ci-contre, nous cherchons à classifier des Iris, grâce à un algorithme, selon 3 familles : les setosa, les versicolor et les virginica.

Sur la diagonale, sont retrouvées les iris dont l'espèce a été correctement prédite.

En revanche, 3 versicolor ont été classées parmi les virginica et 2 virginica ont été classifiées dans les versicolor.



Exemple : Matrice de confusion pour la tâche de reconnaissance d'un chiffre manuscrit. La matrice de confusion permet de voir que les erreurs de prédiction sont concentrées sur les classes 3, 8 et 9 et que les confusions les plus fréquentes de la fonction de prédictions sont sur les couples 8-1, 9-1 3-9 et 8-9 (certainement dû à la présence de boucles dans le tracé des chiffres aux mêmes endroits dans l'image).



2.2.2. Critères de validation des problèmes de régression

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

L'erreur moyenne quadratique est une moyenne d'écart au carré :

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

en notant Yi la valeur réelle (étiquetée) et $\hat{Y}i$ la valeur estimée par l'algorithme.

La qualité d'un modèle est mesurée en comparant les prédictions effectuées sur un ensemble de données de taille N qui ne sont pas dans la base de données d'entraînement appelées données de test.

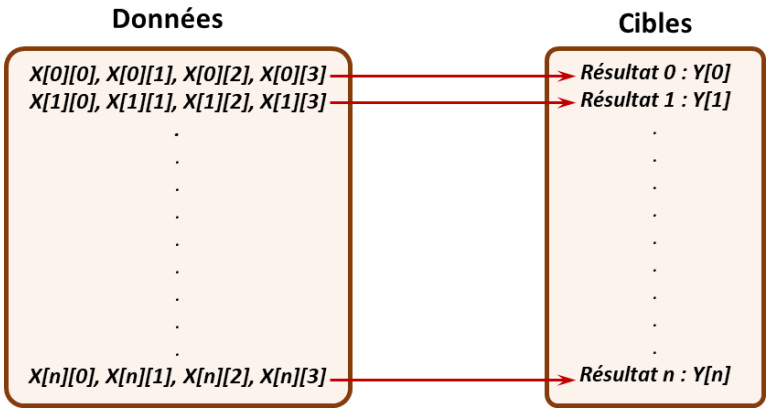
2.3. Séparation des données / Gestion des données ?

Lorsque l'on souhaite disposer d'un modèle défini par un algorithme d'IA, il faut en premier lieu disposer de données.

2.3.1. Types de données – Apprentissage supervisé

En apprentissage supervisé, il est nécessaire de connaître des données d'entrées ainsi que les sorties correspondantes (appelées aussi *cibles* ou *target*).

Données d'entrées : matrice X de n observations avec 4 caractéristiques.
Données de sortie : vecteur Y de n résultats.



Dans le cadre d'une classification, où r résultats sont possibles. On peut attribuer une valeur (entière) entre 1 et r à chacun des résultats, notamment si ceux-ci sont des données qualitatives.

Remarque : Selon la nature des sorties, on distingue plusieurs grandes classes de problèmes.

- Si les sorties y_i sont des **scalaires réels** $\forall i \in \{1, \dots, n\}\ y_i \in \mathbb{R}$, on parle alors de problème de **régression simple (ou monovariable).**
- Si les sorties y_i sont des **vecteurs de réels** (de dimension p finie) $\forall i \in \{1, \dots, n\}\ y_i \in \mathbb{R}^p$, on parle alors de problème de **régression multiple (ou multivariable).**

2.3.2. Types de données – Apprentissage non-supervisé

L'apprentissage non-supervisé est caractérisé par l'absence de sortie y fournie lors de l'apprentissage du modèle prédictif.
Pourtant on construit une fonction $f: X \to Y$ mais qui au lieu de reproduire le comportement observé sur des exemples, doit cette fois produire des sorties satisfaisant certains critères.

Exemple : On peut souhaiter prédire la partie manquante d'une donnée : les données futures pour une série temporelle ou remplacer les pixels flous d'une image par la vraie image nette.

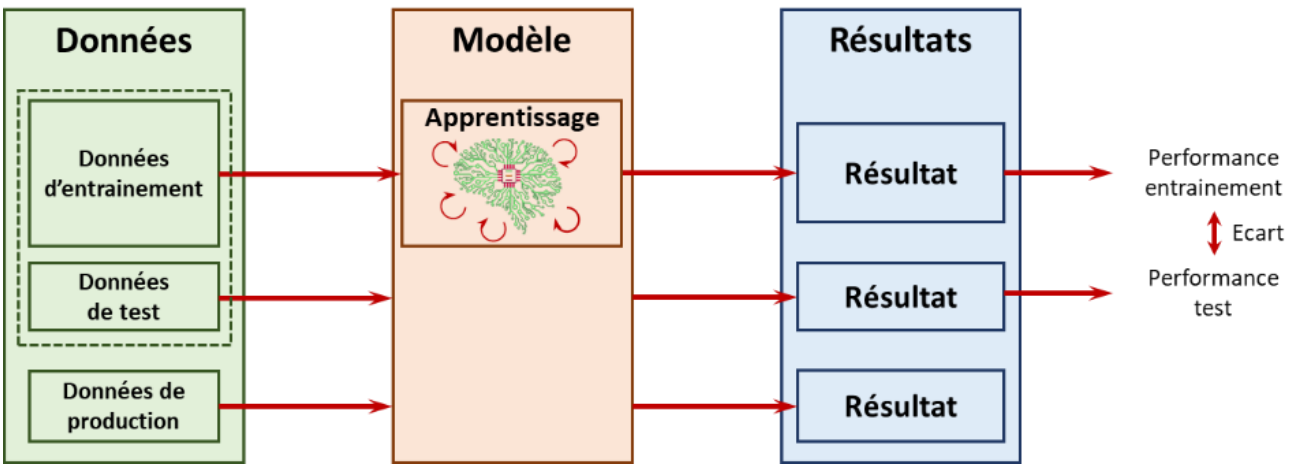
2.3.3. Séparation des données

Lors du démarrage d'un apprentissage supervisé, on dispose d'un jeu de données comprenant les données d'entrée et les cibles correspondantes. Il est d'usage de séparer ces données en deux parties :

- Les données d'entraînement permettant d'entraîner le modèle.
  Dans la pratique on utilise entre 60 et 80% des données de base.
- Les données de test, permettant de valider l'apprentissage (ou le modèle).
  Dans la pratique entre 20 et 40% des données de base.

On commence donc par réaliser un apprentissage sur les données d'entraînement. Cet entraînement produit des résultats. Connaissant les cibles correspondant aux données d'entraînement, on peut donc en déduire une performance du modèle sur le traitement des données d'entraînement.

On utilise alors le modèle en lui donnant les données de test. De même, on peut donc en déduire une performance du modèle sur le traitement des données de test.



On perçoit donc que la qualité de l'apprentissage peut dépendre du choix utilisé lors de la séparation des données.

De plus, certains choix de paramètres permettant d'affiner le modèle sont aussi liés aux données d'entraînement sélectionnées. Une des solutions pour résoudre ce problème est d'avoir recours à la validation croisée.

3. Algorithmes d'apprentissage

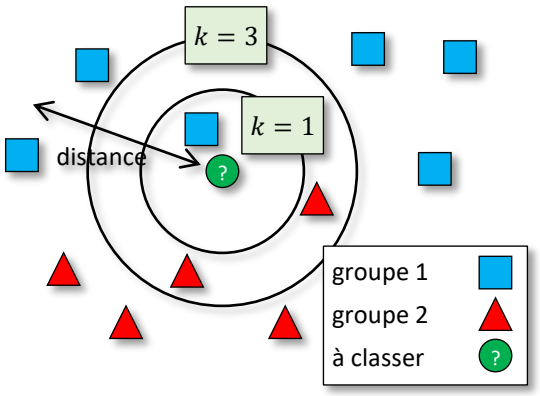
3.1. Algorithme des k plus proches voisins (KNN – K Nearest Neighbors)

3.1.1. Présentation

L'algorithme des *k* plus proches voisins (aussi appelé *k-nearest neighbors algorithm* ou *kNN*) permet de réaliser des opérations de régression et de classification sur un ensemble de données.

Pour une première approche, cette méthode permet d'estimer la classe d'une nouvelle donnée en déterminant la norme entre cette nouvelle donnée et l'ensemble des données du jeu d'entraînement.

Parmi les *k* plus proches voisins, on recherche la classe majoritaire et on attribue cette classe à la nouvelle donnée.



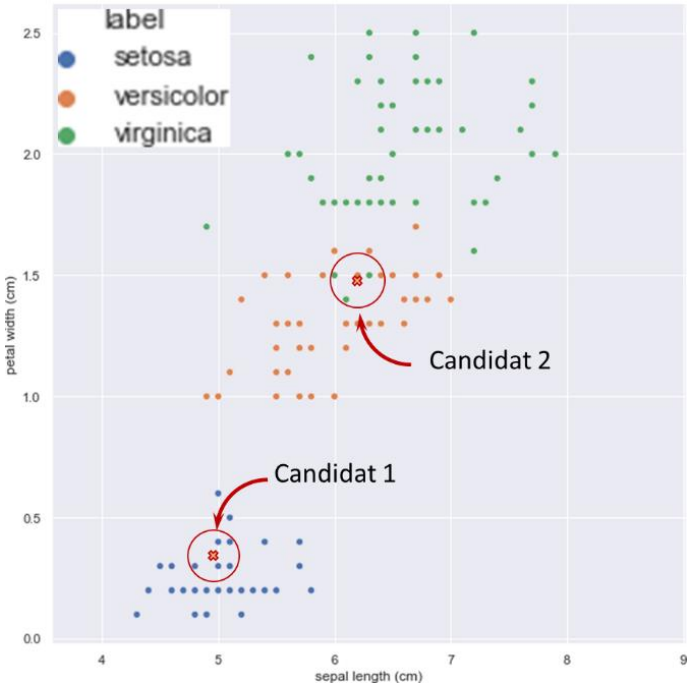
Exemple : Dans le cas ci-contre, on cherche à classifier les Iris selon 3 catégories (setosa, versicolor et virginca).

Les données étiquetées sont représentées dans le plan longueur du sépale en abscisse et largeur du pétale en ordonnée.

Soient les candidats 1 et 2, deux Iris dont on connaît les caractéristiques mais pas l'espèce.

Les 5 plus proches voisins du candidat 1 sont des setosa. Il est donc probable que le candidat 1 soit un setosa aussi.

Concernant le candidat 2, parmi les 5 plus proches voisins, 3 sont des virginica et 2 sont des versicolor. On peut faire l'hypothèse que c'est un virginica.



3.1.2. L'algorithme

L'algorithme des ***k* plus proches voisins** est un algorithme

- **d'apprentissage supervisé** nécessitant l'utilisation d'un jeu de données d'apprentissage, $D = \{x_i, y_i\}$
- **non paramétrique** : le modèle ne dispose pas de paramètres dont la valeur serait à optimiser
- de **classification** : les sorties y_i correspondent aux catégories ou étiquettes possibles

Prenons comme exemple des données sous la forme *[x,y,c]* où *x* et *y* représentent des coordonnées et *c* la classe d'appartenance.

Première étape : Calculer la distance :

Il faut tout d'abord définir une fonction de distance. On peut par exemple utiliser la distance euclidienne.

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

```
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)
```

Deuxième étape : Détermination des plus proches voisins :

Pour cette étape, on dispose de données pour l'entraînement *data_train* et d'une donnée à tester *data_test*.
Il va falloir calculer la distance entre la donnée à tester et les données d'entraînement puis trier les données. Enfin, on retournera 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
```

Troisième étape : Prédiction :

Une fois qu'on connaît les *k* plus proches voisins, on regarde 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 : En Python, la bibliothèque *sklearn* dispose d'un module *neighbors*.

3.1.3. Applications et limites

Cet algorithme est relativement robuste, si suffisamment d'exemples sont fournis dans les données d'apprentissage. Il est, de plus, facilement implémentable dans différents langages de programmation.

Le choix de la valeur de l'**hyperparamètre** k a une forte influence sur la prédiction faite par l'algorithme :

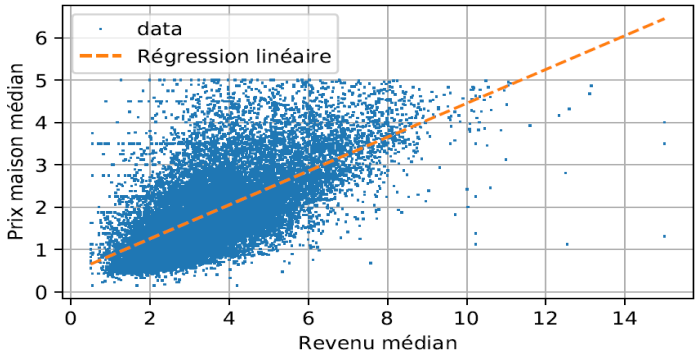
- si k est trop petit, des éléments sortant de l'ordinaire risquent d'influer la prédiction. La généralisation du modèle pour de nouveaux éléments est alors moins bonne, il s'agit du problème de **sur-apprentissage** ;
- si k est trop grand, le modèle prend en compte des données trop éloignées et la catégorie majoritaire dans le jeu de données est alors prédite trop souvent. Il s'agit cette fois du problème de **sous-apprentissage**, le modèle n'utilise que trop peu les données d'apprentissage.

Remarque : **Autres utilisations des k plus proches voisins :** Si, au lieu de faire un vote, on fait une moyenne des variables cibles pour les différents voisins, on obtient alors un outil de régression au lieu d'un outil de classification.

3.2. Régressions

3.2.1. Régression mono variable – Régression linéaire simple

La régression linéaire est un algorithme d'apprentissage supervisé, les paramètres du modèle sont estimés à partir d'un jeu de données d'apprentissage, $D = \{x_i, y_i\}$, permettant l'entrainement du modèle.



Dans cette section, on identifiera l'espace de sortie Y à $\mathbb{R}$. La régression linéaire consiste à choisir des fonctions de prédiction f de la forme :

$$\forall x \in \mathbb{R}\ f(x) = a.x + b \qquad \text{où } a \text{ et b sont des constantes réelles}$$

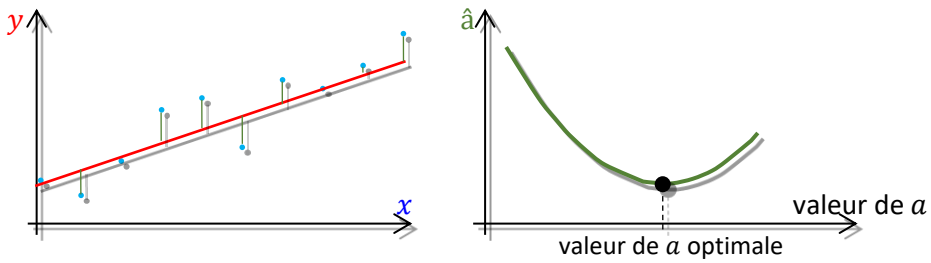
Dans le cadre de la régression linéaire, la fonction de coût L communément utilisée est l'erreur quadratique. La fonction de prédiction optimale est celle qui minimise l'erreur sur la base de données d'entraînement :

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

Sous l'hypothèse d'une solution correspondant à une fonction linéaire de la forme de l'équation $\forall x \in \mathbb{R}\ f(x) = a.x + b$, la constante $\hat{a}$ est obtenue en résolvant le problème de minimisation :

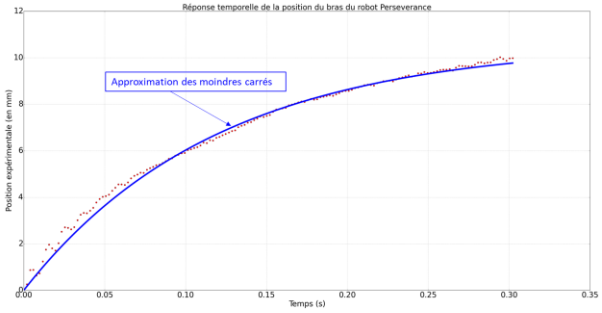
$$\hat{a} = \min_a \frac{1}{n} \sum_{i=1}^{n} ((a.x_i + b) - y_i)^2$$

Cette fonction, appelée **coût**, correspond à **minimiser la somme des écarts au carré entre les prédictions et les valeurs attendues**.



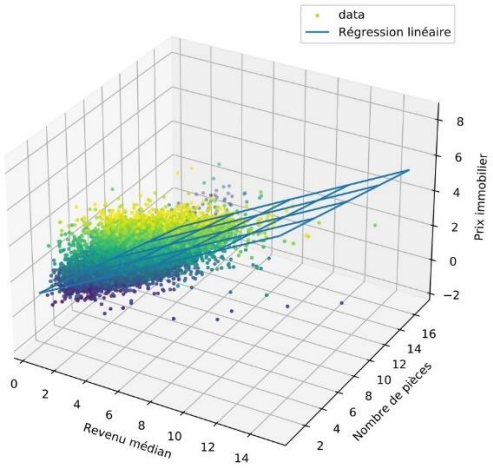
Remarque : En langage Python, la fonction *linregress()* du module *scipy.stats* et le module *linear_model* de la bibliothèque *sklearn* peuvent notamment être utilisés.

Remarque : Une analogie peut être faite avec le TD de SUP en Automatique (TD 4 : Robot Perseverance, **voir sur site du professeur**) dans lequel on souhaite faire une régression linéaire aux moindres carrés pour retrouver les paramètres caractéristiques du système du premier ordre.



3.2.2. Régression multiple ou multi-variables

Une régression linéaire multiple, ou multivariable, est une généralisation de la méthode précédente lorsque les données d'entrée ont une dimension strictement supérieure à 1.



On peut étendre l'étude de la régression au cas où les entrées x_i sont des vecteurs de $\mathbb{R}^d$ et les sorties y_i des vecteurs de $\mathbb{R}^p$. Le modèle linéaire liant les entrées x et les sorties y suppose l'existence d'un vecteur de paramètres $A \in \mathbb{R}^{d.p}$ tel que $y = x^T A$

Ce vecteur est calculé en minimisant l'erreur de prédiction quadratique sur la base de données d'entraînement. Ainsi si on dispose d'une base de données $(x_i, y_i)_{i\in\{1,\dots,n\}}$, on peut calculer la matrice $\hat{A}$ correspondant au 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 $\min_{A\in\mathbb{R}^{d.p}} f(A)$ représente la valeur prise par A lorsque $f(A)$ atteint son minimum.

Soient $Y = \begin{pmatrix} y_1^T \\ y_2^T \\ \dots \\ y_n^T \end{pmatrix} \in \mathbb{R}^{n.p}$ et X$= \begin{pmatrix} x_1^T \\ x_2^T \\ \dots \\ x_n^T \end{pmatrix} \in \mathbb{R}^{n.d}$ On peut montrer que : $\hat{A} = \min_{A\in\mathbb{R}^{d.p}} \frac{1}{n} \|Y - XA\|^2$

La solution de ce problème a pour expression : $\hat{A} = (X^T X)^{-1} X^T Y$

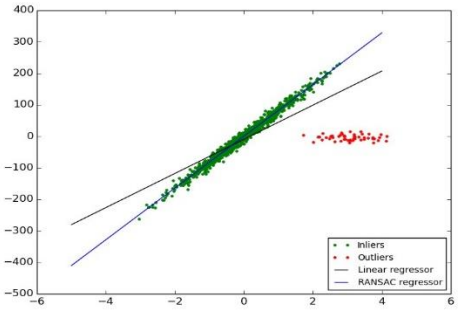
Remarque : En python, les tableaux du module *numpy*, *numpy.array*, sont particulièrement adaptés au calcul matriciel. Le module *linear_model* de la bibliothèque *sklearn* est adapté à la régression linéaire multiple.

3.2.3. Applications et limites

Les modèles de régression linéaire ou multivariables sont facilement et rapidement implémentables.

Toutefois, les modèles de régression présentent de nombreuses limites :

- Domaine d'applications très restreint par la linéarité du modèle. L'extension du modèle à des régressions polynomiales élargie le champ d'application, mais complexifie le modèle et le rend sensible aux problèmes de **sur-apprentissage** et **sous-apprentissage**.
- Grand nombre de données est nécessaire à la phase d'apprentissage, comme pour tout modèle d'apprentissage supervisé.
- Estimation sensible au bruit de mesure présent dans le jeu de données d'apprentissage.
- Outil d'analyse très sensible à la présence de données aberrantes (*outliers*).



Remarque : RANSAC (RANdom SAmple Consensus) = Méthode d'estimation mathématique et itérative utilisée lorsque l'ensemble de données observées peut contenir des valeurs aberrantes (outliers).

3.3. Limites de l'apprentissage automatique

L'apprentissage automatique, incluant l'apprentissage profond, est flexible et permet de résoudre des problèmes complexes. Toutefois, il présente des limites dont il faut avoir conscience :

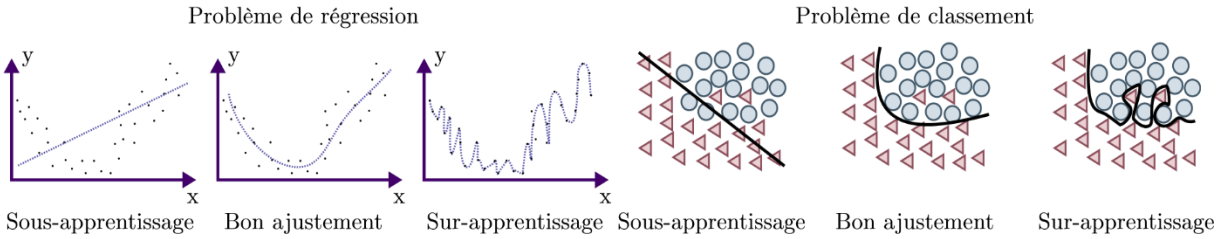
- Une **probabilité d'erreur non nulle**, la précision des prédictions n'étant jamais de 100 %.
- Des **algorithmes opaques**, dont le fonctionnement est difficile à appréhender. Même en connaissant la structure du programme, les étapes d'exécution ne sont pas explicites.
- Des **erreurs difficiles à détecter et corriger** en raison de l'opacité des algorithmes, ce qui rend le développement et l'entraînement du modèle prédictif complexe.
- Des algorithmes de **complexité exponentielle** dans la plupart des cas. Des approximations sont parfois nécessaires pour améliorer l'efficacité, toutefois la précision est alors détériorée.
- Une **forte dépendance entre les performances du modèle prédictif et les données d'apprentissage**, qui doivent être suffisamment représentatives.
- Le « fléau de la dimension » : le nombre de données d'apprentissage nécessaire au bon fonctionnement du modèle prédictif croît fortement avec la dimension du problème. Des méthodes sont parfois utilisées pour réduire la dimension.

Remarque : En raison de la complexité exponentielle des algorithmes et de la quantité colossale de données nécessaires à la phase d'apprentissage, l'utilisation d'une méthode d'apprentissage automatique peut avoir de forts impacts environnementaux, notamment la consommation énergétique ou l'empreinte carbone. Des études visant à réduire ces impacts sont conduites en parallèle des recherches sur l'IA elle-même.

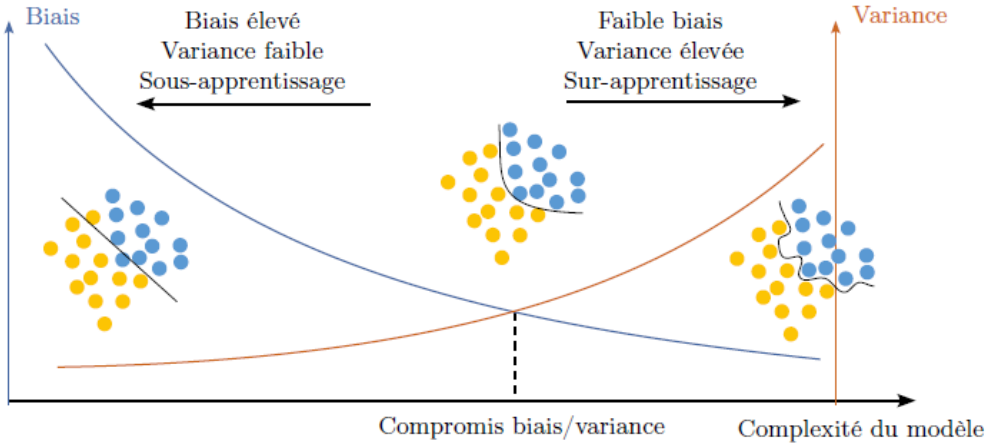
3.4. Le danger du surapprentissage – Dilemme biais-variance

Le problème avec l'élaboration d'un modèle à partir de données préétablies, est qu'il est toujours possible d'obtenir un modèle qui ressorte les données apprises sans avoir su généraliser intelligemment un modèle à partir des données. On parle alors de surapprentissage (*overfitting*).

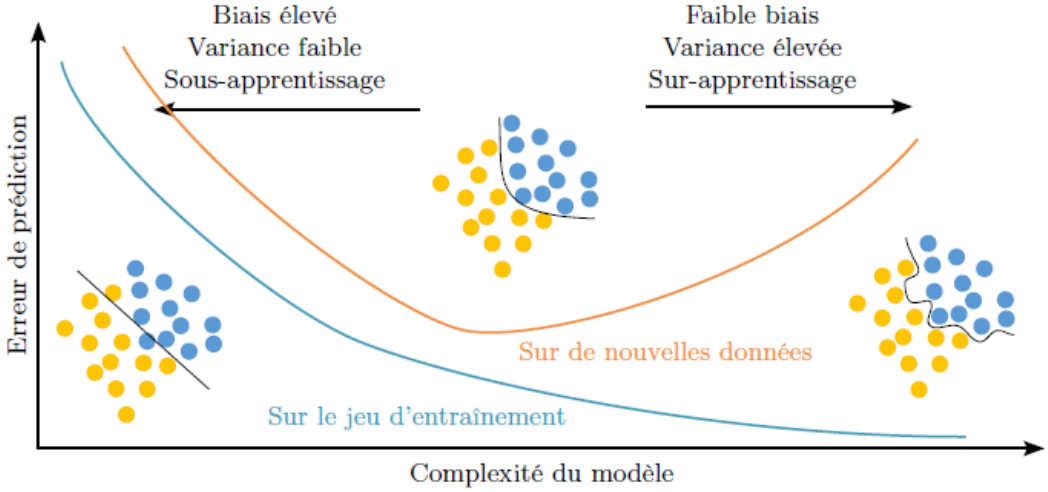
Ce risque apparaît quand le modèle colle trop aux données d'apprentissage : on surévalue la confiance que l'on apporte aux données sans considérer les incertitudes et les bruits de mesures. On parle de dilemme (ou de compromis) biais-variance.



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



Ainsi, un modèle complexe sera très bon sur les enregistrements utilisés pour l'apprentissage mais pourra avoir des performances médiocres sur de nouvelles données.



4. Les réseaux de neurones

3.5. Bref historique

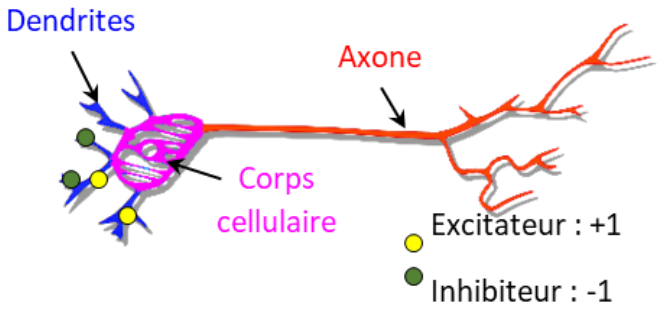
Dans les années 40, les chercheurs tentent de fabriquer une machine capable d'apprendre à partir de données fournies, de mémoriser des informations et de traiter des informations incomplètes. Pour cela, ils essayent de réaliser des modèles mathématiques de neurones biologiques. S'en suit alors la naissance des premiers modèles de neurones. L'apprentissage automatisé va alors connaître des hauts et des bas, au gré des avancées scientifiques et technologiques. Dans les années 60, un des premiers coups d'arrêt fût provoqué par la non-capacité des réseaux de neurones à traiter des problèmes non linéaires. Dans les années 80, la rétropropagation du gradient fut proposée. Mais devant le manque de capacité des ordinateurs, la recherche marquât un second coup d'arrêt. Dans les années 90/2000, l'apparition des réseaux convolutifs et leur capacité à analyser les données des images relançât alors les recherches dans ce domaine.

3.6. 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.

5. 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.

Définition d'un neurone (ou perceptron) :

Prenons la représentation suivante pour un neurone.

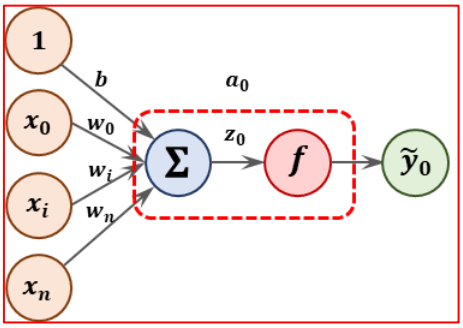
On note :

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

On a donc, dans un premier temps :

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

Après la fonction d'activation, on a donc en sortie du neurone : $\tilde{y}_0 = f(z_0) = f(b + \sum_{i=0}^{n} \omega_i x_i)$



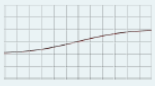
Remarque : La notation tilde ($\tilde{y}_0$) vient du fait que la valeur de sortie d'un neurone est une valeur estimée qu'il faudra comparer à y_0 valeur de l'étiquette utilisée pour l'apprentissage supervisé.

Remarque : Par la suite, dans la représentation graphique on ne fera pas apparaître la somme pondérée et la fonction d'activation, mais seulement la valeur de sortie du neurone (a_0).

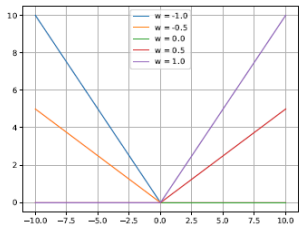
Remarque : Généralement, la fonction d'activation est choisie par le concepteur, puis les paramètres ω et b sont ajustés par une règle d'apprentissage de sorte que la relation entrée/sortie du neurone réponde à un objectif spécifique.

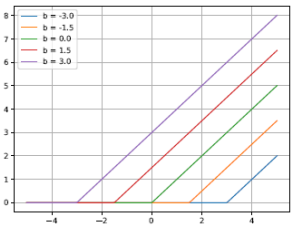
Définition d'une fonction d'activation :

Les fonctions d'activation sont des fonctions mathématiques appliquées au signal de sortie (z). Il est alors possible d'ajouter des non linéarités à la somme pondérée. On donne ci-dessous quelques fonctions usuelles :

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

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

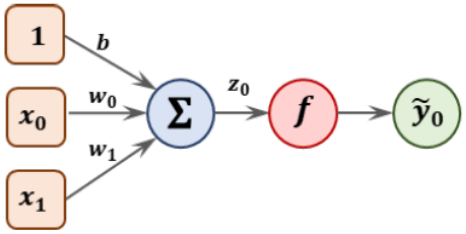




On peut ainsi voir qu'avec la fonction d'activation *ReLU*, plus le poids sera grand en valeur absolu, plus le neurone amplifiera le signal d'entrée. Le biais permettra de prendre en compte le « niveau » du signal d'entrée à partir duquel, le signal doit être amplifié, ou non.

Exemple : Prenons un neurone à deux entrées binaires.

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



On peut donc évaluer l'ensemble des sorties calculable par le neurone.

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

6. Réseaux de neurones

5.1. Modélisation d'un réseau de neurones

Généralement, un seul neurone, même avec de nombreuses entrées, n'est pas suffisant. Il peut en falloir cinq ou dix, fonctionnant en parallèle, dans ce que l'on appelle une couche.

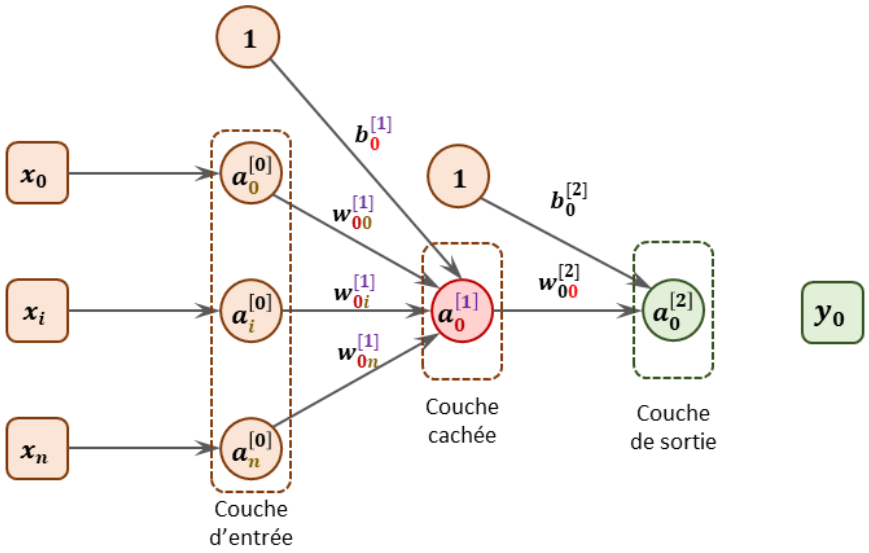
Définitions des couches :

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 $i + 1$.

- ✓ **Couche d'entrée :** cette couche est une copie de l'ensemble des données d'entrées. Le nombre de neurones de cette couche correspond donc aux nombres de données d'entrées. On note **X** = $(x_0, x_1, \ldots, x_n)$ le vecteur d'entrées.

- ✓ **Couche cachée (ou couche intermédiaire) :** il s'agit d'une couche qui a une utilité intrinsèque au réseau de neurones. Ajouter des neurones dans cette couche (ou ces couches) permet donc d'ajouter de nouveaux paramètres. Pour une couche, la même fonction d'activation est utilisée pour tous les neurones. En revanche la fonction d'activation utilisée peut être différente pour deux couches différentes. Les fonctions d'activations des couches intermédiaires sont souvent non linéaires.

- ✓ **Couche de sortie :** le nombre de neurones de cette couche correspond au nombre de sorties attendues. La fonction d'activation de la couche de sortie est souvent linéaire. On note **Y** = $(y_0, y_1, \dots, y_m)$ le vecteur des sorties.



En utilisant la loi de comportement du modèle de neurone, on peut donc exprimer **Y** = F (**X**) où F est une fonction dépendant des entrées, des poids et des biais.

Notations :

- $\omega_{jk}^{[l]}$: poids permettant d'aller vers la couche l depuis le neurone *k* vers le neurone *j*
- $b_j^{[l]}$: biais permettant d'aller sur le neurone *j* de la couche l
- $f^{[l]}$: fonction d'activation de la couche l
- $n^{[l]}$: nombre de neurones de la couche l

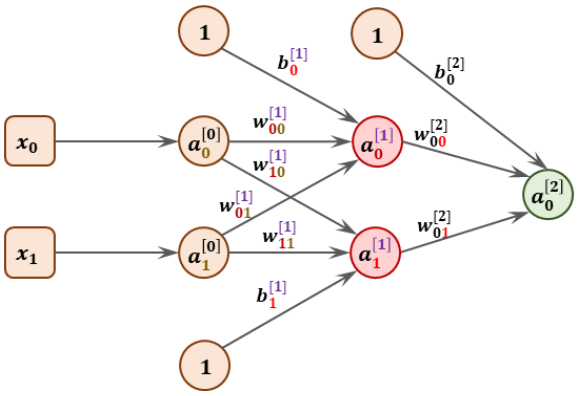
Remarque : Chaque couche a sa propre matrice de poids, son propre vecteur de biais, un vecteur d'entrée et un vecteur de sortie.

Définition de l'équation de propagation :

Pour chacun des neurones $a_j^{[l]}$ on peut donc écrire l'équation de propagation qui lui est associé :

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

Exemple :



Prenons un réseau de neurones 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 : $\omega_0 = -0{,}3 \quad \omega_1 = 0{,}8 \;\; et \; b = 0{,}2.$
Il est possible d'écrire que : $y_0 = a_0^{[2]} = f_2(b_0^{[2]} + \omega_{00}^{[2]}.a_0^{[1]} + \omega_{01}^{[2]}.a_1^{[1]})$

Par ailleurs : $a_0^{[1]} = f_1(b_0^{[1]} + \omega_{00}^{[1]}.a_0^{[0]} + \omega_{01}^{[1]}.a_1^{[0]})$ et $a_1^{[1]} = f_1(b_1^{[1]} + \omega_{10}^{[1]}.a_0^{[0]} + \omega_{11}^{[1]}.a_1^{[0]})$

Au final, on a donc :

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

Définitions des paramètres :

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

Les poids ω_i et le biais b pour chaque neurone seront déterminés à partir des données d'entrainement en utilisant par exemple une **méthode de descente du gradient**.

Méthode : Calcul du nombre de paramètres :

Soit un jeu de données étiquetées avec *n* entrées et *p* sorties.
On construit un réseau possédant l couches et a_l le nombre de neurones de la couche l.
Dans ce cas, la première couche est la couche d'entrée ($a_1 = n$) et la dernière couche et la couche de sortie ($a_l = p$).

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

Au final, le nombre total de paramètre à calculer est donné par $N = n_\omega + n_b$.

Objectif : Soit un jeu de données étiquetées. On note **X** le vecteur des données d'entrées. On note **Y** le vecteur des données de sorties. On note $\tilde{Y}$ le vecteur de sortie calculé par le réseau de neurones. L'objectif de la phase d'apprentissage du réseau de neurones est de déterminer les valeurs de l'ensemble des poids et des biais de telle sorte que l'écart entre **Y** et $\tilde{Y}$ soit minimale.

5.2. Etapes préliminaires à l'entrainement

Un certain nombre d'étapes doivent être réalisées avant l'entrainement du réseau. Elles peuvent être regroupés en trois catégories :

- Sélection des données.
- Prétraitement des données.
- Choix du type de réseau (imposé ici par le type de problème) et de son architecture.

5.2.1. Sélection des données

Il est généralement difficile d'intégrer des connaissances préalables dans un réseau de neurones, par conséquent, la qualité du réseau sera fonction de la qualité des données utilisées pour l'entraîner. Les réseaux de neurones représentent une technologie qui est à la merci des données. Les données relatives à l'entrainement doivent couvrir l'ensemble de l'espace des entrées d'utilisation du réseau.

Une dernière question que nous devons nous poser sur la sélection des données est « *avons-nous suffisamment de données ?* »

Il est difficile de répondre à cette question, surtout avant d'entrainer le réseau. La quantité de données requises dépend de la complexité de la fonction que nous essayons d'approximer. C'est pourquoi l'ensemble du processus d'entrainement du réseau neuronal est itératif. À l'issue de l'entrainement, nous analyserons la performance du réseau. Les résultats de cette analyse peuvent nous aider à décider si nous avons suffisamment de données ou non.

5.2.2. Prétraitement des données

L'objectif principal de la phase de prétraitement des données est de faciliter l'entrainement au réseau. Le prétraitement des données comprend des étapes telles que la normalisation, les transformations non linéaires, l'extraction de caractéristiques, le traitement des données manquantes, etc… L'idée est d'effectuer des traitements des données pour faciliter l'entrainement des réseaux de neurones.

5.2.3. Choix du réseau

L'étape suivante du processus d'entrainement du réseau est le choix de l'architecture du réseau. Le type de base de l'architecture de réseau est déterminé par le type de problème que nous souhaitons résoudre. Une fois que l'architecture de base est choisie, nous devons décider de détails spécifiques tels que le nombre de neurones et de couches que l'on veut utiliser, combien de sorties le réseau devrait avoir et quel type de fonction de performance nous voulons utiliser pour l'entrainement.

5.3. Entrainement

Après la préparation des données et la mise en place de l'architecture du réseau sélectionnés, nous sommes prêts à entrainer le réseau.
Avant d'entrainer le réseau, nous devons initialiser les poids et les biais. La méthode que nous utiliserons dépendra du type de réseau. Pour les réseaux multicouches, les poids et les biais sont généralement fixés à de petites valeurs aléatoires (par exemple, répartis uniformément entre -0,5 et 0,5).
L'entrainement des réseaux de neurones est un processus itératif. Même après convergence de l'algorithme, l'analyse post-entrainement peut suggérer que le réseau soit modifié et/ou de nouveau entrainé. En outre, plusieurs cycles d'entrainement doivent être effectués pour chaque réseau potentiel afin de s'assurer qu'un minimum global ait été atteint.

5.4. Fonction de coût (*cost function* ou de perte, *loss function*)

Dans le but de minimiser l'écart entre la sortie du réseau de neurones et la valeur réelle de la sortie, on utilise une fonction coût (ou fonction de perte). Il est possible de définir plusieurs types de fonctions, notamment en fonction du type de problème à traiter (classification ou régression par exemple).

Définition de la fonction coût :

Notons n_b le nombre de données dans la base d'entraînement. Dans le cadre d'un problème de régression, on peut définir la fonction coût comme la moyenne des erreurs quadratique entre la valeur donnée par l'équation de propagation et la valeur de l'étiquette :

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

L'objectif est dès lors de déterminer les poids et les biais qui minimisent la fonction coût.

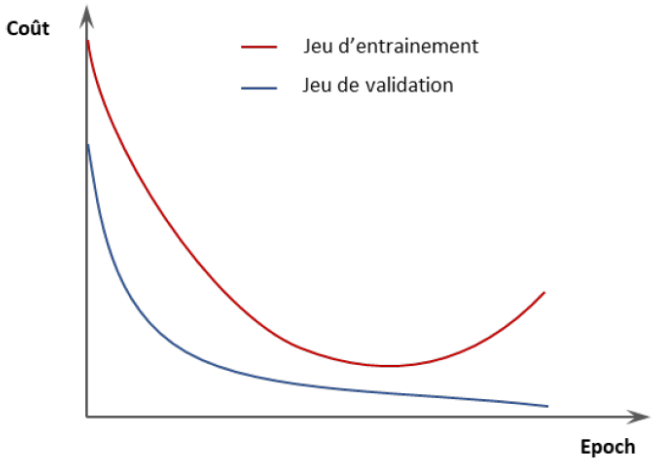
5.5. Fin d'apprentissage

Définition de l'epoch :

On appelle epoch, un 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.

Les méthodes pour stopper l'apprentissage sont essentiellement empiriques. On pourrait en effet fixer un nombre d'epoch ou une valeur d'erreur admissible et s'arrêter à ce moment-là. Dans la pratique, se focaliser sur l'erreur n'est généralement pas satisfaisant. En effet, il y a risque de « suraprentissage ».

Dans la figure ci-contre, on réalise un entraînement sur le jeu de données. A la fin de l'epoch on dispose d'un premier modèle de réseau de neurones.



On détermine alors l'erreur commise en utilisant le jeu d'entraînement puis l'erreur commise sur le jeu de validation.

On calcule ensuite l'erreur commise par le modèle sur le jeu de validation. *(On rappelle que le jeu de validation ne sert pas à modifier les poids et les biais.)*

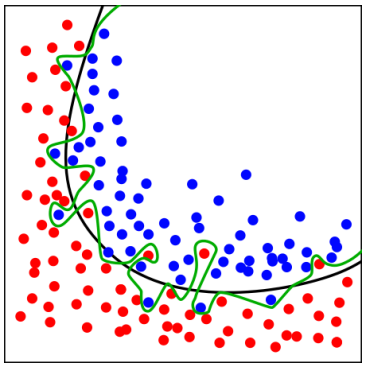
On réalise de même à la fin de l'epoch suivante etc...

« Logiquement » l'erreur décroît toujours avec le jeu d'entraînement.

Sur le jeu de validation, l'erreur décroît pendant un certain nombre d'epoch puis augmente.

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*).

Exemple : La ligne verte représente un modèle sur-appris et la ligne noire représente un modèle régulier. La ligne verte classifie trop parfaitement les données d'entraînement, elle généralise mal et donnera de mauvaises prévisions futures avec de nouvelles données. Le modèle vert est donc au final moins bon que le noir.

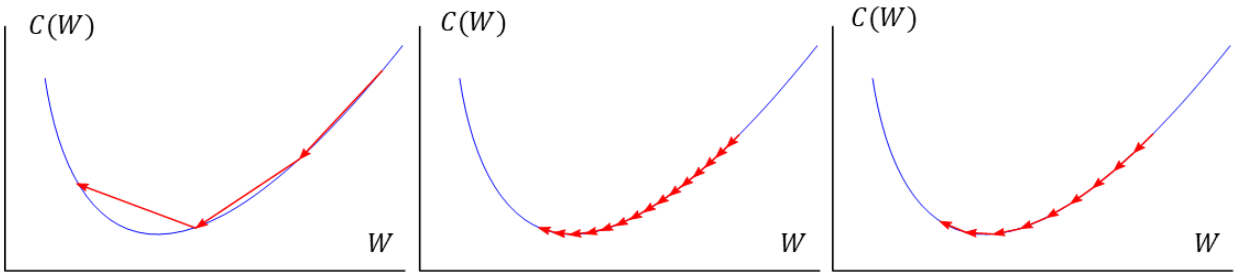


7. Minimiser la fonction le coût (cas général) : La méthode de descente de Gradient

Il peut être intéressant de mesurer la variation de la fonction coût à chaque couche du modèle et d'essayer de minimiser la valeur globale du coût. Cette étape de rétropropagation utilise la méthode de descente de gradient.

La fonction de coût est $C(W) = \sum_i^n (f(x_i) - y_i)^2$ et $W = \{w_1, w_2, \dots, w_i\}$, w_i paramètres de $f(x)$.
Dans le cas d'une fonction différentiable, se déplacer dans la direction opposée au gradient de la fonction est la méthode de « descente » la plus rapide.

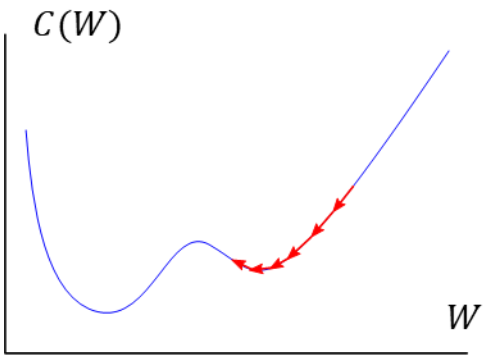
En d'autres termes, on fait varier en suivant la plus grande pente pour arriver le plus vite possible au minimum : $W_{p+1} = W_p - \alpha \nabla \mathrm{C}(\mathrm{W})$ où $\nabla \mathrm{C}(\mathrm{W})$ est la direction du gradient en W, le signe (–) donne la direction opposée au gradient et α est le facteur d'apprentissage (learning rate) dans l'algorithme de descente. Si α est trop grand, on peut rater le point le plus bas, si α est trop petit la descente est trop lente.



Le taux d'apprentissage (*learning rate*) α est un hyper-paramètre et qu'il peut faire augmenter le coût en cas de mauvais choix.

Remarque : On introduit souvent un α variable, compris entre 0 et 1. Dans la pratique on s'arrête si un certain nombre d'itération maxi est atteint, si la différence entre deux valeurs de pentes successives est inférieure à ϵ fixé ou si l'erreur absolue entre deux itérations est inférieure à un seuil fixé. Enfin il n'est pas rare que l'on ait qu'un minimum local …

L'estimation obtenue est sensible au bruit de mesure introduit dans les données d'apprentissage : une donnée aberrante influera sur les estimations des paramètres.



Exemple d'application :

On note X_l= (x_1, x_2, x_3, x_4) le vecteur de la combinaison des entrées. Les poids W et les biais B sont mis à jour par la méthode de la descente du gradient. Afin de se fixer les idées, on s'intéresse à la première couche du réseau de neurones, ayant pour entrée le vecteur X et pour sortie le vecteur d'activations A = f(W_1X + B_1). Le fonctionnement des autres couches sera identique.

Pour des raisons de lisibilité, on posera désormais : W = W1, B = B1 et donc : A = f(Y)= f(WX + B)

Les valeurs des paramètres W et B de cette couche sont mis à jour grâce aux formules suivantes :

$$\begin{cases} W = W - \alpha \frac{\partial C}{\partial W} \\ B = B - \alpha \frac{\partial C}{\partial B} \end{cases}$$

où α est le taux d'apprentissage, un paramètre à choisir manuellement et C la fonction de coût.

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

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

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

Question : En utilisant la relation précédente, donner la relation matricielle $\frac{\partial C}{\partial W}$, $\frac{\partial C}{\partial A}$, $f'(Y)$ 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 $\frac{\partial C}{\partial W} = \left(\frac{\partial C}{\partial A}.f'(Y)\right)X^T$

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

En poursuivant l'analyse il est possible de trouver la relation suivante qui va permettre d'exprimer l'erreur en entrée en fonction de l'erreur en sortie : $\frac{\partial C}{\partial X} = \frac{\partial C}{\partial A}(W.f'(WX + B))$

8. Choix d'une solution pour un problème de prédiction

Notations : On suppose que l'on travaille avec des entrées $x \in \mathbb{R}^d$, des sorties $y \in \mathbb{R}^p$ et une base de données d'entraînement de n exemples.

	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^TX $\mathcal{O}(d^3 + nd^2)$ 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 → 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

9. Pour aller plus loin…

Les réseaux présentés ci-dessus sont les réseaux dits denses (ou *fully-connected*). On utilise d'autre types de réseaux pour une meilleure prédiction en fonction des données d'entrées :

- Réseaux convolutifs (CNN) pour l'analyse d'images.
- Réseaux de neurones récurrents pour les données temporelles, etc..

Références :

M.T. Hagan, H.B. Demuth, M.H. Beale et O. De Jesús : Neural Network Design. Martin Hagan, 2014. ISBN 9780971732117.

Martin T Hagan, Howard B Demuth et Orlando De Jesús : An introduction to the use of neural networks in control systems. International Journal of Robust and Nonlinear Control : IFAC-Affiliated Journal, 12 (11):959–985, 2002.

Kurt Hornik, Maxwell Stinchcombe et Halbert White : Multilayer feedforward networks are universal approximators. Neural networks, 2(5):359–366, 1989.

Derrick Nguyen et Bernard Widrow : Improving the learning speed of 2-layer neural networks by choosing initial values of the adaptive weights. In 1990 IJCNN International Joint Conference on Neural Networks, pages 21–26. IEEE, 1990.

George EP Box, Gwilym M Jenkins, Gregory C Reinsel et Greta M Ljung : Time series analysis : forecasting and control. John Wiley & Sons, 2015.

10. Modules pythons pour l'IA