

Principes du ML & Régression Linéaire

Jour 1 — Après-midi

Julien Rolland

Formation M2 Développement Fullstack

Jour 1

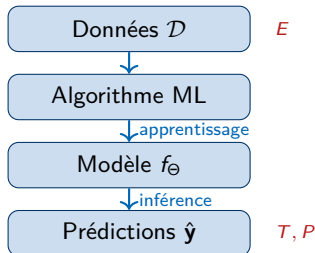
- 1 Vocabulaire du Machine Learning
- 2 Régression Linéaire
- 3 Descente de Gradient

Qu'est-ce que le Machine Learning ?

Définition (Tom Mitchell, 1997)

Pour une **tâche** T , un algorithme **apprend** si sa **performance** P sur T s'améliore avec l'**expérience** E .

	Concept	En pratique
T	Tâche	Prédire un prix
P	Performance	Erreur MSE
E	Expérience	Dataset d'entraînement



Paradigme clé

On ne **programme** plus les règles.

On **montre des exemples** — le modèle induit les règles.

Supervisé



(x_i, y_i)

Données **étiquetées**
Régression, Classification

Non-supervisé



x_i seulement

Données **sans labels**
Clustering, Embeddings

Renforcement



Apprentissage par **récompense**
Jeux, Robotique

Ce cours

Focus sur l'**apprentissage supervisé** (J1–J4) et les outils de mise en production (J5). Le non-supervisé apparaît en J4 (embeddings).

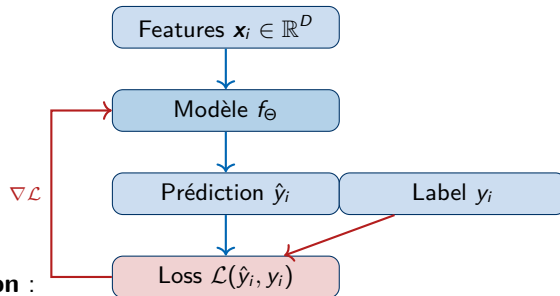
Dataset $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^N$

- $\mathbf{x}_i \in \mathbb{R}^D$ — vecteur de **features**
- $y_i \in \mathbb{R}$ (régression) ou $\{0, 1\}$ (classification)
⇒ le **label** (ground truth)
- $\mathbf{X} \in \mathbb{R}^{N \times D}$ — matrice de données

Θ = paramètres **appris** pendant l'entraînement.

Modèle f_Θ — appliqué à \mathbf{x}_i donne une **prédiction** :

$$\hat{y}_i = f_\Theta(\mathbf{x}_i)$$



Apprentissage = trouver Θ^* qui minimise l'écart entre \hat{y}_i et y_i :

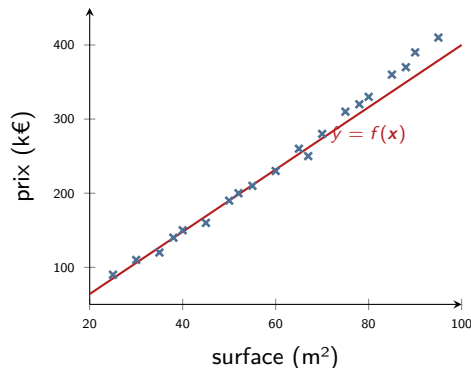
$$\Theta^* = \arg \min_{\Theta} \mathcal{L}(\hat{\mathbf{y}}, \mathbf{y})$$

Objectif : prédire une sortie **continue** $y \in \mathbb{R}$ à partir de features $\mathbf{x} \in \mathbb{R}^D$.

Exemples :

- Surface + localisation \rightarrow prix d'un logement
- Historique météo \rightarrow température demain
- Âge + poids \rightarrow glycémie

On cherche $f : \mathbb{R}^D \rightarrow \mathbb{R}$ telle que $f(\mathbf{x}_i) \approx y_i$.



Modèle linéaire ($D = 1$)

$$f_{w,b}(x) = wx + b$$

Modèle linéaire (multivarié)

$$f_{\mathbf{w},b}(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + b = \sum_{j=1}^D w_j x_j + b$$

Modèle polynomial (feature map Φ)

$$f_{\Theta}(\mathbf{x}) = \boldsymbol{\theta}^\top \Phi(\mathbf{x})$$

Ex : $\Phi(x) = [1, x, x^2, x^3]$ pour degré 3.

Mean Squared Error (MSE)

$$\mathcal{L} = \frac{1}{N} \sum_{i=1}^N (f_{\Theta}(\mathbf{x}_i) - y_i)^2 = \frac{1}{N} \|\hat{\mathbf{y}} - \mathbf{y}\|^2$$

avec $\hat{\mathbf{y}} = [f_{\Theta}(\mathbf{x}_1), \dots, f_{\Theta}(\mathbf{x}_N)]^{\top} \in \mathbb{R}^N$ le vecteur des prédictions.

Hypothèse : $y_i \sim \mathcal{N}(\hat{y}_i, \sigma^2)$ avec $\hat{y}_i = f_\theta(x_i)$ (bruit gaussien)

$$\Rightarrow L = \prod_{i=1}^n \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y_i - \hat{y}_i)^2}{2\sigma^2}\right)$$

$$\Rightarrow \ln L = \sum_{i=1}^n \left[\ln\left(\frac{1}{\sqrt{2\pi\sigma^2}}\right) - \frac{(y_i - \hat{y}_i)^2}{2\sigma^2} \right]$$

$$\Rightarrow \operatorname{argmax}_\theta \ln L \iff \operatorname{argmax}_\theta \left(- \sum_{i=1}^n (y_i - \hat{y}_i)^2 \right)$$

$$\Rightarrow \operatorname{argmax}_\theta \ln L \iff \operatorname{argmin}_\theta \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

$$\Rightarrow \operatorname{argmin}_\theta \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2 \iff \mathbf{argmin \text{ MSE}}$$

Données $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^N$

- $\mathbf{x}_i \in \mathbb{R}^D$ — features (surface, nb pièces...)
- $y_i \in \mathbb{R}$ — valeur cible à prédire (prix...)

Modèle — une droite (ou hyperplan) :

$$\hat{y}_i = f_{\mathbf{w},b}(\mathbf{x}_i) = \mathbf{w}^\top \mathbf{x}_i + b$$

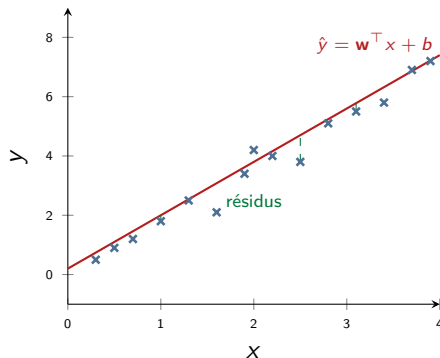
On cherche $\mathbf{w} \in \mathbb{R}^D$ et $b \in \mathbb{R}$ qui **ajustent** cette droite aux données.

Objectif — minimiser l'erreur quadratique moyenne :

$$\operatorname{argmin}_{\mathbf{w}, b} \mathcal{L}(\hat{\mathbf{y}}, \mathbf{y}) \iff \operatorname{argmin}_{\mathbf{w}, b} \frac{1}{N} \sum_{i=1}^N (\hat{y}_i - y_i)^2$$

Trouver la droite la **plus proche** de tous les points.

Ajuster une droite aux données



Forme scalaire — une prédiction à la fois :

$$\hat{y}_i = \mathbf{w}^\top \mathbf{x}_i, \quad i = 1, \dots, N$$

Forme compacte

$$\hat{\mathbf{y}} = \mathbf{X}\mathbf{w}, \quad \mathbf{X} \in \mathbb{R}^{N \times D}$$

Un seul produit matriciel pour les N prédictions.

The diagram illustrates the matrix form of the linear regression prediction equation. It shows the matrix multiplication $\mathbf{X} \cdot \mathbf{w} = \hat{\mathbf{y}}$. Matrix \mathbf{X} is $N \times D$, vector \mathbf{w} is $D \times 1$, and vector $\hat{\mathbf{y}}$ is $N \times 1$.

\mathbf{X}	\mathbf{w}	$\hat{\mathbf{y}}$
\mathbf{x}_1^\top	w_1	\hat{y}_1
\mathbf{x}_2^\top	w_2	\hat{y}_2
\vdots	\vdots	\vdots
\mathbf{x}_N^\top	w_D	\hat{y}_N
$N \times D$	$D \times 1$	$N \times 1$

Problème : gérer le biais b séparément est fastidieux.

$$\hat{y}_i = \mathbf{w}^\top \mathbf{x}_i + b \quad \Rightarrow \quad \text{deux termes à maintenir}$$

Astuce : ajouter une colonne de 1 dans \mathbf{X} :

$$\tilde{\mathbf{x}}_i = \begin{pmatrix} 1 \\ x_{i,1} \\ \vdots \\ x_{i,D} \end{pmatrix} \in \mathbb{R}^{D+1} \quad \tilde{\mathbf{w}} = \begin{pmatrix} b \\ w_1 \\ \vdots \\ w_D \end{pmatrix}$$

$$\hat{y}_i = \tilde{\mathbf{w}}^\top \tilde{\mathbf{x}}_i \quad \Leftarrow \text{un seul terme !}$$

En NumPy :

```
1 # X : (N, D) -> X_aug : (N, D+1)
2 ones = np.ones((N, 1))
3 X_aug = np.hstack([ones, X])
4 # w_aug contient [b, w1, ..., wD]
5
6 # Prediction
7 y_hat = X_aug @ w_aug
```

Avantage

Toutes les équations (gradient, solution analytique) s'écrivent **sans cas particulier** pour b .

$$\begin{aligned}\mathcal{L} &= \frac{1}{N} \|\hat{\mathbf{y}} - \mathbf{y}\|^2 \\ &= \frac{1}{N} (\mathbf{X}\mathbf{w} - \mathbf{y})^\top (\mathbf{X}\mathbf{w} - \mathbf{y}) \\ &= \frac{1}{N} (\mathbf{w}^\top \mathbf{X}^\top \mathbf{X} \mathbf{w} - 2\mathbf{w}^\top \mathbf{X}^\top \mathbf{y} + \mathbf{y}^\top \mathbf{y})\end{aligned}$$

$$\nabla_{\mathbf{w}} \mathcal{L} = \frac{1}{N} (2\mathbf{X}^\top \mathbf{X} \mathbf{w} - 2\mathbf{X}^\top \mathbf{y})$$

$$\nabla_{\mathbf{w}} \mathcal{L} = \frac{2}{N} \mathbf{X}^\top (\mathbf{X}\mathbf{w} - \mathbf{y})$$

Résultat

$$\nabla_{\mathbf{w}} \mathcal{L} = \frac{2}{N} \mathbf{X}^\top (\mathbf{X}\mathbf{w} - \mathbf{y})$$

Convexité

$$\nabla_{\mathbf{w}}^2 \mathcal{L} = \frac{2}{N} \mathbf{X}^\top \mathbf{X}$$

$\mathbf{X}^\top \mathbf{X}$ est une matrice de Gram :

$$\forall \mathbf{v} \in \mathbb{R}^D, \quad \mathbf{v}^\top (\mathbf{X}^\top \mathbf{X}) \mathbf{v} = \|\mathbf{X}\mathbf{v}\|^2 \geq 0$$

$\Rightarrow \nabla_{\mathbf{w}}^2 \mathcal{L} \succeq 0 \Rightarrow$ **minimum global unique.**

La MSE est convexe : son minimum est atteint quand le gradient s'annule.

$$\nabla_{\mathbf{w}} \mathcal{L} = 0 \implies \frac{2}{N} \mathbf{X}^\top (\mathbf{X}\mathbf{w} - \mathbf{y}) = 0 \implies \mathbf{X}^\top \mathbf{X}\mathbf{w} = \mathbf{X}^\top \mathbf{y}$$

$$\mathbf{w}^* = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y}$$

Avantages

- Solution **exacte** en un calcul
- Pas d'hyperparamètre (α , nb itérations)

Limites

- Inverser $\mathbf{X}^\top \mathbf{X} \in \mathbb{R}^{D \times D}$ coûte $\mathcal{O}(D^3)$
- Inutilisable si D ou N est très grand
- Ne généralise pas aux modèles non-linéaires

\Rightarrow En pratique on préfère la **descente de gradient**, généralisable à tout modèle.

On cherche $\mathbf{w}^* = \arg \min_{\mathbf{w}} \mathcal{L}(\hat{\mathbf{y}}, \mathbf{y})$.

Idée : le gradient $\nabla_{\mathbf{w}} \mathcal{L}$ pointe vers la **plus grande pente montante**.

On avance dans la **direction opposée**.

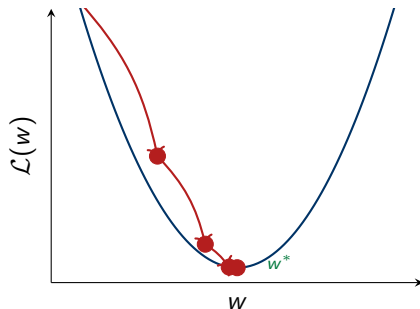
Règle de mise à jour :

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \nabla_{\mathbf{w}} \mathcal{L}(\hat{\mathbf{y}}, \mathbf{y})$$

α learning rate (hyperparamètre)

$\nabla_{\mathbf{w}} \mathcal{L}$ gradient de la loss

Répéter jusqu'à **convergence**.



Algorithme

Entrée $\mathbf{X} \in \mathbb{R}^{N \times D}$, $\mathbf{y} \in \mathbb{R}^N$, α , nb_iter

Init $\theta \leftarrow \theta_0$

Répéter nb_iter fois :

$$\mathbf{g} \leftarrow \nabla_{\theta} \mathcal{L}(f_{\theta}(\mathbf{X}), \mathbf{y})$$

$$\theta \leftarrow \theta - \alpha \mathbf{g}$$

Retourner θ

Complexité

$\mathcal{O}(N \cdot D)$ par itération.

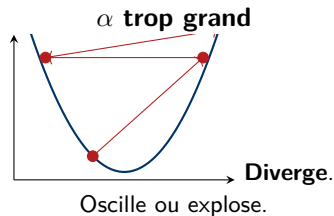
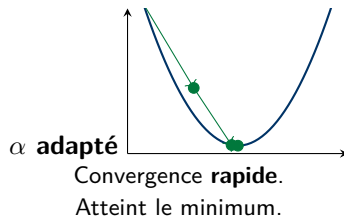
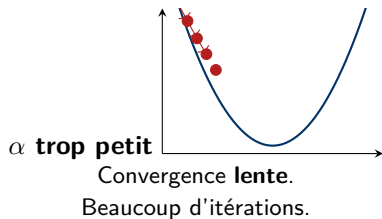
Sur grand dataset : utiliser **Mini-Batch GD**.

Variantes :

Batch GD	Gradient sur tout le dataset
SGD	Gradient sur 1 échantillon aléatoire
Mini-Batch	Gradient sur un batch de taille B

En pratique

Mini-Batch ($B = 32$ à 256) est le standard.
C'est ce qu'utilise PyTorch avec DataLoader.



Règle pratique

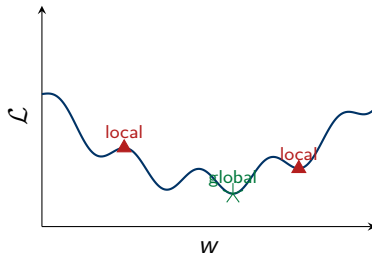
Commencer à $\alpha = 10^{-3}$, observer la courbe de loss.

Si la loss **oscille** → diviser par 10. Si elle **stagne** → multiplier par 3.

Problèmes généraux

- Converge vers un **minimum local** (pas nécessairement global)
- Résultat dépend de **l'initialisation**
- Peut **ne pas converger** si α trop grand
- Lent sur de très grands datasets (Batch GD)

Minima locaux vs global



Bonne nouvelle pour la régression linéaire

La MSE est **convexe** \Rightarrow un seul minimum, GD le trouve toujours.
C'est une propriété exceptionnelle — les réseaux profonds ne l'ont pas.

Vocabulaire ML

- Task T , Performance P , Experience E
- Supervisé / Non-supervisé / Renforcement
- Features \mathbf{X} , labels \mathbf{y} , modèle f_{Θ} , loss \mathcal{L}

Descente de Gradient

- $\mathbf{w} \leftarrow \mathbf{w} - \alpha \nabla_{\mathbf{w}} \mathcal{L}$
- Batch / SGD / Mini-Batch
- α trop petit : lent α trop grand : diverge
- MSE convexe \Rightarrow minimum global garanti

Régression Linéaire

- $\hat{y} = \mathbf{w}^{\top} \mathbf{x} + b$
- MSE : $\mathcal{L} = \frac{1}{N} \|\mathbf{X}\mathbf{w} - \mathbf{y}\|^2$
- Gradient : $\frac{2}{N} \mathbf{X}^{\top} (\mathbf{X}\mathbf{w} - \mathbf{y})$
- Augmented data : biais intégré dans \mathbf{w}