Session 1 — Rappels mathématiques et approfondissement des méthodes de classification

Maxence Cabiddu maxence.cabiddu@gmail.com

1. Rappels mathématiques fondamentaux

1.1 Algèbre linéaire

Concepts clés :

- **Vecteurs** :
  - Liste ordonnée de nombres.
  - Exemple : caractéristiques d'un échantillon.

  Exemple de vecteur v :

  $$v = \begin{bmatrix} 170 \\ 65 \end{bmatrix}$$

  où :

  - 170 = taille en cm,
  - 65 = poids en kg.
- **Matrices** :
  - Tableau rectangulaire de vecteurs.
  - Plusieurs échantillons organisés en lignes/colonnes.

  Exemple de matrice A représentant trois individus :

  $$A = \begin{bmatrix} 170 & 65 \\ 160 & 60 \\ 180 & 75 \end{bmatrix}$$

  Chaque ligne correspond à un individu : $(\text{taille}, \text{poids})$.

Opérations fondamentales :

- **Produit scalaire (dot product)** :

  Combine deux vecteurs v et u en un seul nombre réel :

  $$v \cdot u = \sum_{i=1}^{n} v_i u_i$$

➔ Cela mesure **à quel point deux vecteurs pointent dans la même direction**.
Si le produit est élevé, les vecteurs sont "alignés" ; s'il est nul, ils sont perpendiculaires.

Exemple :

Supposons deux individus décrits par (taille en cm, poids en kg) :

- Individu A :

$$v = \begin{bmatrix} 170 \\ 65 \end{bmatrix}$$

- Individu B :

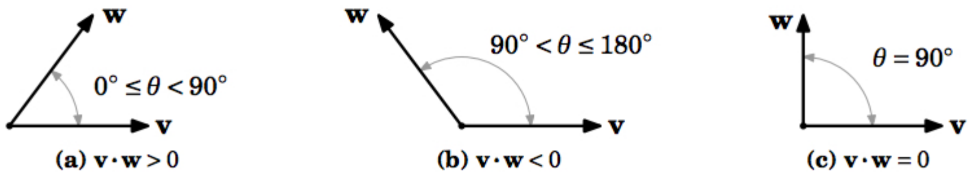
$$u = \begin{bmatrix} 180 \\ 70 \end{bmatrix}$$

Calcul du produit scalaire :

$$v \cdot u = (170 \times 180) + (65 \times 70)$$

$$v \cdot u = 30600 + 4550 = 35150$$

➔ Le produit scalaire est élevé, ce qui signifie que **les vecteurs (caractéristiques)** sont **assez alignés** :
les deux individus sont **de taille et de poids relativement similaires**.



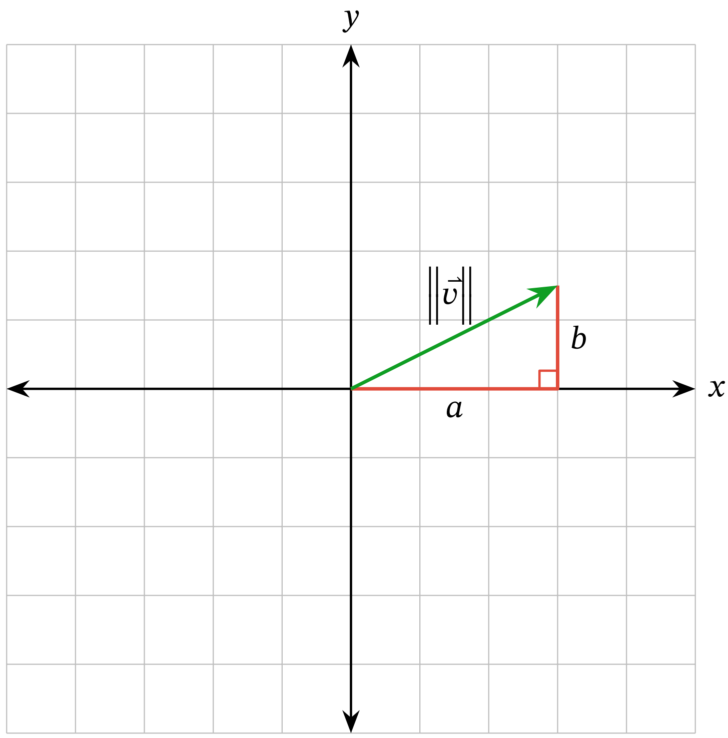
Source : math.libretexts.org – Illustration produit scalaire

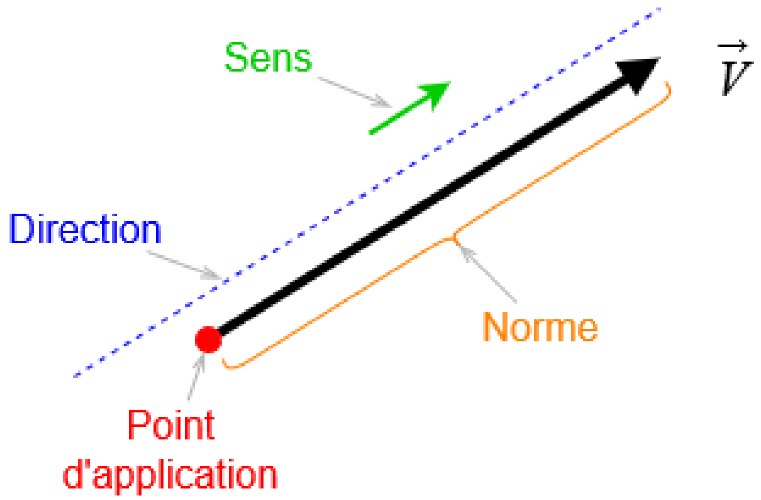
- **Norme** :

Donne la **longueur** d'un vecteur v, c'est-à-dire sa distance à l'origine :

$$\|v\| = \sqrt{\sum_{i=1}^{n} v_i^2}$$

➔ Permet de calculer des distances ou de comparer la taille de différents vecteurs.





Source : nagwa.com – Illustration vecteur 1 Source : blaisepascal.fr – Illustration vecteur 2

Exemple :

$$v = (3, 4)$$

$$\|v\| = \sqrt{3^2 + 4^2} = \sqrt{25} = 5$$

- **Multiplication matrice-vecteur** :

  Multiplie une matrice A par un vecteur x :

  $$y = A \times x$$

  ➔ Permet d'**appliquer une transformation linéaire** à un vecteur : rotation, changement d'échelle, etc.

Exemple :

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}, \quad x = \begin{bmatrix} 5 \\ 6 \end{bmatrix}$$

$$y = \begin{bmatrix} 1 \times 5 + 2 \times 6 \\ 3 \times 5 + 4 \times 6 \end{bmatrix} = \begin{bmatrix} 17 \\ 39 \end{bmatrix}$$

- **Multiplication matrice-matrice** :

  Multiplie deux matrices A (de taille $m \times n$) et B (de taille $n \times p$) :

  $$C = A \times B$$

  Chaque élément C_{ij} est calculé par :

  $$C_{ij} = \sum_{k=1}^{n} A_{ik} B_{kj}$$

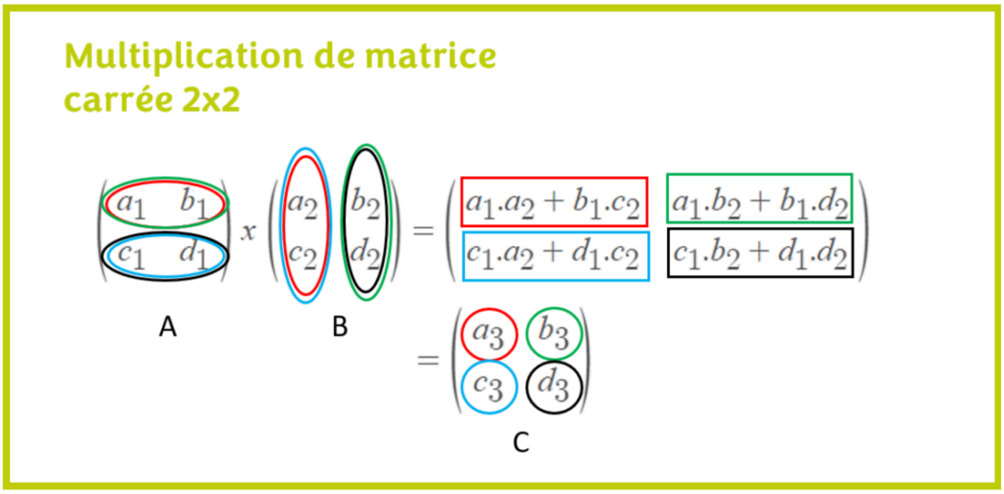
  ➔ Permet de **combiner plusieurs transformations** successives en une seule opération.

  Exemple :

  $$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}, \quad B = \begin{bmatrix} 7 & 8 \\ 9 & 10 \end{bmatrix}$$

  $$C = A \times B = \begin{bmatrix} 1 \times 7 + 2 \times 9 & 1 \times 8 + 2 \times 10 \\ 3 \times 7 + 4 \times 9 & 3 \times 8 + 4 \times 10 \end{bmatrix} = \begin{bmatrix} 25 & 28 \\ 57 & 64 \end{bmatrix}$$

Illustration de la multiplication de matrices



Source : Tutorat Homes – Multiplication de matrice

In [30]:
```python
# Module interactif pour tester les opérations de base (sur des individus

import numpy as np
```

```python
# Définir un vecteur v (exemple : caractéristiques d'une personne : taill
v = np.array([170, 65])  # Personne A : 170 cm, 65 kg

# Définir un autre vecteur u
u = np.array([180, 70])  # Personne B : 180 cm, 70 kg

# Définir une matrice A (ex : 3 personnes différentes)
A = np.array([
    [170, 65],  # Personne 1
    [160, 60],  # Personne 2
    [180, 75]   # Personne 3
])

# Définir un vecteur x pour tester la multiplication matrice-vecteur
x = np.array([1, 2])

# Définir une autre matrice B pour tester la multiplication matrice-matri
B = np.array([
    [7, 8],
    [9, 10]
])

# ----------------------
# Calculs
# ----------------------

print("Vecteur v (Personne A) :", v)
print("Vecteur u (Personne B) :", u)
print("Produit scalaire v . u :", np.dot(v, u))
print("Norme de v (longueur du vecteur v) :", np.linalg.norm(v))

print("\nMatrice A (3 personnes) :\n", A)
print("Dimension de A: ", A.shape)
print("Vecteur x :", x)
print("Multiplication matrice-vecteur A @ x :\n", A @ x)

print("\nMatrice B :\n", B)
print("Dimension de B: ", B.shape)
print("Multiplication matrice-matrice A[:2, :] @ B :\n", A @ B)  # attent
# A @ B equivalent A.dot(B)
A.dot(B)
```

```
Vecteur v (Personne A) : [170  65]
Vecteur u (Personne B) : [180  70]
Produit scalaire v . u : 35150
Norme de v (longueur du vecteur v) : 182.00274723201295

Matrice A (3 personnes) :
 [[170  65]
 [160  60]
 [180  75]]
Dimension de A:  (3, 2)
Vecteur x : [1 2]
Multiplication matrice-vecteur A @ x :
 [300 280 330]

Matrice B :
 [[ 7  8]
 [ 9 10]]
Dimension de B:  (2, 2)
Multiplication matrice-matrice A[:2, :] @ B :
 [[1775 2010]
 [1660 1880]
 [1935 2190]]
```

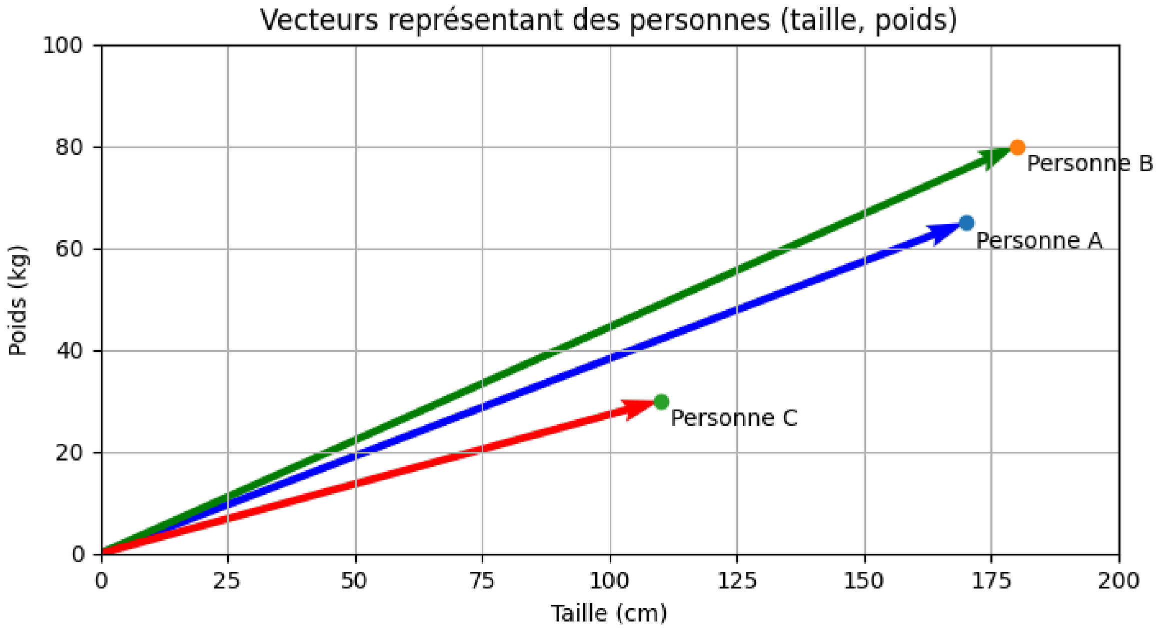
Out[30]:

```
array([[1775, 2010],
       [1660, 1880],
       [1935, 2190]])
```

In [29]:

```python
# Affichage des vecteurs représentant les personnes dans un repère, depui

import numpy as np
import matplotlib.pyplot as plt

# Définir les vecteurs des personnes (taille en cm, poids en kg)
personnes = np.array([
    [170, 65],  # Personne A
    [180, 80],  # Personne B
    [110, 30]   # Personne C
])

noms = ['Personne A', 'Personne B', 'Personne C']

# Création du graphique
plt.figure(figsize=(8, 6))

# Tracer les vecteurs depuis l'origine (0, 0)
origin = np.zeros((3, 2))
plt.quiver(
    origin[:, 0], origin[:, 1],
    personnes[:, 0], personnes[:, 1],
    angles='xy', scale_units='xy', scale=1, color=['blue', 'green', 'red'
)

# Ajouter les extrémités
for i in range(len(personnes)):
    plt.plot(personnes[i, 0], personnes[i, 1], 'o')
    plt.text(personnes[i, 0] + 2, personnes[i, 1] - 5, noms[i])

# Configurer les axes pour démarrer à 0
plt.xlim(0, 200)
plt.ylim(0, 100)
```

```
plt.xlabel('Taille (cm)')
plt.ylabel('Poids (kg)')
plt.title('Vecteurs représentant des personnes (taille, poids)')
plt.grid()
plt.gca().set_aspect('equal')  # Échelle équivalente pour X et Y

plt.show()
```



1.2 Probabilités et statistiques

Concepts clés :

- **Variable aléatoire** :
  - Discrète : prend un nombre fini ou dénombrable de valeurs (ex : lancer un dé → {1,2,3,4,5,6})
  - Continue : peut prendre toutes les valeurs d'un intervalle (ex : température mesurée en degrés)

- **Espérance** (valeur moyenne théorique) :

$$\mathbb{E}[X] = \sum_x x\,\mathbb{P}(X = x) \quad \text{(discret)}$$

ou

$$\mathbb{E}[X] = \int_{-\infty}^{+\infty} x\,f(x)\,dx \quad \text{(continu)}$$

➔ L'espérance donne **la moyenne attendue** d'une variable aléatoire si l'on répète l'expérience à l'infini.

Exemple discret : lancer de dé équilibré

$$\mathbb{E}[X] = \frac{1+2+3+4+5+6}{6} = 3.5$$

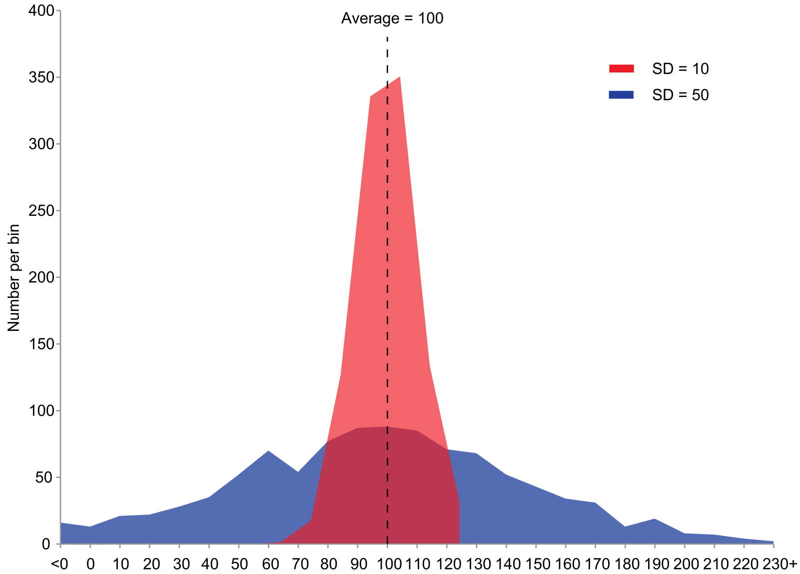
- **Variance** (dispersion autour de l'espérance) :

$$\mathrm{Var}(X) = \mathbb{E}[(X - \mathbb{E}[X])^2]$$

➔ La variance mesure **l'écart moyen au carré** entre les réalisations et la moyenne attendue.
➔ Plus la variance est grande, plus les valeurs de X sont **dispersées** autour de $\mathbb{E}[X]$.

Intuition :

  - Faible variance : les tirages sont proches de la moyenne.
  - Forte variance : les tirages peuvent être très éloignés de la moyenne.



Source : Wikimedia - Illustration variance

Exemple rapide :

Si X est un dé équilibré :

$$\mathrm{Var}(X) = \mathbb{E}[X^2] - (\mathbb{E}[X])^2$$

où

$$\mathbb{E}[X^2] = \frac{1^2 + 2^2 + 3^2 + 4^2 + 5^2 + 6^2}{6} = \frac{91}{6}$$

donc

$$\mathrm{Var}(X) = \frac{91}{6} - (3.5)^2 \approx 2.916$$

1.3 Optimisation

Concepts clés :

- **Fonction de coût** :
  - Mesure **l'erreur** entre les prédictions du modèle et les vraies valeurs.
  - Objectif : **minimiser** cette fonction.
- **Convexité** :
  - Une fonction est convexe si **n'importe quel segment entre deux points de la courbe reste au-dessus de la courbe**.
  - Convexité $\Rightarrow$ **minimum global unique**.
- **Descente de gradient** :

$$\theta \leftarrow \theta - \eta \nabla_\theta J(\theta)$$

  - À chaque étape, on ajuste les paramètres θ **dans la direction qui réduit** la fonction de coût $J(\theta)$.
  - η est le **taux d'apprentissage** (learning rate).

Exemple simple :

Minimiser :

$$f(x) = (x-3)^2$$

- La fonction $f(x)$ est convexe (parabole tournée vers le haut).
- Son minimum global est atteint pour $x = 3$.

2. Régression logistique

2.1 Modèle et fonction de coût

Modèle binaire (classification à 2 classes) :

On cherche à prédire une probabilité $\hat{y}$ que l'échantillon appartienne à la classe 1 :

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

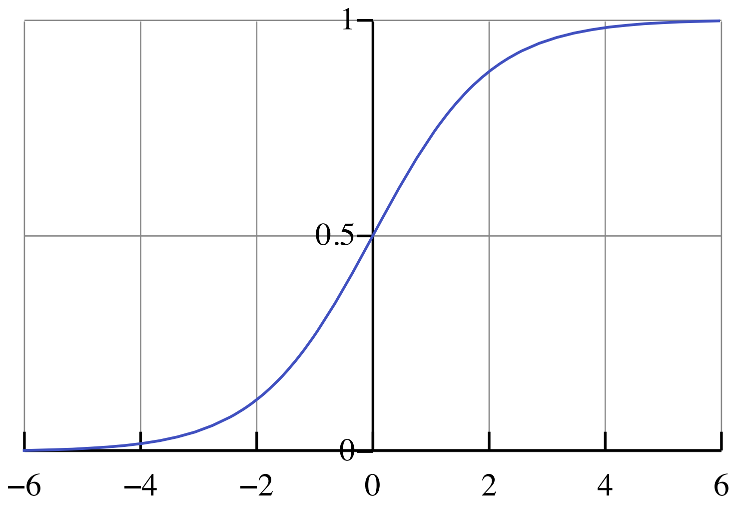
où :

- w est le vecteur de poids,
- x est le vecteur d'entrée,
- b est le biais,
- $\sigma(z)$ est la **fonction sigmoïde** :

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

➔ La sigmoïde "écrase" n'importe quel nombre réel entre 0 et 1 : elle transforme une somme pondérée de variables en **probabilité**.

Visualisation de la fonction sigmoïde :



Source : Wikipedia – Logistic Function

- L'axe horizontal montre z (le score linéaire).
- L'axe vertical montre $\sigma(z)$ (la probabilité prédite).

On voit que :

- Quand z est très grand, $\sigma(z) \approx 1$,
- Quand z est très petit, $\sigma(z) \approx 0$,
- Et autour de $z = 0$, la courbe est **la plus raide**.

Fonction de coût (log-loss) :

On mesure **l'écart entre les prédictions $\hat{y}^{(i)}$ et les vraies étiquettes $y^{(i)}$** avec une fonction appelée **log-loss** ou **cross-entropy binaire** :

$$J(w,b) = -\frac{1}{N}\sum_{i=1}^{N}\left[y^{(i)}\log(\hat{y}^{(i)}) + (1-y^{(i)})\log(1-\hat{y}^{(i)})\right]$$

Intuition :

- Si la prédiction est correcte ($\hat{y} \approx y$), la perte est faible.
- Si la prédiction est fausse ($\hat{y}$ est loin de y), la perte est forte.

➔ L'objectif est de **minimiser cette fonction de coût**.

Remarque

Quand on calcule la log-loss pour un seul exemple :

- Si la vraie étiquette $y = 1$, la loss devient :

$$-\log(\hat{y})$$

- Si la vraie étiquette $y = 0$, la loss devient :

$$-\log(1-\hat{y})$$

➔ La log-loss **s'adapte automatiquement** selon que l'exemple appartient à la classe 1 ou la classe 0.

Exemple rapide :

- Vraie étiquette : $y = 1$
- Prédiction : $\hat{y} = 0.9$

La contribution à la fonction de coût est :

$$-(1 \times \log(0.9) + (1-1) \times \log(1-0.9)) = -\log(0.9) \approx 0.105$$

(valeur faible = bonne prédiction)

Extension multiclasses :

Quand il y a plus de deux classes :

- On utilise **softmax** à la place de la sigmoïde :

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

- Chaque sortie représente une **probabilité par classe**. **Exemple rapide :**
  Si un modèle produit les scores $(2.0,\ 1.0,\ 0.1)$, alors après softmax :

$$\hat{y} \approx (0.66,\ 0.24,\ 0.10)$$

➔ Le modèle attribue 66% à la première classe, 24% à la deuxième, 10% à la troisième.

- La fonction de coût devient la **cross-entropy catégorique** :

$$J = -\frac{1}{N} \sum_{i=1}^{N} \sum_{c=1}^{C} y_c^{(i)} \log(\hat{y}_c^{(i)})$$

où :

- C est le nombre de classes,
- $y_c^{(i)}$ est 1 si l'observation i appartient à la classe c, sinon 0.

Résumé visuel :

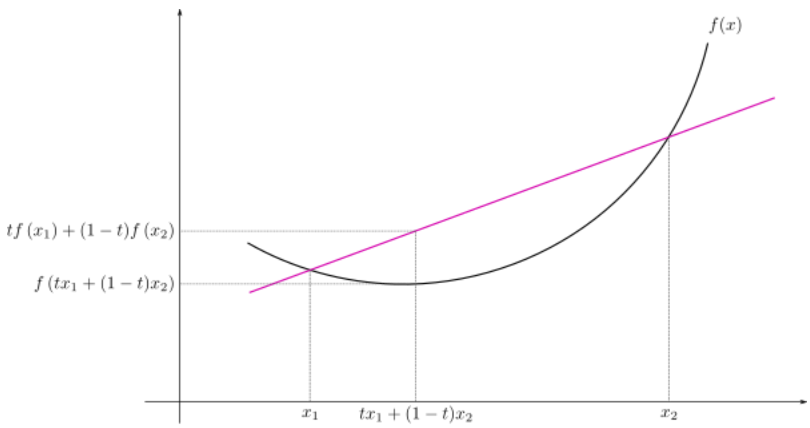
Cas	Activation	Coût utilisé
Binaire (2 classes)	Sigmoïde	Log-loss
Multiclasses	Softmax	Cross-entropy catégorique

```
In [41]: import numpy as np
         import torch

         scores = np.array([2.0, 1.0, 0.1])

         softmax_class1 = np.exp(2.0) / np.sum(np.exp(scores))
         print(softmax_class1)

         torch.softmax(torch.tensor(scores), dim=0)
```

```
0.6590011388859679
Out[41]: tensor([0.6590, 0.2424, 0.0986], dtype=torch.float64)
```

Pourquoi la fonction de coût doit être convexe

- Une fonction **convexe** a **un seul minimum global**.
- Cela garantit que **la descente de gradient** trouvera ce minimum sans être piégée dans un mauvais "creux".



Source : [Wikipedia — Convex Function]

- fonction **convexe** ➔ minimum global facile à trouver,

Dans la régression logistique :

- La log-loss est **convexe** par rapport à w et b.
- C'est pour cela que l'optimisation par descente de gradient **fonctionne bien**.

Exemple : classification binaire avec régression logistique

On considère un modèle de régression logistique simple avec :

- Vecteur de poids : $w = \begin{bmatrix} 2 \\ -1 \end{bmatrix}$
- Biais : $b = 0.5$

On veut prédire pour l'exemple suivant :

- Entrée $x = \begin{bmatrix} 1 \\ 3 \end{bmatrix}$

- Vraie étiquette $y = 0$

Étape 1 : Calcul du score linéaire z

$$z = w^T x + b = (2 \times 1) + (-1 \times 3) + 0.5 = 2 - 3 + 0.5 = -0.5$$

Étape 2 : Passage par la fonction sigmoïde

$$\hat{y} = \sigma(z) = \frac{1}{1 + e^{-z}} = \frac{1}{1 + e^{0.5}} \approx \frac{1}{1 + 1.6487} \approx 0.3775$$

➔ Le modèle prédit $\hat{y} \approx 0.3775$ (probabilité d'appartenir à la classe 1).

Étape 3 : Calcul de la log-loss pour cet exemple

La fonction de coût pour un seul exemple est :

$$\text{log-loss} = -\left(y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})\right)$$

Ici, $y = 0$, donc :

$$\text{log-loss} = -\log(1 - \hat{y}) = -\log(1 - 0.3775) = -\log(0.6225)$$

$$\text{log-loss} \approx -(-0.4737) = 0.4737$$

Résumé

Étape	**Résultat**
Score linéaire z	-0.5
Probabilité prédite $\hat{y}$	0.3775
Fonction de coût (log-loss)	0.4737

Interprétation :

- Le modèle donne 37% de chances que x soit de classe 1.
- Comme la vraie classe est 0, la perte est **modérée** (≈ 0.47).

2.2 Dérivées, dérivées partielles et gradient

Pourquoi utiliser des dérivées ?

Notre objectif est de **minimiser** la fonction de coût $J(w, b)$.

- **Minimiser**, c'est trouver là où la fonction atteint son **point le plus bas**.
- En mathématiques, pour savoir comment une fonction change, on utilise **la dérivée**.

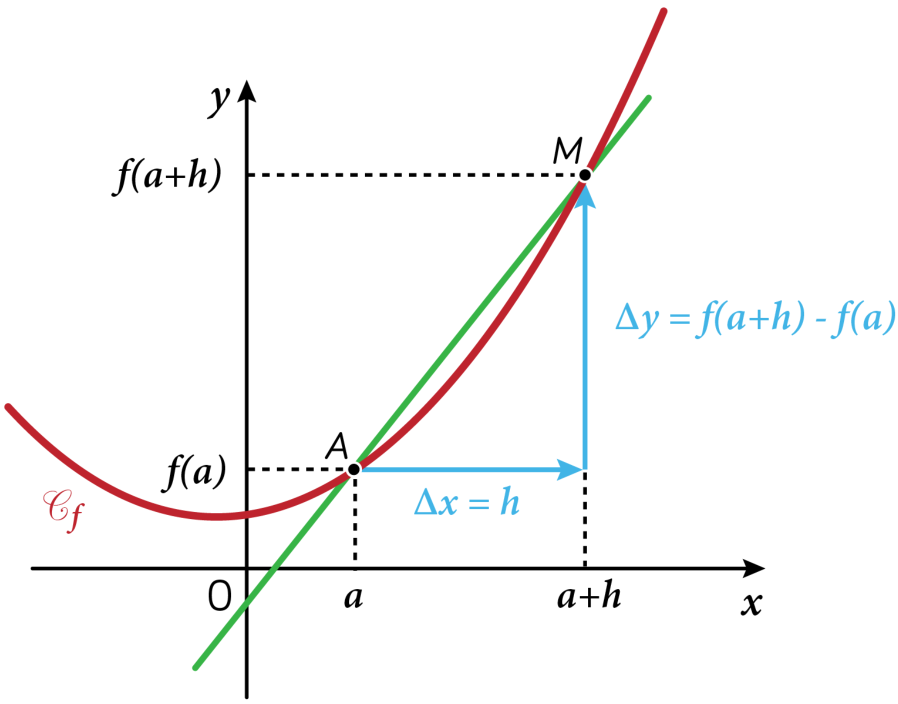
La **dérivée** d'une fonction $f(x)$ au point x mesure :

> "À quelle vitesse (et dans quelle direction) $f(x)$ change si je fais **un tout petit pas** autour de x."

Définition formelle (limite) :

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

- h est un tout petit déplacement.
- Le rapport mesure **le taux de variation**.



Source : schoolmouv.fr – Illustration dérivée

En optimisation :

- Si la dérivée est **positive**, la fonction monte → on veut aller **vers la gauche** (réduire x).
- Si la dérivée est **négative**, la fonction descend → on veut aller **vers la droite** (augmenter x).

Descente de gradient = **aller dans la direction opposée à la dérivée** pour diminuer la fonction.

Dérivées partielles

Quand la fonction dépend de **plusieurs variables** (par exemple tous les poids $w_1, w_2, \ldots, w_n$), on parle de **dérivées partielles**.

Dérivée partielle = dérivée **par rapport à une seule variable**, **en laissant les autres constantes**.

Notation pour $J(w_1, w_2, w_3)$:

- $\frac{\partial J}{\partial w_1}$: variation de J quand **seul** w_1 change,
- $\frac{\partial J}{\partial w_2}$: variation de J quand **seul** w_2 change,
- etc.

Pourquoi ?

- Cela nous permet de savoir **comment ajuster chaque poids individuellement** pour réduire la perte.

Gradient

Le **gradient** est simplement **le vecteur de toutes les dérivées partielles** :

$$\nabla_w J = \begin{bmatrix} \frac{\partial J}{\partial w_1} \\ \frac{\partial J}{\partial w_2} \\ \vdots \\ \frac{\partial J}{\partial w_n} \end{bmatrix}$$

- Le gradient indique **la direction dans laquelle J augmente le plus vite**.
- En descente de gradient, **on suit la direction opposée** pour **réduire J**.

Métaphore simple :

> Imagine que tu es dans une vallée en montagne avec les yeux fermés.
> Le **gradient** est le **vecteur qui te dit dans quelle direction la pente monte le plus fort**.
> Pour descendre, tu fais l'inverse.

Résumé visuel :

Concept	Explication simple
Dérivée	Variation locale d'une fonction par rapport à son entrée
Dérivée partielle	Variation locale par rapport à **une seule variable**
Gradient	Ensemble des dérivées partielles = **direction de la plus forte montée**
Descente de gradient	Faire des petits pas dans la direction opposée pour **réduire la perte**

Exemple : Calcul des gradients pour un seul exemple de régression logistique

Rappel

- Poids :

$$w = \begin{bmatrix} 2 \\ -1 \end{bmatrix}$$

- Biais :

$$b = 0.5$$

- Entrée :

$$x = \begin{bmatrix} 1 \\ 3 \end{bmatrix}$$

- Vraie étiquette :

$$y = 0$$

Étape 1 : Calcul du score linéaire z

Déjà fait précédemment :

$$z = w^T x + b = -0.5$$

Étape 2 : Calcul de la probabilité prédite $\hat{y}$

Déjà fait précédemment :

$$\hat{y} = \sigma(z) \approx 0.3775$$

Étape 3 : Calcul de l'erreur $(\hat{y} - y)$

$$\hat{y} - y = 0.3775 - 0 = 0.3775$$

Étape 4 : Calcul du gradient par rapport à w

La formule pour un seul exemple est :

$$\nabla_w J = (\hat{y} - y)x$$

Calcul :

$$\nabla_w J = 0.3775 \times \begin{bmatrix} 1 \\ 3 \end{bmatrix} = \begin{bmatrix} 0.3775 \\ 1.1325 \end{bmatrix}$$

Étape 5 : Calcul du gradient par rapport à b

La formule est :

$$\partial_b J = (\hat{y} - y)$$

Donc :

$$\partial_b J = 0.3775$$

Résumé des gradients

Élément	Valeur
$\nabla_w J$	$\begin{bmatrix} 0.3775 \\ 1.1325 \end{bmatrix}$
$\partial_b J$	0.3775

Étape 6 : Mise à jour des paramètres (si learning rate $\eta = 0.1$)

- Mise à jour de w :

  $$w \leftarrow w - \eta \nabla_w J$$

  Calcul :

  $$w = \begin{bmatrix} 2 \\ -1 \end{bmatrix} - 0.1 \times \begin{bmatrix} 0.3775 \\ 1.1325 \end{bmatrix} = \begin{bmatrix} 2 - 0.03775 \\ -1 - 0.11325 \end{bmatrix} = \begin{bmatrix} 1.96225 \\ -1.11325 \end{bmatrix}$$

- Mise à jour de b :

  $$b \leftarrow b - \eta \partial_b J$$

  Calcul :

  $$b = 0.5 - 0.1 \times 0.3775 = 0.5 - 0.03775 = 0.46225$$

Résumé après mise à jour

Élément	Nouvelle valeur
w	$\begin{bmatrix} 1.96225 \\ -1.11325 \end{bmatrix}$
b	0.46225

Interprétation

Après une itération de descente de gradient :

- Les poids w et le biais b **ont été corrigés**,
- Le modèle **va mieux prédire** l'étiquette $y = 0$ pour cet exemple particulier.

2.3 Impact du learning rate

Qu'est-ce que le learning rate ?

- Le **learning rate** (η) contrôle **la taille des pas** que l'on fait à chaque mise à jour des paramètres.
- Il joue un rôle **crucial** dans l'efficacité et la stabilité de l'apprentissage.

Effets d'un learning rate mal choisi

- **Learning rate trop petit** :
  - Les mises à jour sont **très lentes**.
  - L'optimisation peut mettre **beaucoup de temps** à converger.
- **Learning rate trop grand** :
  - Le modèle peut **osciller** autour du minimum sans jamais s'en approcher.
  - Peut même **diverger**, c'est-à-dire que la perte augmente au lieu de diminuer.

Illustration schématique

Learning rate	Comportement
Petit η	Convergence lente
Bon η	Convergence rapide et stable
Grand η	Divergences, oscillations

Recommandation pratique

- Toujours **tester plusieurs valeurs** de learning rate (ex : 0.001, 0.01, 0.1, 1.0).
- Observer l'évolution de la **fonction de coût** au cours des itérations.
- Éventuellement utiliser des méthodes de réglage automatique (scheduler, annealing).

2.4 Standardisation des données

Pourquoi standardiser les données ?

- La régression logistique repose sur l'optimisation du produit $w^T x$.
- Si certaines features ont des valeurs beaucoup plus grandes que d'autres, elles peuvent **dominer l'apprentissage**.
- Cela peut rendre la convergence **plus difficile** et **plus lente**.

Qu'est-ce que standardiser ?

Standardiser consiste à transformer chaque feature x_j selon la formule :

$$x'_j = \frac{x_j - \mu_j}{\sigma_j}$$

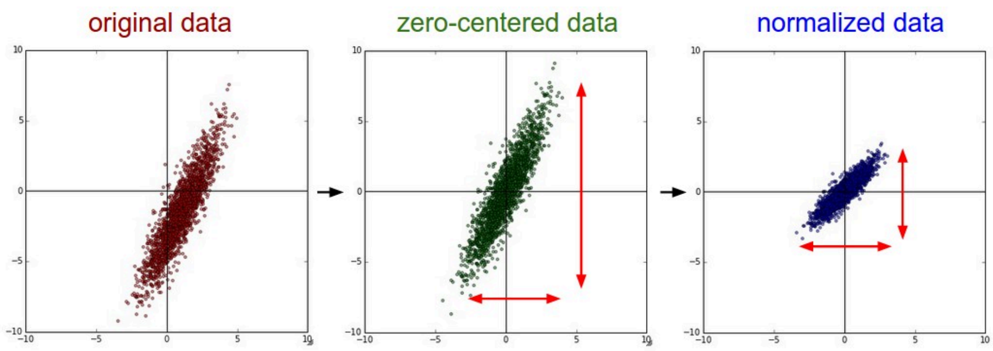
où :

- μ_j est la **moyenne** de la feature x_j sur le dataset,
- σ_j est **l'écart-type** de la feature x_j.

Après standardisation :

- Chaque feature a une moyenne 0 et un écart-type 1.

Normalization



Source : [Normalisation des données]

Effets positifs de la standardisation

- Facilite **la descente de gradient** (les courbes de coût sont plus "circulaires" au lieu d'être allongées).
- Permet au modèle d'**apprendre plus rapidement**.
- Rend **l'interprétation** des poids plus cohérente.

Quand est-ce indispensable ?

- Lorsque les features sont sur **des échelles différentes** (ex : taille en cm, poids en kg, revenu en euros).

- En pratique, il est souvent **préférable de standardiser systématiquement** pour les modèles linéaires.

Remarque

- Si une régularisation est utilisée (comme L2), la standardisation est **encore plus importante** pour éviter un biais artificiel dû à la différence d'échelle des features.

Exemple pratique

Utilisation classique avec `scikit-learn` :

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

2.5 Régularisation (L2)

Pourquoi ajouter de la régularisation ?

- Quand le modèle est trop **complexe** (beaucoup de features), il peut **sur-apprendre** (overfitting) les détails du jeu de données.
- La **régularisation** pénalise les poids trop grands pour **forcer le modèle à rester simple**.

Régularisation L2 (Ridge)

La régularisation L2 ajoute un terme de pénalisation à la fonction de coût :

$$J_{\text{total}}(w, b) = J(w, b) + \lambda \|w\|_2^2$$

où :

- $J(w, b)$ est la fonction log-loss classique,
- $\|w\|_2^2 = \sum_j w_j^2$ est la norme $L2$ au carré (la somme des carrés des poids),
- λ est un hyperparamètre qui contrôle l'intensité de la pénalisation.

Effet de la régularisation

- Encourage les **petits poids** w_j (proches de zéro).
- Réduit le **risque de sur-apprentissage**.
- Améliore parfois la **généralisation** du modèle sur de nouvelles données.

Formules de gradient modifiées

- Pour les poids w :

$$\nabla_w J_{\text{total}} = \nabla_w J + 2\lambda w$$

- Le gradient par rapport au biais b **ne change pas** car on ne pénalise pas le biais.

Recommandation pratique

- Utiliser un λ modéré au début (ex : 0.01, 0.1).
- Régler λ par validation croisée pour trouver le meilleur compromis entre biais et variance.

Exemple pratique avec `scikit-learn`

La régularisation est gérée automatiquement dans `LogisticRegression` :

```python
from sklearn.linear_model import LogisticRegression

# C est l'inverse de la régularisation (C = 1 / lambda)
model = LogisticRegression(C=1.0, penalty='l2')
model.fit(X_train, y_train)
```