
# Théorie et Algorithmes de l'Apprentissage Automatique

## 2 - Régression linéaire

Simon BERNARD
simon.bernard@univ-rouen.fr

# Introduction

## Premier exemple : Estimer l'altitude avec un thermomètre

- Expérience de Joseph D. Hooker en 1849
- Mesure de pression atmosphérique $p_i$ et de température d'ébullition de l'eau $t_i$ dans l'Himalaya
- Les lois de la physique disent que $y_i = \ln(p_i)$ est (approx.) proportionnel à $t_i$ :

$$y_i = \alpha t_i + \beta + u_i$$

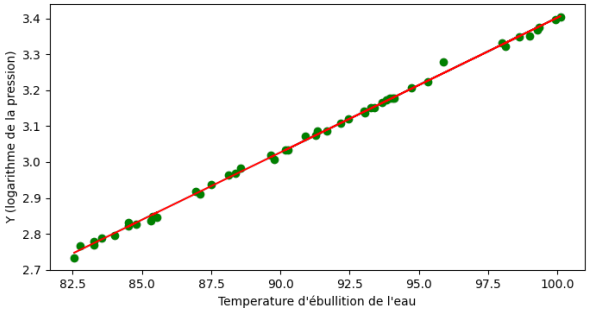
où $u_i$ représente l'erreur de mesure (ou bruit)

- Objectif : Prédire la pression atmosphérique à partir de la température d'ébullition de l'eau (ce qui permet ensuite de déduire l'altitude, sans utiliser de baromètre)

## Premier exemple : Estimer l'altitude avec un thermomètre

Voici le tracé des points représentant les mesures, ainsi qu'une estimation de la droite :

$$y_i = wt_i + b$$




Les paramètres $(w, b)$ sont estimés en minimisant l'erreur quadratique (moindre carrés)

- La droite est un modèle de regression linéaire
- Elle explique au mieux une grandeur $Y$ (variable cible) en fonction de $d$ autres grandeurs $\{X^{(j)}\}_{j=1..d}$ (variables explicatives)
- Elle permet également d'étudier l'erreur irreductible, i.e. le bruit $u_i$, e.g. on peut supposer que $u_i$ suit une distribution gaussienne centrée et estimer l'écart-type $\sigma$
- Elle permet de prédire n'importe quelle valeur de pression étant donné la température d'ébullition de l'eau

## Deuxième exemple : Prédire la progression du diabète à partir de données cliniques [1]

| | AGE | SEX | BMI | BP | ··· Serum Measurements ··· | | | | | | Response |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Patient | x1 | x2 | x3 | x4 | x5 | x6 | x7 | x8 | x9 | x10 | y |
| 1 | 59 | 2 | 32.1 | 101 | 157 | 93.2 | 38 | 4 | 4.9 | 87 | 151 |
| 2 | 48 | 1 | 21.6 | 87 | 183 | 103.2 | 70 | 3 | 3.9 | 69 | 75 |
| 3 | 72 | 2 | 30.5 | 93 | 156 | 93.6 | 41 | 4 | 4.7 | 85 | 141 |
| 4 | 24 | 1 | 25.3 | 84 | 198 | 131.4 | 40 | 5 | 4.9 | 89 | 206 |
| 5 | 50 | 1 | 23.0 | 101 | 192 | 125.4 | 52 | 4 | 4.3 | 80 | 135 |
| 6 | 23 | 1 | 22.6 | 89 | 139 | 64.8 | 61 | 2 | 4.2 | 68 | 97 |
| ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ |
| 441 | 36 | 1 | 30.0 | 95 | 201 | 125.2 | 42 | 5 | 5.1 | 85 | 220 |
| 442 | 36 | 1 | 19.6 | 71 | 250 | 133.2 | 97 | 3 | 4.6 | 92 | 57 |

**Table 1.** Diabetes study. 442 diabetes patients were measured on 10 baseline variables. A prediction model was desired for the response variable, a measure of disease progression one year after baseline.

On suppose que la relation entre les entrées et la sortie est linéaire :

$$y_i = w_1 x_i^{(1)} + w_2 x_i^{(2)} + \cdots + w_d x_i^{(10)} + b + u_i$$

1. Bradley Efron et al., “Least Angle Regression”, Annals of Statistics, 2004

Deuxième exemple : Prédire la progression du diabète à partir de données cliniques [2]

- Le modèle $h : \mathbb{R}^d \to \mathbb{R}$ à déterminer est de la forme

$$h(\mathbf{x}) = \sum_{i=1}^{d} w_i x^{(i)} + b = \mathbf{x}^\top \mathbf{w} + b = [\mathbf{x}^\top 1]\boldsymbol{\alpha}$$

avec

- $\mathbf{w} \in \mathbb{R}^d$, un vecteur qui définit un hyperplan
- $b \in \mathbb{R}$ un biais qui déplace la fonction perpendiculairement à l'hyperplan
- $\boldsymbol{\alpha} = \begin{bmatrix} \mathbf{w} \\ b \end{bmatrix} \in \mathbb{R}^{d+1}$

- L'objectif est de déterminer $(\mathbf{w}, b)$ à partir de l'ensemble d'apprentissage $\mathcal{D}$

2. Bradley Efron et al., "Least Angle Regression", Annals of Statistics, 2004

Deuxième exemple : Prédire la progression du diabète à partir de données cliniques [3]

```
from sklearn import datasets
from sklearn.linear_model import LinearRegression

X, y = datasets.load_diabetes(scaled=False, return_X_y=True)
clf = LinearRegression().fit(X, y)
print(clf.coef_)
print(clf.intercept_)
```

```
> [-3.63612242e-02 -2.28596481e+01  5.60296209e+00  1.11680799e+00
   -1.08999633e+00  7.46450456e-01  3.72004715e-01  6.53383194e+00
    6.84831250e+01  2.80116989e-01]
  -334.5671385187874
```

3. Bradley Efron et al., “Least Angle Regression”, Annals of Statistics, 2004

# Méthode des moindres carrés

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_1^\top & 1 \\ \mathbf{x}_2^\top & 1 \\ \vdots & \vdots \\ \mathbf{x}_i^\top & 1 \\ \vdots & \vdots \\ \mathbf{x}_n^\top & 1 \end{bmatrix} = \begin{bmatrix} x_1^{(1)} & x_1^{(2)} & \dots & x_1^{(j)} & \dots & x_1^{(d)} & 1 \\ x_2^{(1)} & x_2^{(2)} & \dots & x_2^{(j)} & \dots & x_2^{(d)} & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_i^{(1)} & x_i^{(2)} & \dots & x_i^{(j)} & \dots & x_i^{(d)} & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_n^{(1)} & x_n^{(2)} & \dots & x_n^{(j)} & \dots & x_n^{(d)} & 1 \end{bmatrix}, \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_i \\ \vdots \\ y_n \end{bmatrix}$$

- $\mathbf{x}_i \in \mathbb{R}^d$ sont les observations (instances) pour $i = 1, \dots, n$
- $y_i \in \mathbb{R}$ sont les valeurs observées à prédire (réponse) pour $i = 1, \dots, n$
- $\mathbf{X} \in \mathbb{R}^{n \times (d+1)}$ telle que $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n, \mathbf{e}]^\top$ avec $\mathbf{e} \in \mathbb{R}^d$ et $e_i = 1, \forall i$
- $\mathbf{y} \in \mathbb{R}^n$ telle que $\mathbf{y} = [y_1, y_2, \dots, y_n]^\top$

- Sous sa forme matricielle, la relation s'écrit

$$\mathbf{y} = \mathbf{X}\boldsymbol{\alpha} + \mathbf{u}$$

où $\mathbf{u}$ est un vecteur de bruit

- On suppose que $\mathbf{X}$ est de rang colonnes plein [4]
- Dans ce cas, $\mathbf{X}^\top\mathbf{X}$ est inversible, nécessaire pour la méthode des moindres carrés
- Dans le cas contraire, cela signifie qu'une des variables s'obtient par combinaison linéaire des autres et qu'on ne peut pas estimer $\boldsymbol{\alpha}^*$ sans hypothèses supplémentaires.

4. Toutes les colonnes sont linéairement indépendantes

- On veut estimer les paramètres $(\mathbf{w}, b)$ tels que

$$h(\mathbf{x}) = \mathbf{x}^\top \mathbf{w} + b$$

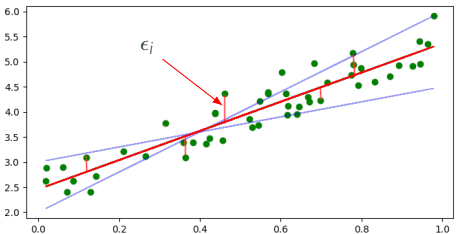
- Pour cela, on cherche à minimiser les erreurs de prédiction sur les exemples d'apprentissage, aussi appelé résidus :

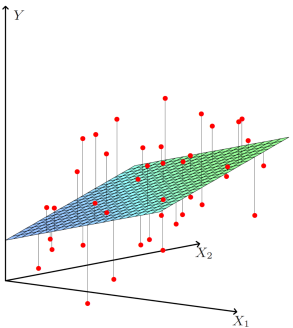
$$\epsilon_i = y_i - h(\mathbf{x}_i) = y_i - \mathbf{x}_i^\top \mathbf{w} - b$$

ou, sous la forme matricielle $\boldsymbol{\epsilon} \in \mathbb{R}^n$ :

$$\boldsymbol{\epsilon} = \mathbf{y} - \mathbf{X}\boldsymbol{\alpha}$$

Ce problème peut s'interpréter comme la recherche de l'hyperplan $y = \mathbf{x}^\top \mathbf{w} + b$ qui minimise les distances des observations $(\mathbf{x}_i, y_i), i = 1, \ldots, n$ à l'hyperplan



- La méthode des moindres carrés consiste à minimiser la somme des résidus au carré :

$$\min_{h} \quad \sum_{i=1}^{n} (y_i - h(\mathbf{x}_i))^2$$

$$\min_{(\mathbf{w},b)} \quad \sum_{i=1}^{n} (y_i - \mathbf{x}_i^\top \mathbf{w} - b)^2$$

$$\min_{(\mathbf{w},b)} \quad \sum_{i=1}^{n} \epsilon_i^2$$

- Sous sa forme matricielle :

$$\min_{\boldsymbol{\alpha}} \quad \|\boldsymbol{\epsilon}\|^2 = \|\mathbf{y} - \mathbf{X}\boldsymbol{\alpha}\|^2$$

où $\|.\|$ est la norme euclidienne d'un vecteur telle que $\|\boldsymbol{\epsilon}\|^2 = \sum_{i=1}^{n} \epsilon_i^2$

- Nous voulons résoudre le problème d'optimisation [5] :

$$\min_{\boldsymbol{\alpha}} J(\boldsymbol{\alpha}) \quad avec \quad J(\boldsymbol{\alpha}) = \frac{1}{2}\|\mathbf{y} - \mathbf{X}\boldsymbol{\alpha}\|^2$$

  où la fonction $J(\boldsymbol{\alpha})$ est convexe [6].

- Rappel : dans ce cas, $\boldsymbol{\alpha}^*$ est un minimum de la fonction $J(\boldsymbol{\alpha})$ si et seulement si :

$$\nabla J(\boldsymbol{\alpha}^*) = \mathbf{0}$$

  où $\nabla J(\boldsymbol{\alpha})$ est le gradient de la fonction en $\boldsymbol{\alpha}$ tel que :

$$\nabla J(\boldsymbol{\alpha}) = \frac{\partial J(\boldsymbol{\alpha})}{\partial \alpha_i}, \forall i$$

5. le terme $\frac{1}{2}$ simplifie les calculs mais il ne change rien au problème d'optimisation
6. $f(\mathbf{x}) = \mathbf{x}^2$ est convexe et les transformations affines préservent la convexité

- En développant :

$$\begin{aligned} J(\boldsymbol{\alpha}) &= \frac{1}{2}\|y - X\boldsymbol{\alpha}\|^2 = \frac{1}{2}(y - X\boldsymbol{\alpha})^\top (y - X\boldsymbol{\alpha}) \\ &= \frac{1}{2}(y^\top - (X\boldsymbol{\alpha})^\top)(y - X\boldsymbol{\alpha}) = \frac{1}{2}(y^\top - \boldsymbol{\alpha}^\top X^\top)(y - X\boldsymbol{\alpha}) \\ &= \frac{1}{2}y^\top y - \frac{1}{2}y^\top X\boldsymbol{\alpha} - \frac{1}{2}\boldsymbol{\alpha}^\top X^\top y + \frac{1}{2}\boldsymbol{\alpha}^\top X^\top X\boldsymbol{\alpha} \\ &= \frac{1}{2}y^\top y - \boldsymbol{\alpha}^\top X^\top y + \frac{1}{2}\boldsymbol{\alpha}^\top X^\top X\boldsymbol{\alpha} \end{aligned}$$

car $(A + B)^\top = A^\top + B^\top$, $(AB)^\top = B^\top A^\top$ et $y^\top X\boldsymbol{\alpha} = \boldsymbol{\alpha}^\top X^\top y$ (un scalaire).

$$\begin{aligned}\frac{\partial J(\boldsymbol{\alpha})}{\partial \alpha_i} &= \frac{\partial}{\partial \alpha_i}\left(\frac{1}{2}\mathbf{y}^\top\mathbf{y} - \boldsymbol{\alpha}^\top\mathbf{X}^\top\mathbf{y} + \frac{1}{2}\boldsymbol{\alpha}^\top\mathbf{X}^\top\mathbf{X}\boldsymbol{\alpha}\right)\\ &= \frac{\partial}{\partial \alpha_i}\frac{1}{2}\mathbf{y}^\top\mathbf{y} - \frac{\partial}{\partial \alpha_i}\boldsymbol{\alpha}^\top\mathbf{X}^\top\mathbf{y} + \frac{\partial}{\partial \alpha_i}\frac{1}{2}\boldsymbol{\alpha}^\top\mathbf{X}^\top\mathbf{X}\boldsymbol{\alpha}\end{aligned}$$

En posant $\mathbf{p} = \mathbf{X}^\top\mathbf{y}$ et $\mathbf{M} = \mathbf{X}^\top\mathbf{X}$, on a :

$$\frac{\partial}{\partial \alpha_i}\boldsymbol{\alpha}^\top\mathbf{p} = \frac{\partial}{\partial \alpha_i}\sum_{j=1}^{d+1} p_j\alpha_j = p_i$$

$$\frac{\partial}{\partial \alpha_i}\boldsymbol{\alpha}^\top\mathbf{M}\boldsymbol{\alpha} = \frac{\partial}{\partial \alpha_i}\sum_{j=1}^{d+1}\sum_{k=1}^{d+1}\alpha_j\alpha_k M_{jk} = \sum_{j=1}^{d+1}\alpha_j M_{ji} + \sum_{k=1}^{d+1}\alpha_k M_{ik}$$

Car $(uv)' = uv' + u'v$ avec $u = \alpha_j$ et $v = \sum_{k=1}^{d+1}\alpha_k M_{jk}$ [7]

7. détails du calcul en annexe, à la fin du support

$$\begin{aligned}\frac{\partial J(\boldsymbol{\alpha})}{\partial \alpha_i} &= \frac{\partial}{\partial \alpha_i}\left(\frac{1}{2}\mathsf{y}^\top\mathsf{y} - \boldsymbol{\alpha}^\top\mathsf{X}^\top\mathsf{y} + \frac{1}{2}\boldsymbol{\alpha}^\top\mathsf{X}^\top\mathsf{X}\boldsymbol{\alpha}\right)\\ &= \frac{\partial}{\partial \alpha_i}\frac{1}{2}\mathsf{y}^\top\mathsf{y} - \frac{\partial}{\partial \alpha_i}\boldsymbol{\alpha}^\top\mathsf{X}^\top\mathsf{y} + \frac{\partial}{\partial \alpha_i}\frac{1}{2}\boldsymbol{\alpha}^\top\mathsf{X}^\top\mathsf{X}\boldsymbol{\alpha}\\ &= 0 - p_i + \frac{1}{2}\sum_{j=1}^{d+1}(M_{ij} + M_{ji})\alpha_j\end{aligned}$$

Ce qui donne sous la forme matricielle :

$$\begin{aligned}\nabla J(\boldsymbol{\alpha}) &= -\mathsf{p} + \mathsf{M}\boldsymbol{\alpha}\\ &= -\mathsf{X}^\top\mathsf{y} + \mathsf{X}^\top\mathsf{X}\boldsymbol{\alpha}\end{aligned}$$

- Minimiser $J(\boldsymbol{\alpha})$ revient à trouver le $\boldsymbol{\alpha}$ qui annule le gradient :

$$\nabla J(\hat{\boldsymbol{\alpha}}) = 0 \quad \Leftrightarrow \quad -\mathbf{X}^\top \mathbf{y} + \mathbf{X}^\top \mathbf{X}\hat{\boldsymbol{\alpha}} = 0$$

- La solution du problème de minimisation des moindres carrés est le vecteur $\hat{\boldsymbol{\alpha}}$ défini par :

$$\hat{\boldsymbol{\alpha}} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y}$$

que l'on appelle l'estimateur des moindres carrés

```python
import numpy as np
from sklearn import datasets

X, y = datasets.load_diabetes(scaled=False, return_X_y=True)
Xa = np.concatenate((X, np.ones((X.shape[0], 1))), axis=1)
w = np.dot(np.linalg.inv(np.dot(Xa.T, Xa)), np.dot(Xa.T, y))
print(w[:-1])
print(w[-1])
```

```
>  [-3.63612242e-02 -2.28596481e+01  5.60296209e+00  1.11680799e+00
    -1.08999633e+00  7.46450456e-01  3.72004715e-01  6.53383194e+00
     6.84831250e+01  2.80116989e-01]
   -334.5671385187874
```

# Moindres carrés régularisés

- La méthode précédente s'appelle la méthode des moindres carrés "ordinaires"
- On cherche à minimiser :

$$\min_{\boldsymbol{\alpha}} J(\boldsymbol{\alpha}) \quad avec \quad J(\boldsymbol{\alpha}) = \frac{1}{2}\|\mathbf{y} - \mathbf{X}\boldsymbol{\alpha}\|^2$$

- La solution du problème est :

$$\hat{\boldsymbol{\alpha}} = (\mathbf{X}^\top \mathbf{X})^{-1}\mathbf{X}^\top \mathbf{y}$$

- Problème :
  - Lorsque $n < d + 1$, $\mathbf{X}^\top \mathbf{X}$ n'est pas inversible
  - La solution n'est pas unique : le problème est mal posé
- Solution : régularisation

- Régularisation : ajout d'une contrainte sur les paramètres à estimer

$$J(\boldsymbol{\alpha}) = \frac{1}{2}\|\mathbf{y} - \mathbf{X}\boldsymbol{\alpha}\|^2 + \lambda\Omega(\boldsymbol{\alpha})$$

- Vise à pénaliser les modèles trop complexes :
  - Modèle complexe : fort risque de sur-apprentissage
  - Plusieurs solutions : on favorise la solution la moins complexe
- L'influence de ce terme est contrôlée par un hyperparamètre de régularisation $\lambda$ ($\lambda = 0 \rightarrow$ moindres carrés ordinaire)

$$\min_{\mathbf{w},b} \quad \frac{1}{2}\sum_{i=1}^{n}(y_i - \mathbf{x}_i^\top \mathbf{w} - b)^2 + \frac{\lambda}{2}\|\mathbf{w}\|^2$$

- Promouvoir les $\mathbf{w}$ de norme minimale et rendre le problème strictement convexe
- $\lambda$ est un hyperparamètre qui permet de limiter le sur-apprentissage
- $\lambda = 0$ permet de revenir à la régression des moindres carrés (MCO)
- Cette régularisation est appelée régularisation de Tikhonov
- La méthode résultante s'appelle la regression ridge (ou regression de crête)

$$\min_{\boldsymbol{\alpha}} J(\boldsymbol{\alpha}) \quad avec \quad J(\boldsymbol{\alpha}) = \frac{1}{2}\|\mathbf{y} - \mathbf{X}\boldsymbol{\alpha}\|^2 + \frac{\lambda}{2}\boldsymbol{\alpha}^\top \mathbf{S}\boldsymbol{\alpha}$$

avec $\mathbf{S} \in \mathbb{R}^{(d+1)\times(d+1)}$, une matrice dont le terme général est :

$$S_{i,j} = \begin{cases} 1 & si\ i = j\ et\ i \leq d \\ 0 & sinon \end{cases}$$

$\mathbf{S}$ est une matrice diagonale unitaire dont le dernier terme diagonal est nul. On trouve donc :

$$\boldsymbol{\alpha}^\top \mathbf{S}\boldsymbol{\alpha} = \sum_{i,j=1}^{d+1} \alpha_i \alpha_j S_{i,j} = \sum_{i=1}^{d} \alpha_i^2 = \sum_{i=1}^{d} \mathbf{w}_i^2 = \|\mathbf{w}\|^2$$

- En reprenant la formulation des MCO :

$$\begin{aligned} J(\boldsymbol{\alpha}) &= \frac{1}{2}\mathbf{y}^\top\mathbf{y} - \boldsymbol{\alpha}^\top\mathbf{X}^\top\mathbf{y} + \frac{1}{2}\boldsymbol{\alpha}^\top\mathbf{X}^\top\mathbf{X}\boldsymbol{\alpha} + \frac{\lambda}{2}\boldsymbol{\alpha}^\top\mathbf{S}\boldsymbol{\alpha} \\ &= \frac{1}{2}\mathbf{y}^\top\mathbf{y} - \boldsymbol{\alpha}^\top\mathbf{X}^\top\mathbf{y} + \frac{1}{2}\boldsymbol{\alpha}^\top(\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{S})\boldsymbol{\alpha} \end{aligned}$$

- En posant $\mathbf{p} = \mathbf{X}^\top\mathbf{y}$ et $\mathbf{M} = \mathbf{X}^\top\mathbf{X} + \lambda\mathbf{S}$, on retrouve :

$$\begin{aligned} \frac{\partial J(\boldsymbol{\alpha})}{\partial \alpha_i} &= \frac{\partial}{\partial \alpha_i}\frac{1}{2}\mathbf{y}^\top\mathbf{y} - \frac{\partial}{\partial \alpha_i}\boldsymbol{\alpha}^\top\mathbf{X}^\top\mathbf{y} + \frac{\partial}{\partial \alpha_i}\frac{1}{2}\boldsymbol{\alpha}^\top(\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{S})\boldsymbol{\alpha} \\ &= 0 - p_i + \frac{1}{2}\sum_{j=1}^{d+1}(M_{ij} + M_{ji})\alpha_j \end{aligned}$$

- Forme matricielle

$$\nabla J(\boldsymbol{\alpha}) = -\mathbf{p} + \mathbf{M}\boldsymbol{\alpha} = -\mathbf{X}^\top\mathbf{y} + (\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{S})\boldsymbol{\alpha}$$

- La minimisation de $J(\boldsymbol{\alpha})$ est réalisée lorsque le gradient s'annule :

$$\nabla J(\hat{\boldsymbol{\alpha}}) = 0 \quad \Leftrightarrow \quad -\mathsf{X}^\top \mathsf{y} + (\mathsf{X}^\top \mathsf{X} + \lambda \mathsf{S})\hat{\boldsymbol{\alpha}} = 0$$

- La solution est le vecteur $\hat{\boldsymbol{\alpha}}$ défini par :

$$\hat{\boldsymbol{\alpha}} = (\mathsf{X}^\top \mathsf{X} + \lambda \mathsf{S})^{-1} \mathsf{X}^\top \mathsf{y}$$

- Régularisation :
  - $\mathsf{S}$ ajoute $\lambda$ à tous les éléments de la diagonale de $\mathsf{X}^\top \mathsf{X}$, ce qui la rend inversible.
  - Le problème est maintenant bien posé et une solution unique existe

- Basé sur la norme $\ell_1$ (vs norme $\ell_2$ pour ridge)

$$\|\mathbf{w}\|_1 = \sum_{i=1}^{d} |w_i|$$

- Regression LASSO (*least absolute shrinkage and selection operator*) :

$$\min_{\mathbf{w},b} \quad \frac{1}{2}\sum_{i=1}^{n}(y_i - \mathbf{x}_i^\top \mathbf{w} - b)^2 + \frac{\lambda}{2}\|\mathbf{w}\|_1$$

- Pas de calcul direct : algorithmes itératifs jusqu'à obtenir une solution

- Combinaison des deux régularisations :

$$\min_{\mathbf{w},b} \quad \frac{1}{2}\sum_{i=1}^{n}(y_i - \mathbf{x}_i^\top \mathbf{w} - b)^2 + \frac{\lambda_1}{2}\|\mathbf{w}\|^2 + \frac{\lambda_2}{2}\|\mathbf{w}\|_1$$

avec $(\lambda_1 = 0$ et $\lambda_2 > 0)$ : Ridge ; $(\lambda_1 > 0$ et $\lambda_2 = 0)$ : LASSO ; $(\lambda_1 = 0$ et $\lambda_2 = 0)$ : MCO

- ou

$$\min_{\mathbf{w},b} \quad \frac{1}{2}\sum_{i=1}^{n}(y_i - \mathbf{x}_i^\top \mathbf{w} - b)^2 + \frac{\lambda}{2}\left(\omega\|\mathbf{w}\|^2 + (1-\omega)\|\mathbf{w}\|_1\right)$$

avec $(\omega = 0)$ : Ridge ; $(\omega = 1)$ : LASSO

- Pas de calcul direct mais plusieurs algorithmes itératifs.

- Quelques soit $\lambda$, Ridge n'autorise pas $w_i = 0$ (tous les coefficients sont non nuls)
- En revanche, c'est souvent la cas avec LASSO = sélection de caractéristiques
- Si beaucoup de caractéristiques et/ou certaines sont non pertinentes, LASSO est préférable
- Si caractéristiques corrélées, Ridge est préférable car LASSO peut sélectionner "au hasard"
- Elastic net est un bon compromis entre les deux mais nécessite de régler deux hyperparamètres ($\lambda_1$ et $\lambda_2$)

# En pratique

# Exemple de régression ridge avec diabetes

1. Preparer les données :
   - charger le dataset (augmenter **X** avec une colonne de 1 si nécessaire)
   - Séparer le dataset en sous-ensemble d'apprentissage et de test

```
from sklearn import datasets

X, y = datasets.load_diabetes(scaled=False, return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3333, random_state=42)

# Note: on augmente pas X car c'est fait automatiquement avec l'implémentation de Scikit-learn
```

2. Choisir une métrique d'évaluation :

- Erreur quadratique moyenne (MSE pour *mean suqared error*)

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

  - 0 quand la prédiction est parfaite
  - Pas normalisée (dépend de la variance de $\mathbf{y}$)
  - Peu interprétable

2. Choisir une métrique d'évaluation :

- Erreur absolue moyenne (MAE pour *mean absolute error*)

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

  - 0 quand la prédiction est parfaite
  - Pas normalisée (dépend de la variance de $\mathbf{y}$)
  - Plus interprétable que MSE

2. Choisir une métrique d'évaluation :

- Coefficient de détermination ($R^2$)

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

  - 1 quand la prédiction est parfaite
  - Normalisée (entre 0 et 1) mais valeurs négatives possibles en test
  - La plus interprétable

3. Sélectionner la valeur de $\lambda$ :
   - choisir une méthode de validation : cross-validation
   - sélectionner un intervalle de valeur pour $\lambda$, e.g. $\in [10^{-4}, 10^{4}]$

```
from sklearn.linear_model import Ridge

alphas = np.logspace(-4, 4, 100)
ridge = Ridge()
grid = GridSearchCV(estimator=ridge, param_grid=dict(alpha=alphas), cv=5,
     scoring='neg_mean_squared_error')
grid.fit(X_train, y_train)
print(grid.best_params_)
print(grid.best_score_)
```

```
> {'alpha': np.float64(0.6280291441834259)}
  -3221.541906652895
```

4. Apprentissage et évaluation

```
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

ridge = grid.best_estimator_
ridge.fit(X_train, y_train)
predictions = ridge.predict(X_test)
print(mean_squared_error(y_test, predictions))
print(mean_absolute_error(y_test, predictions))
print(r2_score(y_test, predictions))
```

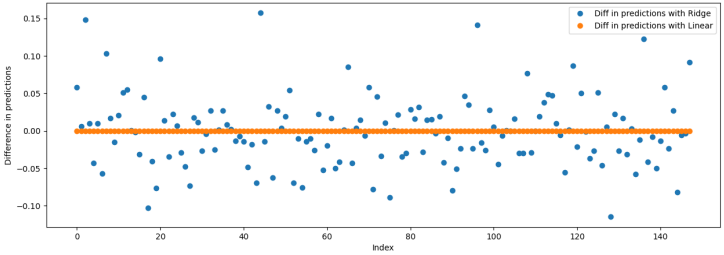
```
> 2785.964605380989
  41.515630696714524
  0.5102031756244414
```

- Normalisation (standardisation)

$$X = \frac{X - \mu}{\sigma}$$

avec $\mu$ et $\sigma$ respectivement la moyenne et l'écart type de chaque colonne de $X$

- Aucun effet sur la solution des MCO, mais mportant pour LASSO et Ridge



# Normalisation des données

```
from sklearn.preprocessing import StandardScaler

X, y = datasets.load_diabetes(return_X_y=True, scaled=False)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3333, random_state=42)
scaler = StandardScaler()
X_train_std = scaler.fit_transform(X_train)
X_test_std = scaler.transform(X_test)
```

# Annexes

On veut calculer :

$$\frac{\partial l(\boldsymbol{\alpha})}{\partial \alpha_i} = \frac{\partial}{\partial \alpha_i} \frac{1}{2} \mathbf{y}^\top \mathbf{y} - \frac{\partial}{\partial \alpha_i} \boldsymbol{\alpha}^\top \mathbf{X}^\top \mathbf{y} + \frac{\partial}{\partial \alpha_i} \frac{1}{2} \boldsymbol{\alpha}^\top \mathbf{X}^\top \mathbf{X} \boldsymbol{\alpha}$$

Pour calculer le troisième terme, on pose $\mathbf{M} = \mathbf{X}^\top \mathbf{X}$ et on utilise $(uv)' = uv' + u'v$ avec $u = \alpha_j$ et $v = \sum_{k=1}^{d+1} \alpha_k M_{jk}$ :

$$\frac{\partial}{\partial \alpha_i} \boldsymbol{\alpha}^\top \mathbf{M} \boldsymbol{\alpha} = \frac{\partial}{\partial \alpha_i} \sum_{j=1}^{d+1} \sum_{k=1}^{d+1} \alpha_j \alpha_k M_{jk} = \sum_{j=1}^{d+1} \frac{\partial}{\partial \alpha_i} \left( \alpha_j \sum_{k=1}^{d+1} \alpha_k M_{jk} \right) = \sum_{j=1}^{d+1} (uv)'$$

$$= \sum_{j=1}^{d+1} \left( u'v + uv' \right) = \sum_{j=1}^{d+1} u'v + \sum_{j=1}^{d+1} uv'$$

Comme $u' = 0$ pour $j \neq i$ et $u' = 1$ pour $j = i$, on a

$$\sum_{j=1}^{d+1} u'v = v \quad avec\ j = i \quad \Rightarrow \quad \sum_{j=1}^{d+1} u'v = \sum_{k=1}^{d+1} \alpha_k M_{ik}$$

Et comme $v' = 0$ pour $k \neq i$ et $v' = M_{ji}$ pour $k = i$

$$\sum_{j=1}^{d+1} uv' = \sum_{j=1}^{d+1} uM_{ji} \quad \Rightarrow \quad \sum_{j=1}^{d+1} uv' = \sum_{j=1}^{d+1} \alpha_j M_{ji}$$

Et donc

$$\frac{\partial}{\partial \alpha_i} \boldsymbol{\alpha}^\top \mathbf{M} \boldsymbol{\alpha} = \sum_{j=1}^{d+1} \alpha_j M_{ji} + \sum_{k=1}^{d+1} \alpha_k M_{ik}$$