

IA Géosciences

- Deep learning:

Méthode de la descente du gradient

Romain Wenger (Laboratoire Image Ville
Environnement)

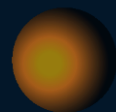


Table des matières

01

Préliminaires

Minimiser la fonction
de coût

02

Descente du gradient

Ajuster les poids

03

Régression linéaire

Démonstrations
mathématiques
appliquées à un
problème linéaire

04

Prédiction du prix des maisons

Exercice pratique

Table des matières

01

Préliminaires

Minimiser la fonction
de coût

02

Descente du gradient

Ajuster les poids

03

Régression linéaire

Démonstrations
mathématiques
appliquées à un
problème linéaire

04

Prédiction du prix des maisons

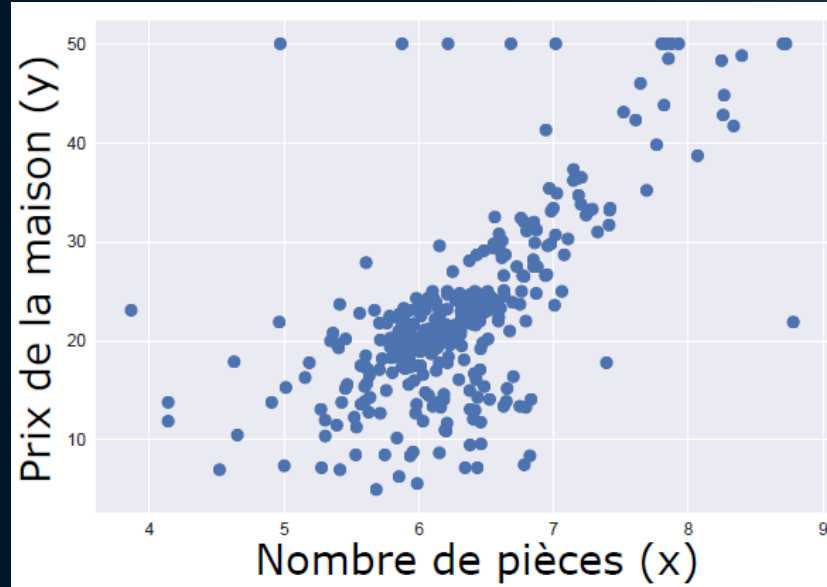
Exercice pratique

Préliminaires

1. Définir un modèle (e.g. fonction linéaire)
2. Définir un objectif (e.g. fonction de coût)
3. Minimiser la fonction objective
 - Essayer avec la force brute (trop long)
 - Solution optimale (dérivée égale zéro)
 - **Descente du gradient (signe de la dérivée)**

Boston House Prices

Prédire y en fonction de x



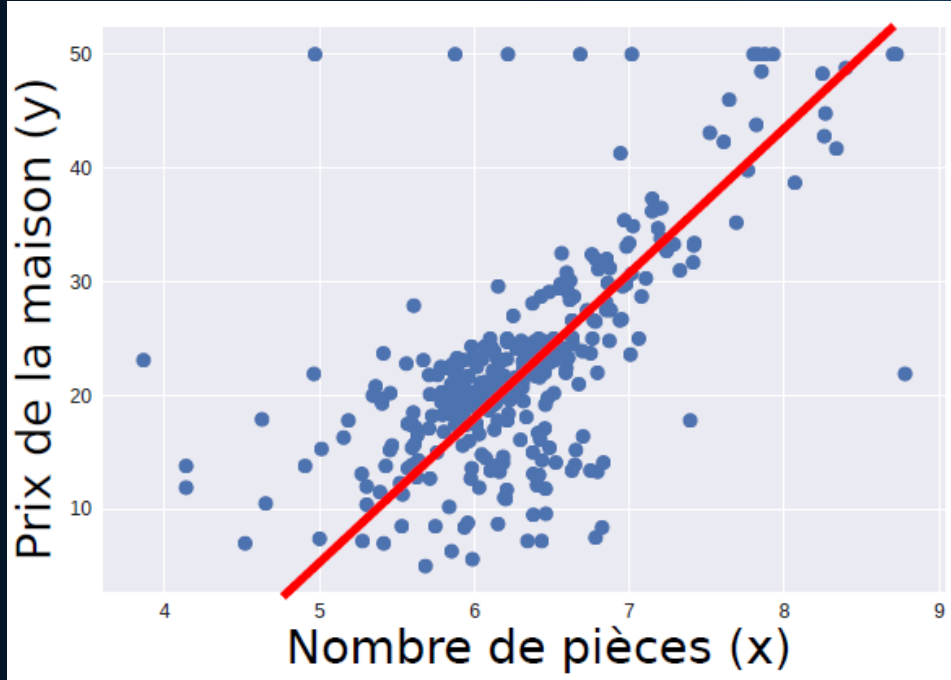
Définir un modèle

Une hypothèse :

$$\hat{y} = h(x)$$

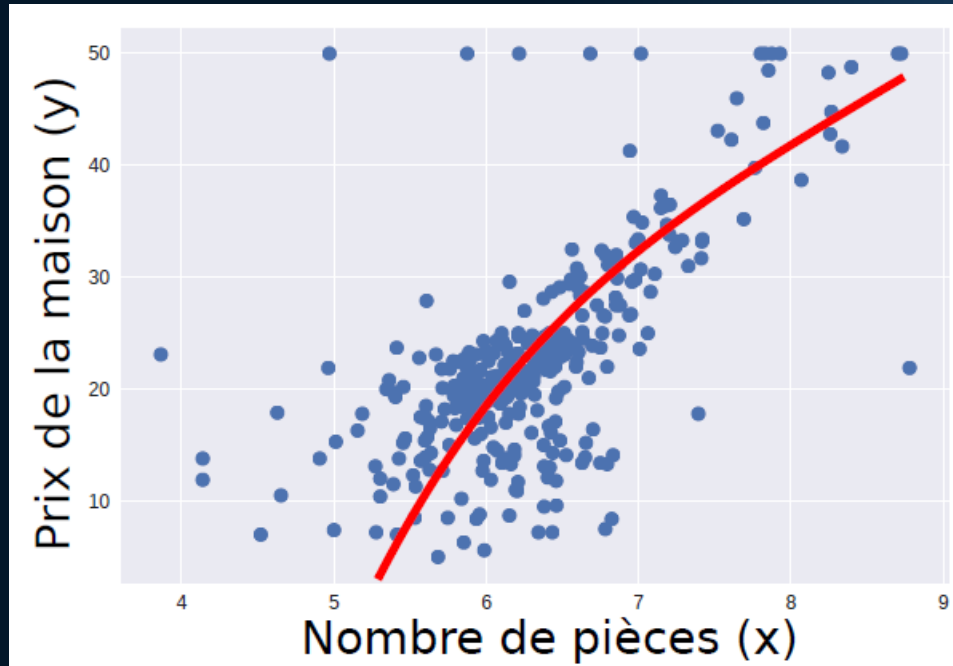
Hypothèse : droite linéaire

$$\hat{y} = h(x) = w * x + b$$



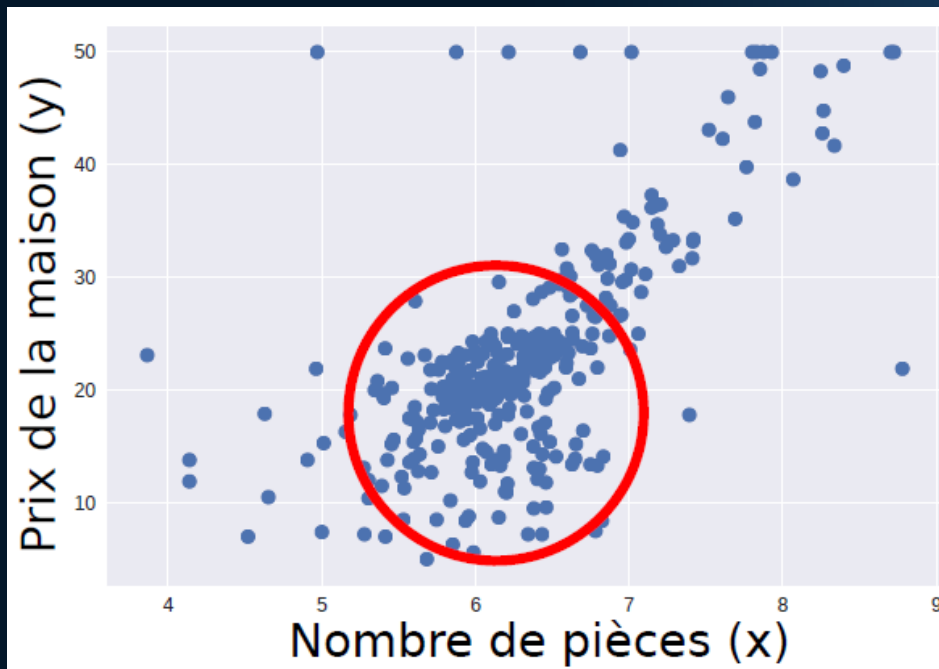
Hypothèse : courbe

$$\hat{y} = h(x) = w_1 * x^2 + w_2 * x + b$$



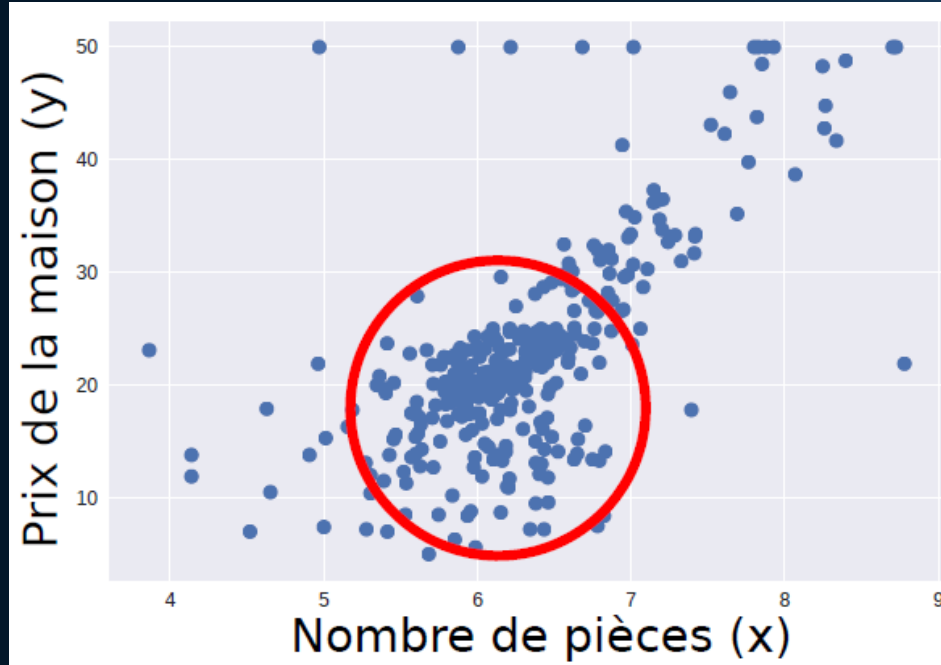
Hypothèse : cercle

$$\hat{y} = h(x) = \pm \sqrt{x^2 - b^2}$$



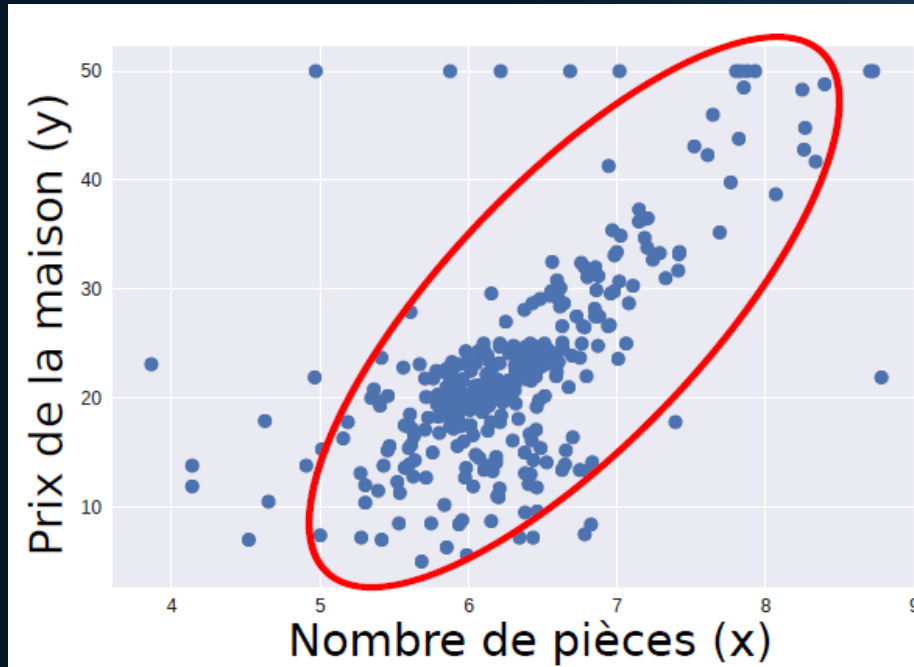
Hypothèse : ellipse

$$\hat{y} = h(x) = \pm \sqrt{1 - w_1 * x^2}$$



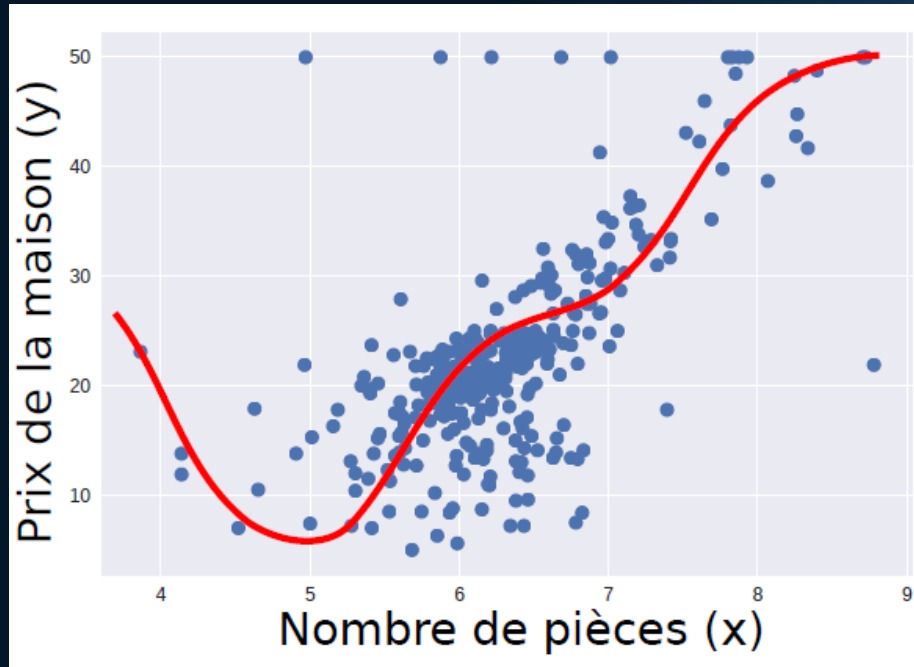
Hypothèse : ellipse

$$\hat{y} = h(x) = \pm \sqrt{1 - w_1 * x^2}$$



Hypothèse : fonction complexe

$$\hat{y} = h(x) = f(x, w_1, w_2, \dots, w_n)$$



Définir un objectif

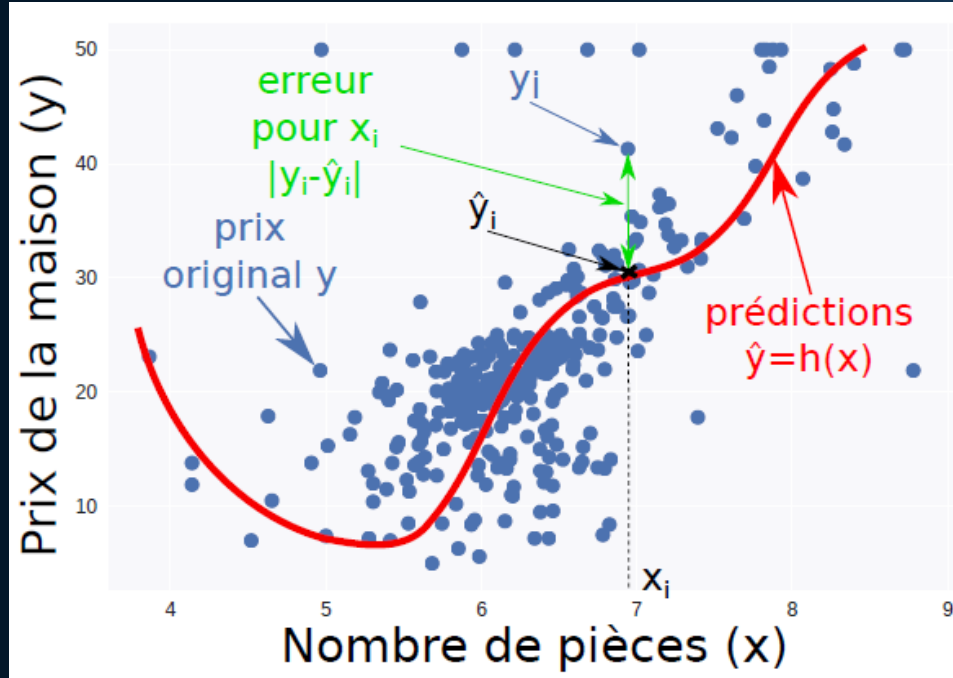
Fonction de coût :

$$L(h)$$

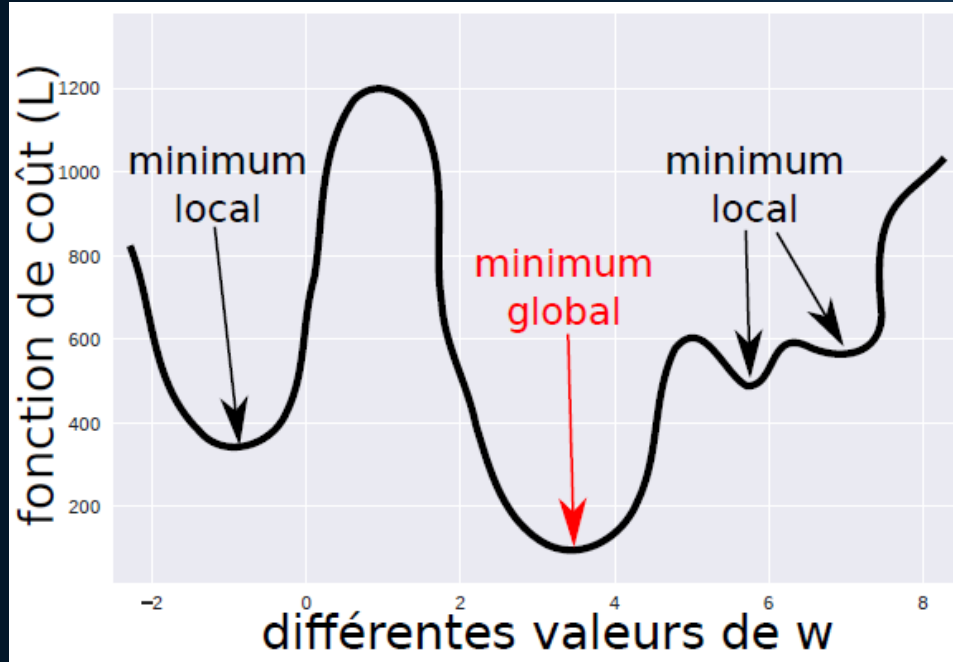
Objectif :

$$\min_h L(h)$$

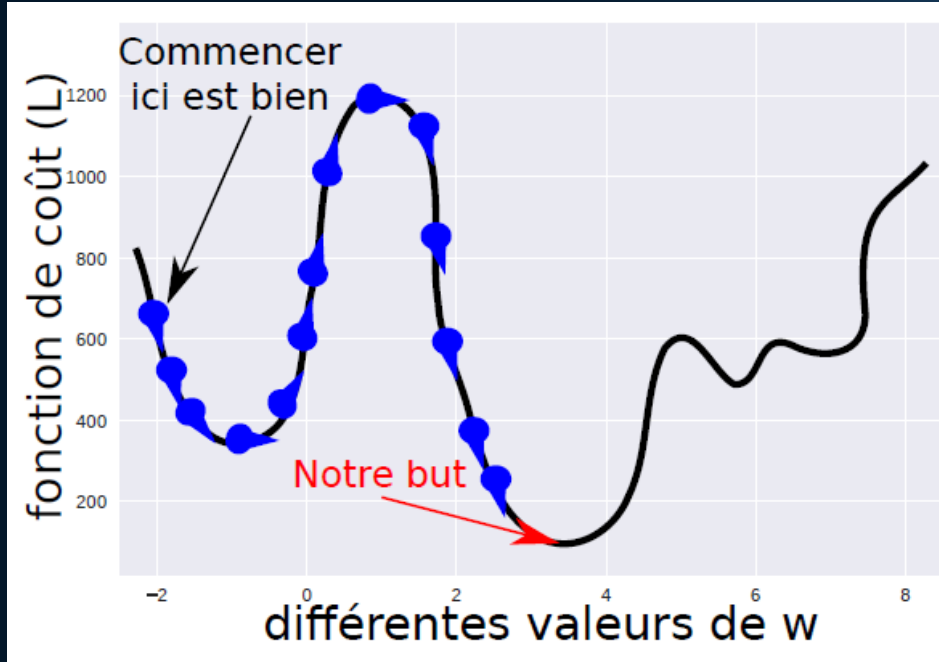
Définition de l'objectif est indépendante du choix du modèle



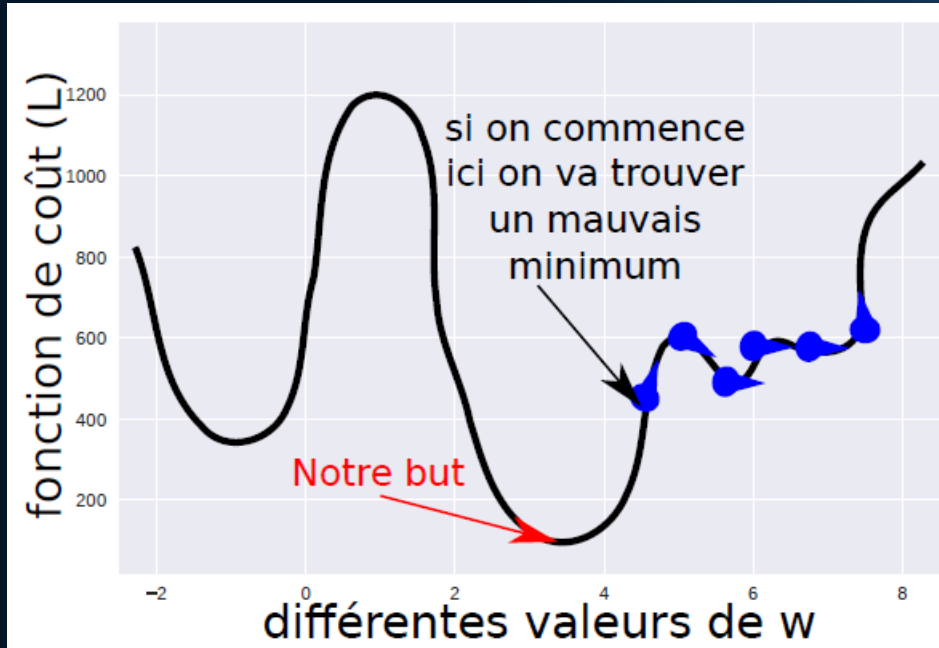
Minimiser une fonction de coût



Méthode par force brute



Méthode par force brute



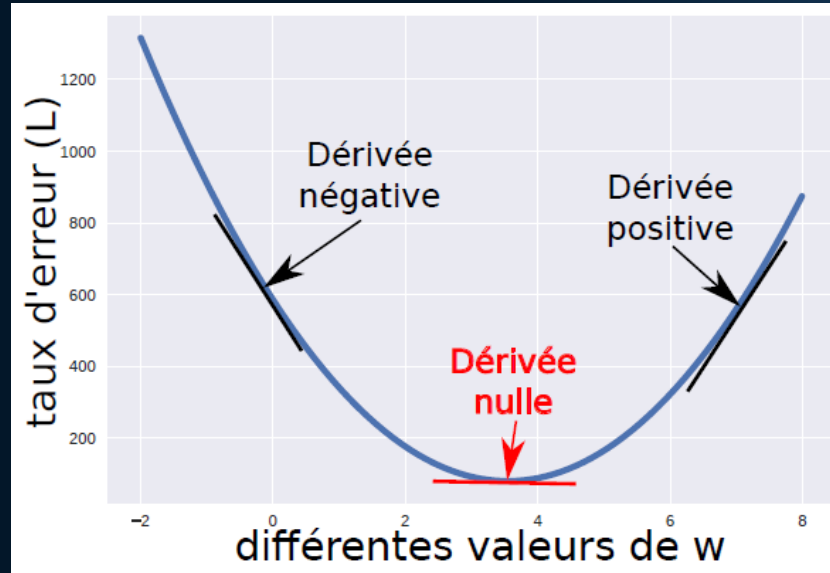
Méthode par force brute

Ne pas appliquer la force brute car :

- Méthode très lente (si on a $w_1, w_2, \dots, w_{1000}$)
- Difficile de savoir avec quel w commencer
- On peut tomber sur un mauvais minimum

Dérivée (ou gradient)

Dérivée augmentation partielle (très faible) de w



Solution optimale

But : trouver w pour

$$\frac{dL}{dw} = 0$$

Plusieurs méthodes existent :

- Inverser les équations normales
- Décompositions orthogonales

Limitations :

- Méthodes très lentes surtout si le modèle est complexe
- Problème si plusieurs solutions existent pour l'équation ci-dessus

Table des matières

01

Préliminaires

Minimiser la fonction
de coût

02

Descente du gradient

Ajuster les poids

03

Régression linéaire

Démonstrations
mathématiques
appliquées à un
problème linéaire

04

Prédiction du prix des maisons

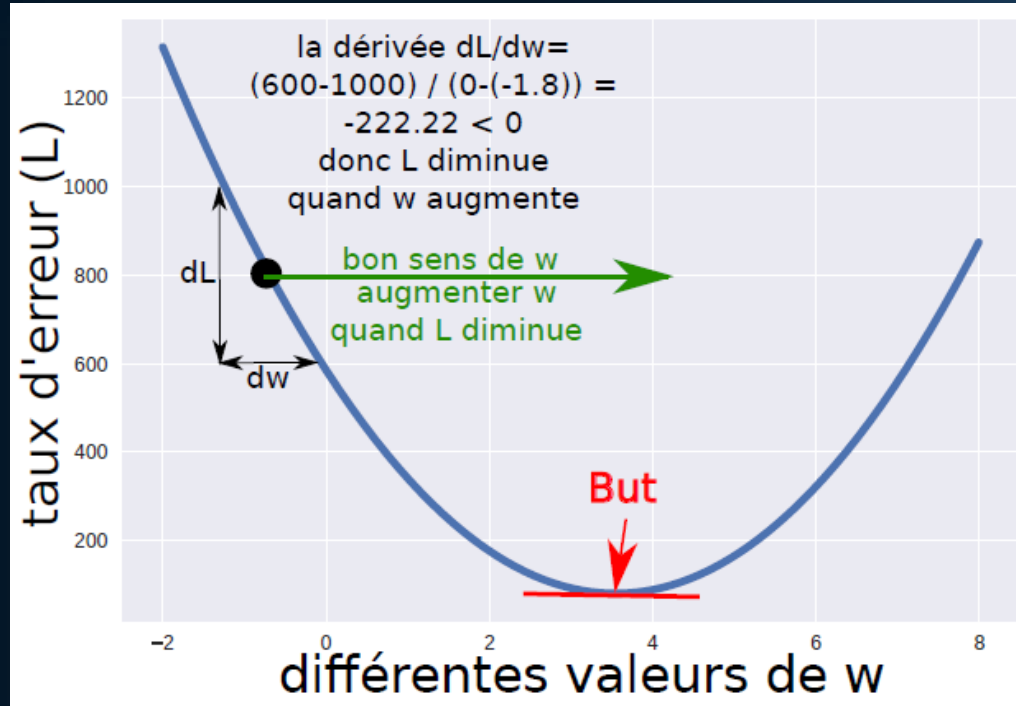
Exercice pratique

Analogie de la descente du gradient

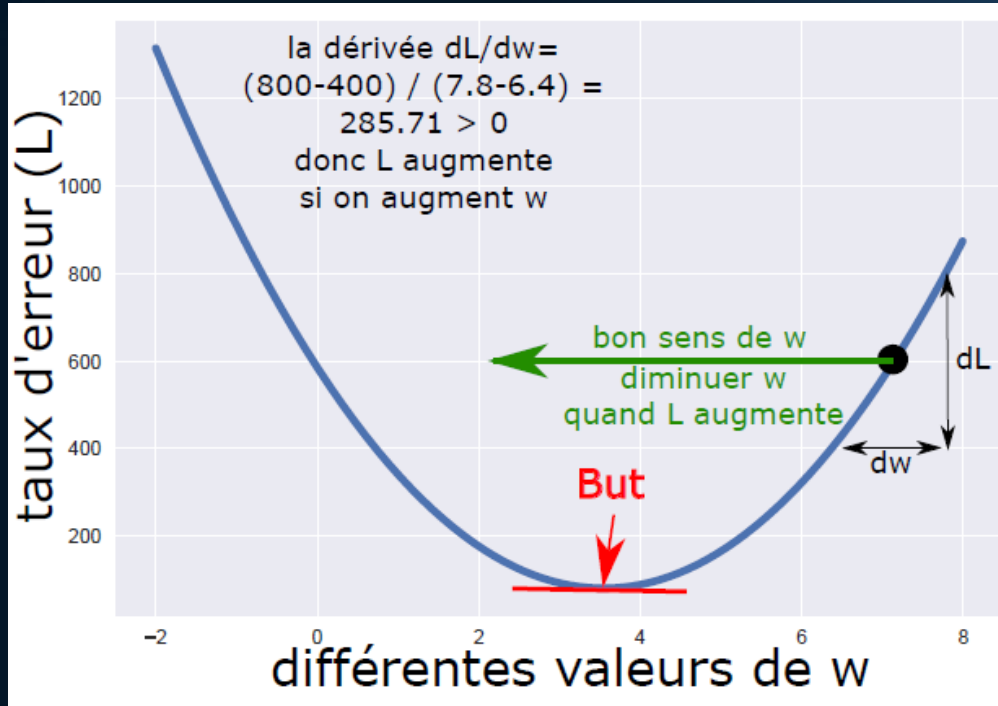


<https://www.kdnuggets.com/2018/06/intuitive-introduction-gradient-descent.html>

Le coût diminue



Le coût augmente



Comment varier w ?

Comment varier w ?

- Si w augmente, le taux d'erreur L augmente
 - On est dans une montée
 - Il faut diminuer w
- Si w augmente, le taux d'erreur L diminue
 - On est dans une descente
 - Il faut augmenter w

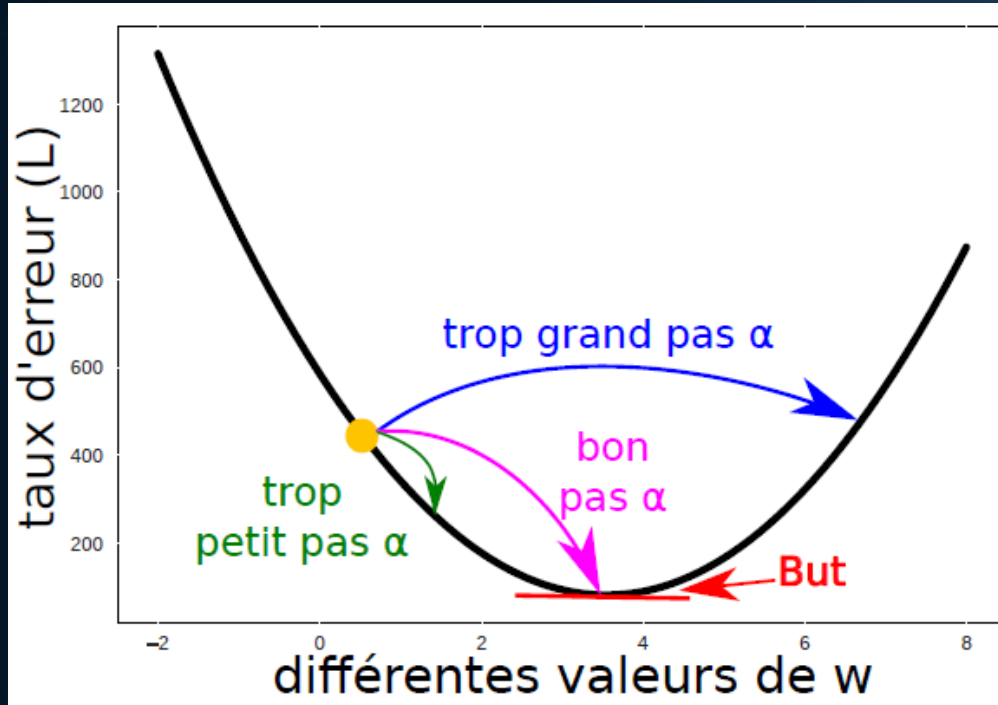
Ok maintenant formalisons tout cela

Comment varier w ?

$$w = w - \alpha \frac{dL}{dw}$$

- w : l'ensemble des paramètres à apprendre
- L : la fonction de coût qu'on veut minimiser
- α : le taux d'apprentissage qui contrôle la variation de w
- $-$: (le signe moins) signifie qu'on varie w dans le sens inverse de la variation de L \Rightarrow on minimise l'erreur

Pourquoi un taux d'apprentissage α ?



Bon choix de α

Comment choisir α ?

- Un très grand α peut amener à rater l'objectif
- Un très petit α peut amener à une convergence trop lente nécessitant trop d'itérations

⇒ Il faut choisir α pour avoir un équilibre entre le temps de calcul disponible et la précision du minimum qu'on veut atteindre

⇒ Un choix souvent empirique et basé sur les travaux d'autres personnes

Notion de mini-batch

- L'algorithme « *Batch Gradient Descent* » optimise la fonction :

$$L(w) = \frac{1}{n} \sum_{i=1}^n \text{erreur}(y_i, \hat{y}_i)$$

- Il faut calculer l'erreur sur l'ensemble du jeu d'entraînement ce qui est problématique si n est très grand

⇒ Notion de « mini-batch »

⇒ Calculer L pour $m \ll n$

$$L(w) = \frac{1}{m} \sum_{i=s}^{s+m-1} \text{erreur}(y_i, \hat{y}_i) \quad \forall s \in \{1, m, 2m, \dots, N - m\}$$

- Une fois qu'on a passé tous les mini-batches = une époque (*epoch* en anglais)
- On répète le processus d'entraînement sur plusieurs époques

Pseudo-code pour mini-batch gradient descent

Entrées : X, Y l'ensemble du jeu de données d'entraînement

Sortie : W les paramètres qui modélisent le mieux la relation $X \Rightarrow Y$

Initialisation aléatoire de W

- 1: **Pour** $epoque = 1$ à 1000 **faire**
- 2: **Pour** chaque mini-batch X_m de taille m **faire**
- 3: $predictions = model(X_m, W)$
- 4: $erreurs = fonction_{erreur}(Y_m, predictions)$
- 5: $gradient = \frac{dL(W)}{dW}$
- 6: $W = W - \alpha * gradient$
- 7: **fin Pour**
- 8: **fin Pour**
- 9: **Renvoyer** W

Variants de la méthode de descente du gradient

Soit m la taille du mini-batch

- Si $m = n \Rightarrow$ *Batch Gradient Descent*
 - Il faut passer sur tout l'ensemble pour mettre w à jour
- Si $m = 1 \Rightarrow$ *Stochastic Gradient Descent*
 - On perd la parallélisation du calcul sur plusieurs exemples
- Si $1 < m \ll n \Rightarrow$ *Minibatch Gradient Descent*
 - On accélère une mise à jour de w en parallélisant sur m exemples

Table des matières

01

Préliminaires

Minimiser la fonction
de coût

02

Descente du gradient

Ajuster les poids

03

Régression linéaire

Démonstrations
mathématiques
appliquées à un
problème linéaire

04

Prédiction du prix des maisons

Exercice pratique

Fonction de coût

Forme générale

$$L(w) = \frac{1}{n} \sum_{i=1}^n \text{erreur}(y_i, \hat{y}_i)$$

L'erreur quadratique moyenne

$$L(w) = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i(w))^2$$

L'erreur quadratique moyenne pour la régression linéaire

$$L(w, b) = \frac{1}{n} \sum_{i=1}^n (y_i - (w * x_i + b))^2$$

Pour la régression linéaire : deux paramètres à apprendre w et b

Calcul du gradient par rapport à w pour un modèle linéaire

$$\frac{dL(w, b)}{dw} = \frac{d\left(\frac{1}{n} \sum_{i=1}^n \left(y_i - (w * x_i + b)\right)^2\right)}{dw}$$

$$\frac{dL(w, b)}{dw} = \frac{1}{n} \sum_{i=1}^n \frac{d\left(y_i - (w * x_i + b)\right)^2}{dw}$$

Or, $\frac{dy_i}{dw} = 0$ car y_i est indépendant de w :

$$\frac{dL(w, b)}{dw} = \frac{1}{n} \sum_{i=1}^n (-2x_i(y_i - (w * x + b)))$$

$$\frac{dL(w, b)}{dw} = -\frac{2}{n} \sum_{i=1}^n (x_i(y_i - \hat{y}_i))$$

Mise à jour de w :

$$w = w - \alpha * \left(-\frac{2}{n} \sum_{i=1}^n (x_i(y_i - \hat{y}_i))\right)$$

Calcul du gradient par rapport à b pour un modèle linéaire

$$\frac{dL(w, b)}{db} = \frac{d\left(\frac{1}{n} \sum_{i=1}^n (y_i - (w * x_i + b))^2\right)}{db}$$

$$\frac{dL(w, b)}{db} = \frac{1}{n} \sum_{i=1}^n \frac{d\left(y_i - (w * x_i + b)\right)^2}{db}$$

Or, $\frac{dy_i}{db} = 0$ car y_i est indépendant de b :

$$\frac{dL(w, b)}{db} = \frac{1}{n} \sum_{i=1}^n (-2(y_i - (w * x + b)))$$

$$\frac{dL(w, b)}{db} = -\frac{2}{n} \sum_{i=1}^n (y_i - \hat{y}_i)$$

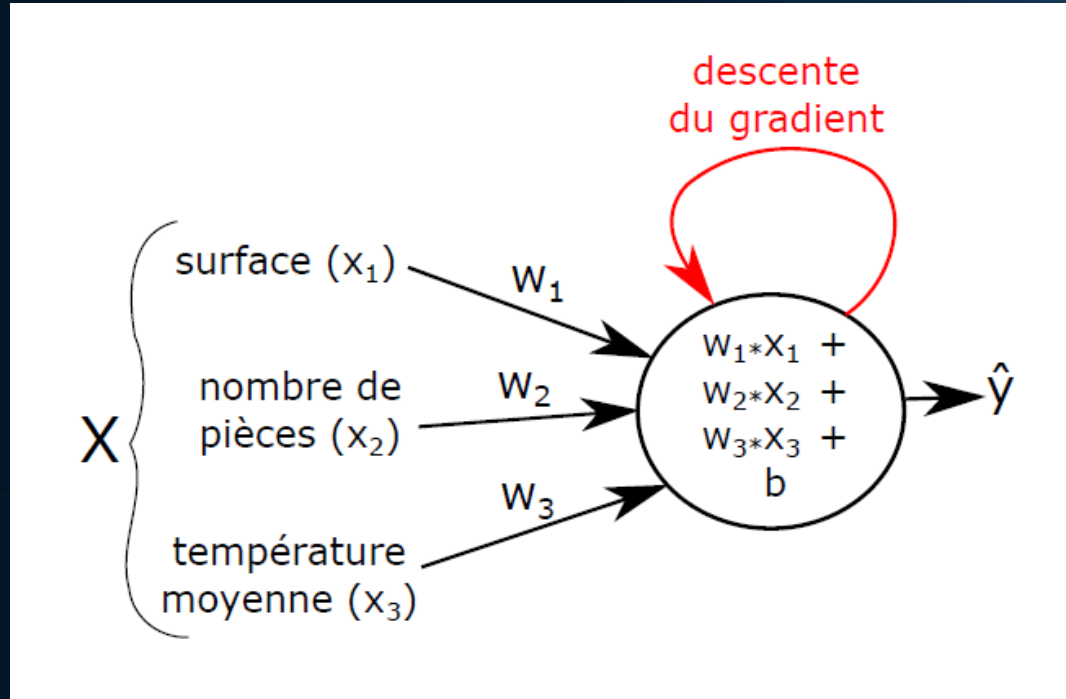
Mise à jour de b :

$$b = b - \alpha * \left(-\frac{2}{n} \sum_{i=1}^n (y_i - \hat{y}_i)\right)$$

Régression linéaire avec plusieurs variables

- Si l'instance possède plusieurs caractéristiques
- **Exemple** : prédire le prix d'une maison à partir nombre de pièces, population du quartier, le climat de la région, etc.
- Au lieu d'un seul w on aura $W = (w_1, w_2, \dots, w_r)$ pour r caractéristiques différentes
- On répète le même processus pour chaque $w_i \in W$
- Le modèle devient alors : $\hat{y}_i = b + \sum_{j=1}^r w_j * x_{i,j}$
- Avec $x_{i,j}$ la $j^{\text{ième}}$ caractéristique du $i^{\text{ième}}$ individu de l'ensemble X d'entraînement

Exemple de régression linéaire avec un neurone (Perceptron)



Normalisation des données en entrée

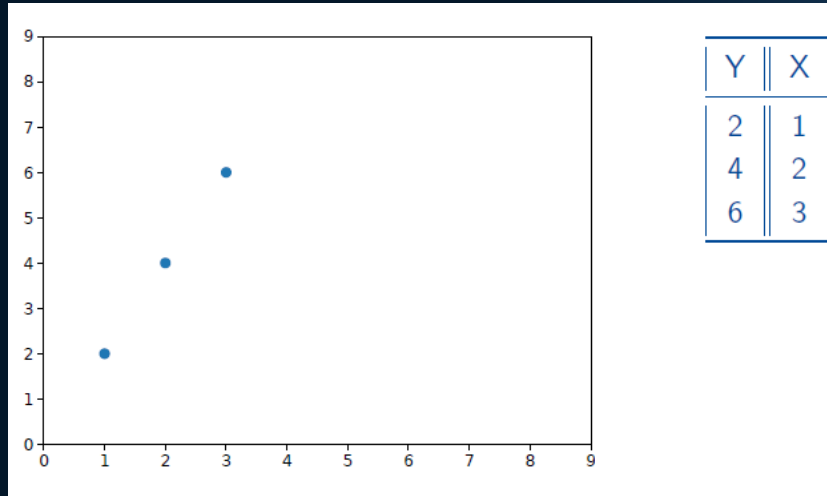
- Consiste à rendre chaque caractéristique entre 0 et 1
- C'est important pour que toutes les caractéristiques aient la même contribution
- Par exemple :
 - La surface d'une maison est entre 0 et 1000 m^2
 - Le nombre de pièce est entre 0 et 10

Sans normalisation \Rightarrow une petite variation du nombre de pièces n'a pas un grand effet par rapport aux valeurs énormes de la surface \Rightarrow le modèle aura des difficultés à apprendre

$$X_{:,i} = \frac{X_{:,i} - \min(X_{:,i})}{\max(X_{:,i}) - \min(X_{:,i})} \quad \forall i \in [1, \dots, r]$$

r étant le nombre de caractéristiques dans le jeu de données

Exercice (1/2)



Exercice : Trouver w en fixant $b = 0$ et $\alpha = 0.2$ et initialiser $w = 1$
Il faut qu'on arrive à une valeur proche de $w = 2$ (la solution optimale)

Exercice (2/2)

Première époque

$$\text{erreur} = \frac{1}{3}[(2 - 1 + 1)^2 + (4 - 1 * 2)^2 + (6 - 1 * 3)^2] \approx 4.67$$

$$\text{gradient} = \frac{-2}{3}[1(2 - 1) + 2(4 - 2) + 3(6 - 3)] \approx -9.33$$

$$w = 1 - 0.2(-9.33) \approx 1.87$$

Deuxième époque

$$\text{erreur} = \frac{1}{3}[(2 - 1.87 + 1)^2 + (4 - 1.87 * 2)^2 + (6 - 1.87 * 3)^2] \approx 0.08$$

$$\text{gradient} = \frac{-2}{3}[1(2 - 1.87) + 2(4 - 3.74) + 3(6 - 5.61)] \approx -1.21$$

$$w = 1.87 - 0.2(-1.21) \approx 2.112$$

On s'approche de $w = 2$ (la solution optimale)

Table des matières

01

Préliminaires

Minimiser la fonction
de coût

02

Descente du gradient

Ajuster les poids

03

Régression linéaire

Démonstrations
mathématiques
appliquées à un
problème linéaire

04

Prédiction du prix des maisons

Exercice pratique

Boston house prices (régression linéaire avec toutes les caractéristiques)

Prix	CRIM	ZN	INDUS	CHAS	NOX	RM	...
18.2	0.7258	0.	8.14	0.	0.538	5.727	...
20.3	0.3494	0.	9.9	0.	0.544	5.972	...
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

- Chaque maison possède un prix et 13 autres caractéristiques
- Une maison est représentée par le vecteur $x_i = (x_{i,1}, x_{i,2}, \dots, x_{i,13})$
- 13 attributs \Rightarrow 13 w_i et 1 b
- Soit un vecteur $W = (w_1, w_2, \dots, w_{13})$

Boston house prices (régression linéaire avec deux caractéristiques)

Prix	CRIM	ZN	INDUS	CHAS	NOX	RM	...
18.2	0.7258	0.	8.14	0.	0.538	5.727	...
20.3	0.3494	0.	9.9	0.	0.544	5.972	...
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Prix (y)	RM ($X_{:,1}$)	LSTAT ($X_{:,2}$)
18.2	5.727	0.18
20.3	5.972	0.25
⋮	⋮	⋮

Boston house prices (régression linéaire avec deux caractéristiques)

But : prédire le prix en fonction des deux variables

$$prix = f(X) \mid X = [X_{:,1}, X_{:,2}]$$

Modèles à apprendre

$$\hat{y} = W * X + B \Leftrightarrow \hat{y} = w_1 * X_{:,1} + w_2 * X_{:,2} + b$$

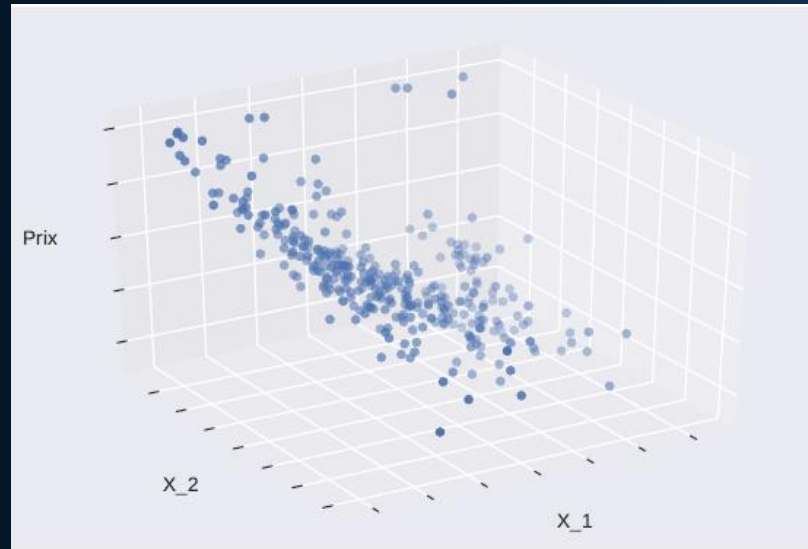
Cette équation correspond à celle d'un plan dans un espace 3D

$$z = w_1 * x + w_2 * y + b$$

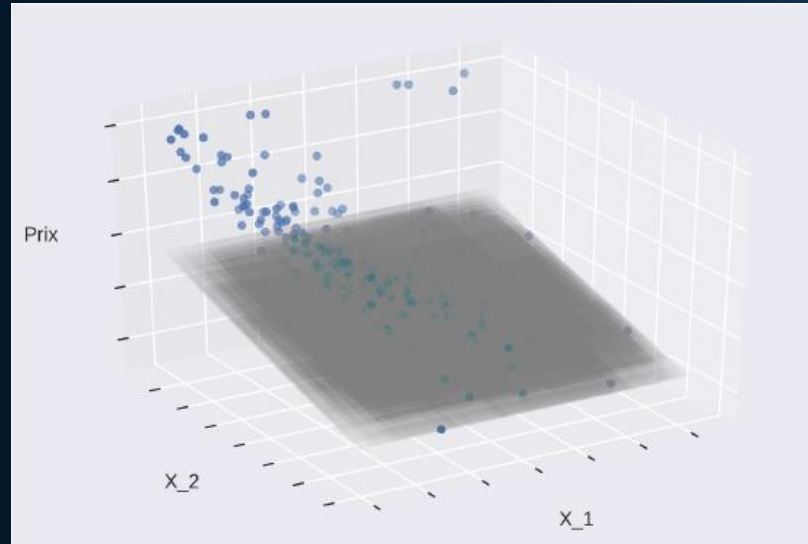
Avec

$$x = X_{:,1}, y = X_{:,2}, z = \hat{y}$$

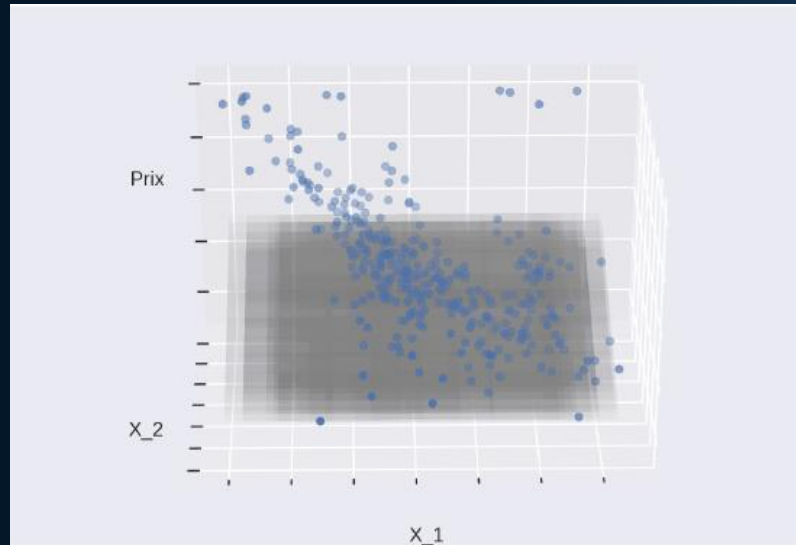
Boston house prices (régression linéaire avec deux caractéristiques)



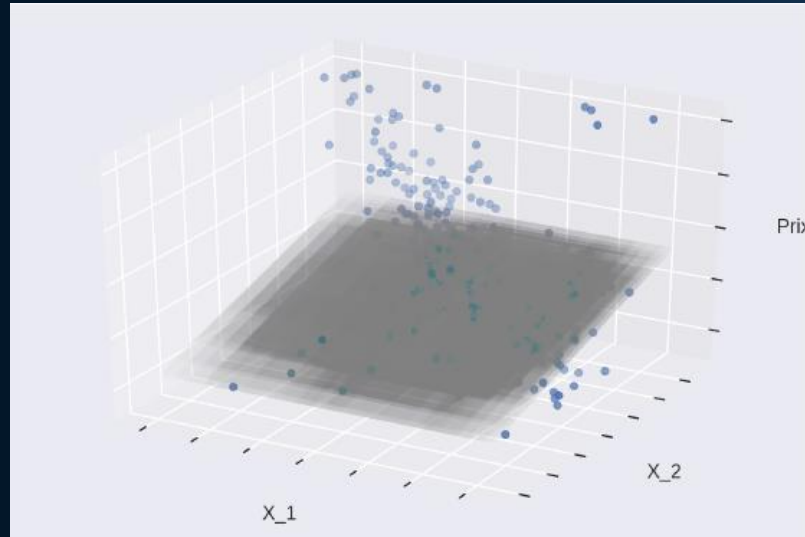
Boston house prices (régression linéaire avec deux caractéristiques)



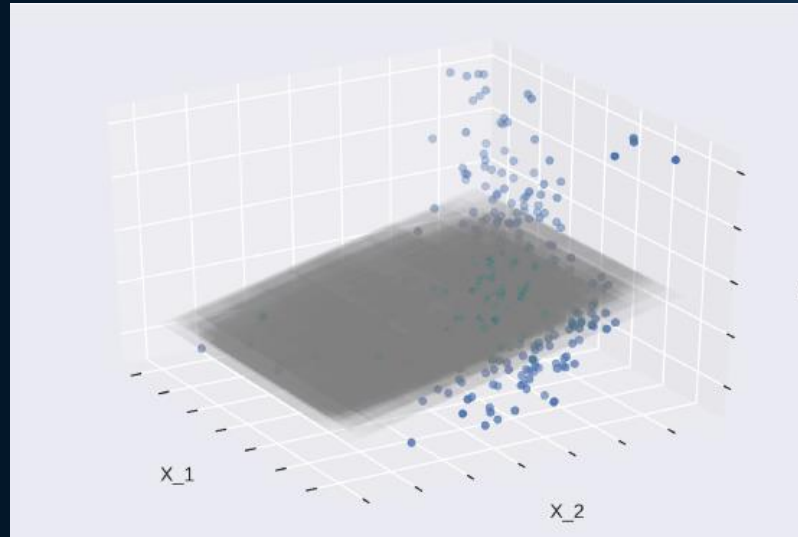
Boston house prices (régression linéaire avec deux caractéristiques)



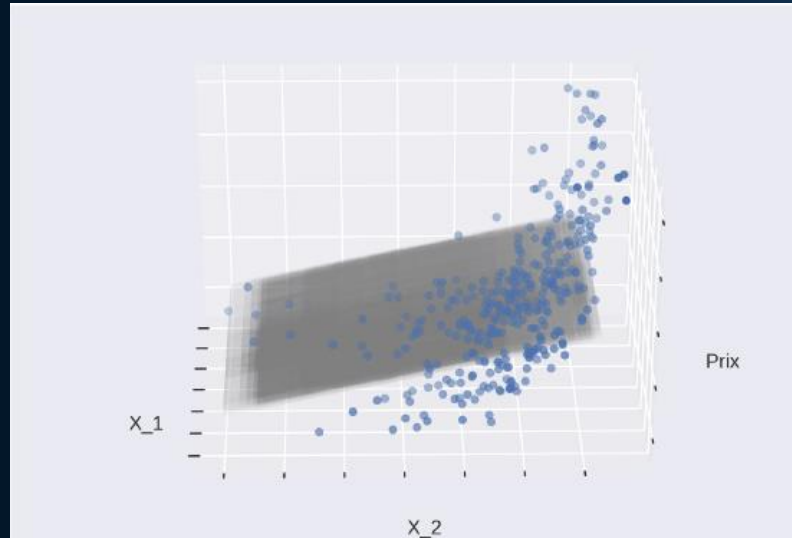
Boston house prices (régression linéaire avec deux caractéristiques)



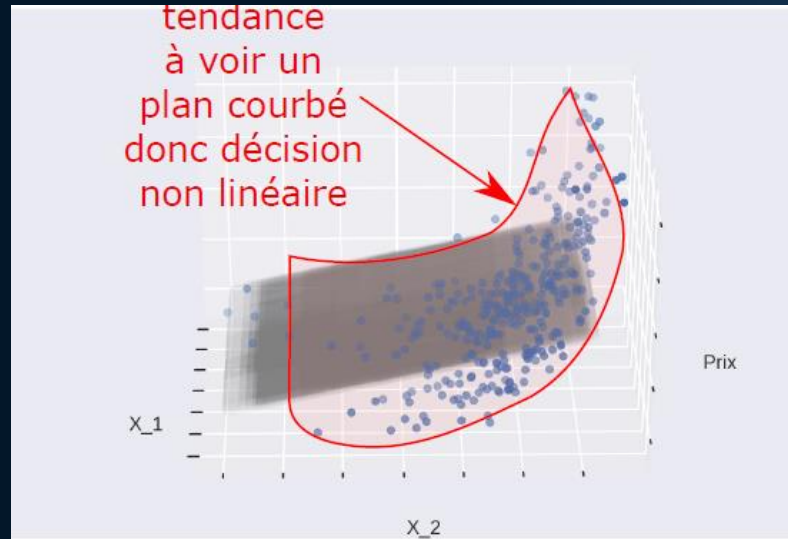
Boston house prices (régression linéaire avec deux caractéristiques)



Boston house prices (régression linéaire avec deux caractéristiques)



Boston house prices (régression linéaire avec deux caractéristiques)



Boston house prices (peut-être prédiction non linéaire ?)

- Non linéaire \Rightarrow une interaction non linéaire entre les caractéristiques
- Par exemple : $x_1 * x_2$ ou x_1^2

$$\hat{y} = w_1 * X_{:,1} + w_2 * X_{:,2} + w_3 * X_{:,3} + b$$

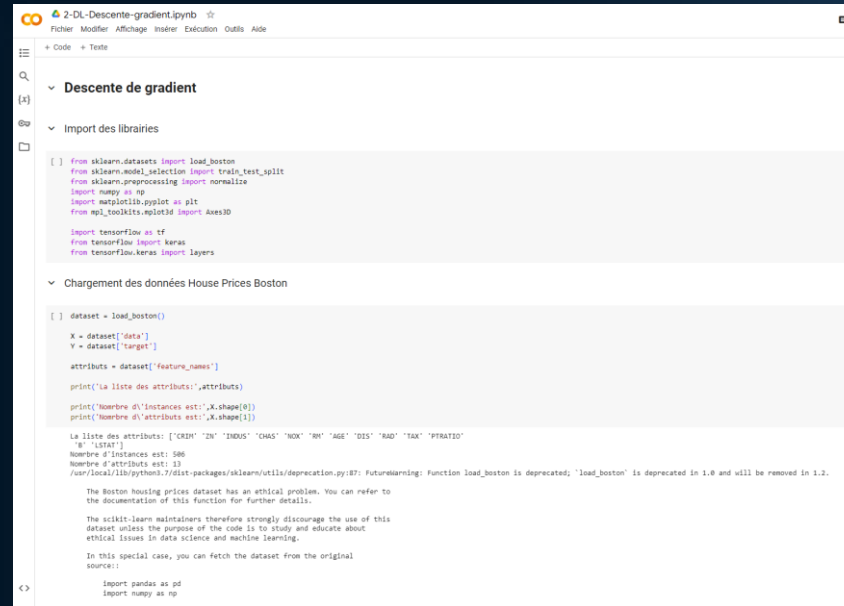
Avec

$$X_{:,3} = X_{:,1} * X_{:,2}$$

Les données deviennent ...

Prix (y)	RM ($X_{:,1}$)	LSTAT ($X_{:,2}$)	RM*LSTAT($X_{:,3}$)
18.2	5.727	0.18	5.727*0.18=1.03
20.3	5.972	0.25	5.972*0.25=1.493
\vdots	\vdots	\vdots	\vdots

De la théorie vers la pratique



```
from sklearn.datasets import load_boston
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import normalize
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers

dataset = load_boston()
X = dataset['data']
Y = dataset['target']
attributs = dataset['feature_names']

print('La liste des attributs:', attributs)
print('Nombre d\'instances est:', X.shape[0])
print('Nombre d\'attributs est:', X.shape[1])

La liste des attributs: ['CRIM', 'IND', 'NOX', 'AGE', 'DIS', 'RAD', 'TAX', 'PTRATIO',
                        'B', 'LSTAT']
Nombre d'instances est: 506
Nombre d'attributs est: 13
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning: Function load_boston is deprecated; 'load_boston' is deprecated in 1.0 and will be removed in 1.2.
The Boston housing prices dataset has an ethical problem. You can refer to
the documentation of this function for further details.
The scikit-learn maintainers therefore strongly discourage the use of this
dataset unless the purpose of the code is to study and educate about
ethical issues in data science and machine learning.
In this special case, you can fetch the dataset from the original
source:
import pandas as pd
import numpy as np
```

Télécharger le fichier ([https://github.com/r-wenger/cours_ml-m2-OTG/blob/main/IA_geosciences M2/CM/2_DL_Descente_gradient.i
pynb](https://github.com/r-wenger/cours_ml-m2-OTG/blob/main/IA_geosciences_M2/CM/2_DL_Descente_gradient.ipynb)) et l'importer sur Google Colab