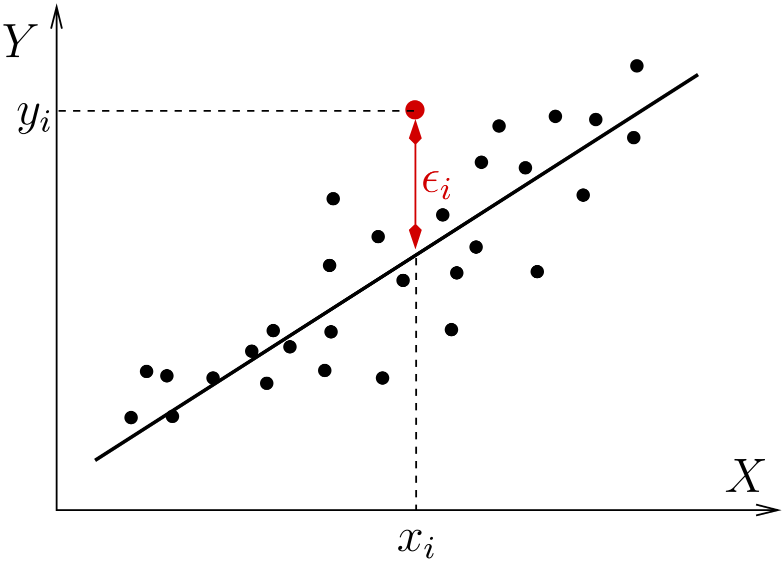
DESCENTE DE GRADIENT (ET RÉGRESSION)

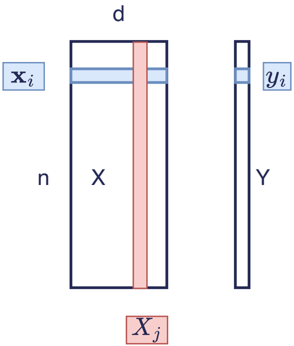
Vincent Guigue
vincent.guigue@agroparistech.fr

Régression

Régression linéaire



Régression linéaire au sens des moindres carrés



- $\mathbf{x}_i \in \mathbb{R}^d$, $y_i \in \mathbb{R}$
- Modèle linéaire : $\hat{y}_i = \sum_{j=1}^{d} x_{ij} w_j + b$
- Simplification : $\hat{y}_i = \sum_{j=1}^{d+1} x_{ij} w_j$ avec $\forall i,\ x_{i,d+1} = 1$
- Critère d'optimisation : $C = \sum_{i=1}^{n} (\hat{y}_i - y_i)^2$
- Problème d'apprentissage :

$$\mathbf{w}^\star = \arg\min_{\mathbf{w}} \sum_{i=1}^{n} (\sum_{j=1}^{d+1} x_{ij} w_j - y_i)^2$$

Ecriture matricielle & résolution analytique

$$C = \sum_{i=1}^{n} (\hat{y}_i - y_i)^2 = \sum_{i=1}^{n} \left(\sum_{j=1}^{d+1} x_{ij} w_j - y_i \right)^2 = \|Xw - Y\|^2$$

Résolution par annulation des dérivées partielles sur une formulation convexe :

$$\nabla_{\mathbf{w}} C = \begin{pmatrix} \frac{\partial C}{\partial w_1} \\ \vdots \\ \frac{\partial C}{\partial w_d} \end{pmatrix} = 2X^T(X\mathbf{w} - Y), \qquad \frac{\partial C}{\partial w_j} = \sum_{i=1}^{n} 2x_{ij} \left(\sum_{j=1}^{d+1} x_{ij} w_j - y_i \right)$$

$$\mathbf{w}^\star \quad \Leftrightarrow \quad \nabla_{\mathbf{w}} C = 0 \quad \Leftrightarrow \quad 2X^T(X\mathbf{w} - Y) = 0 \quad \Leftrightarrow \quad X^T X\mathbf{w} = X^T Y$$

Cette dernière forme correspond à un système d'équations linéaires :

$$\begin{pmatrix} X_1^T X_1 & X_1^T X_2 & \dots & X_1^T X_d \\ \vdots & \vdots & \vdots & \vdots \\ X_d^T X_1 & X_d^T X_2 & \dots & X_d^T X_d \end{pmatrix} \begin{pmatrix} w_1 \\ \vdots \\ w_d \end{pmatrix} = \begin{pmatrix} X_1^T Y \\ \vdots \\ X_d^T Y \end{pmatrix}$$

Résolution rapide & efficace

Problème de la forme :

$$A\mathbf{w} = B, \qquad A = X^T X \in \mathbb{R}^{d\times d}, \qquad B = X^T Y \in \mathbb{R}^d$$

Résolution `w=np.linalg.solve(A,B)`

Pourquoi aller plus loin ? ? ?

Résolution rapide & efficace

Problème de la forme :

$$A\mathbf{w} = B, \qquad A = X^T X \in \mathbb{R}^{d\times d}, \qquad B = X^T Y \in \mathbb{R}^d$$

Résolution `w=np.linalg.solve(A,B)`

Pourquoi aller plus loin ? ? ?

Pour passer à l'échelle

Résolution rapide & efficace

Problème de la forme :

$$A\mathbf{w} = B, \qquad A = X^T X \in \mathbb{R}^{d\times d}, \qquad B = X^T Y \in \mathbb{R}^d$$

Résolution `w=np.linalg.solve(A,B)`

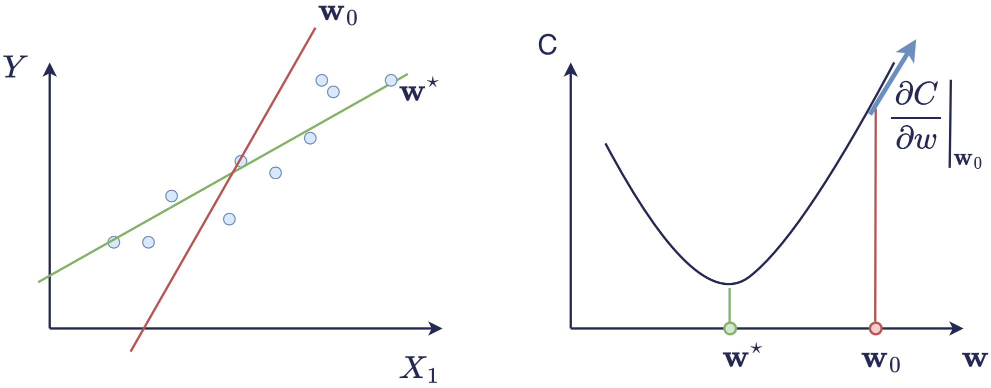
Pourquoi aller plus loin ? ? ?

Pour passer à l'échelle Pour profiter d'Un cadre idéal pour étudier la descente de gradient :

1. descente de gradient = solution approchée
2. descente de gradient = le 4x4 de l'optimisation :
   - on fait tout... Et parfois n'importe quoi

Gradient

Espace de description *vs* espace des paramètres

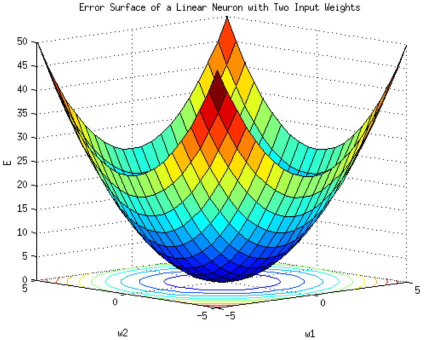


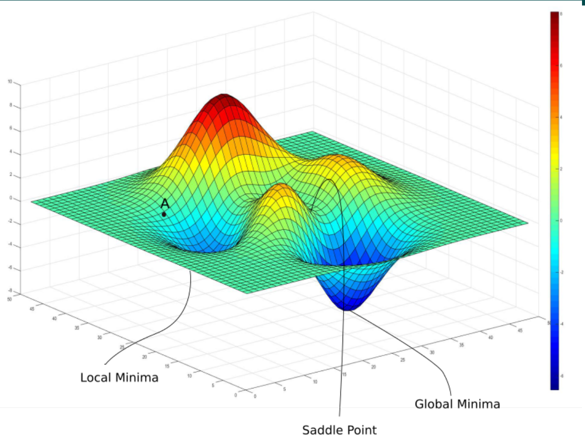
Algorithme itératif de la descente de grandient :

1. Initialiser $\mathbf{w}_0$
2. En boucle (avec mise à jour du gradient) :

$$\mathbf{w}^{t+1} = \mathbf{w}^{t} - \varepsilon \nabla_{\mathbf{w}} C, \qquad \varepsilon \text{ : learning rate}$$

Espace de description *vs* espace des paramètres





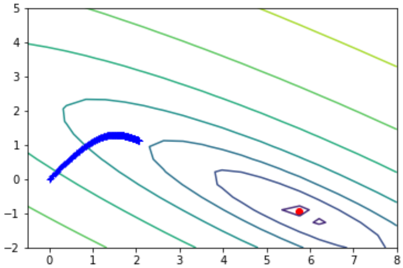
Algorithme itératif de la descente de grandient :

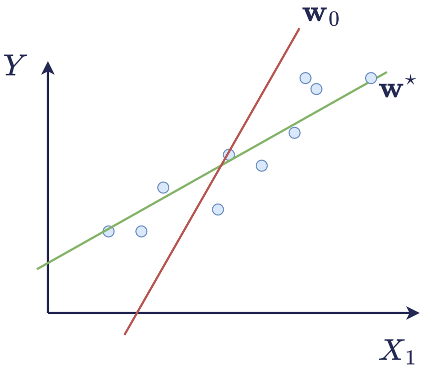
1. Initialiser $\mathbf{w}_0$
2. En boucle (avec mise à jour du gradient) :

$$\mathbf{w}^{t+1} = \mathbf{w}^{t} - \varepsilon \nabla_{\mathbf{w}} C, \qquad \varepsilon : \text{learning rate}$$

But du TP associé

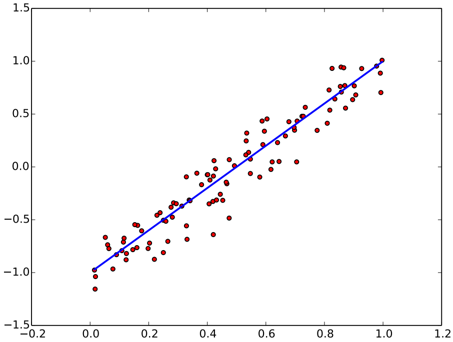
1. Rappeler les bases de la régression
2. Générer des données jouet
3. Implémenter une stratégie d'apprentissage à base de gradient :
   - Initialiser aléatoirement un régresseur
   - Améliorer itérativement le classifieur
4. Comprendre et illustrer le processus
   - Dans l'espace des points & dans l'espace des paramètres
   - Analyser empiriquement le processus itératif par rapport à la solution analytique

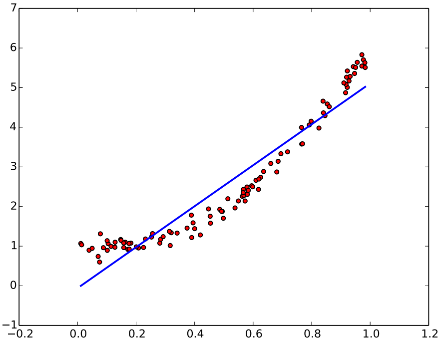




Extensions

Passer à la regression non linéaire





Bonne nouvelle : le cadre précédent va nous permettre de passer facilement au cas non linéaire.

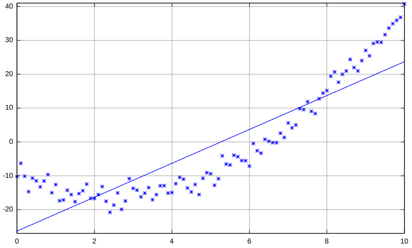
Transition facile

1. Transformer les données

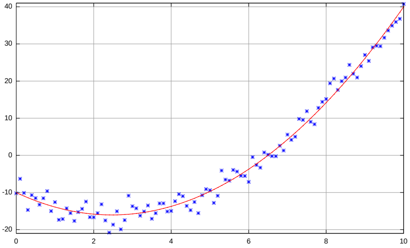
$$Xinit = \begin{pmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{pmatrix} \Rightarrow X = \begin{pmatrix} x_0^2 & x_0 & 1 \\ \vdots & \vdots & \\ x_n^2 & x_n & 1 \end{pmatrix}$$

2. Tout est fait !

$$\hat{Y} = X \cdot w, \qquad X \in \mathbb{R}^{n\times 3}, \qquad w = \begin{pmatrix} a \\ b \\ c \end{pmatrix} \in \mathbb{R}^2, \qquad \forall i,\ \hat{y}_i = ax_i^2 + bx_i + c$$



$\Rightarrow$



Gradient stochastique

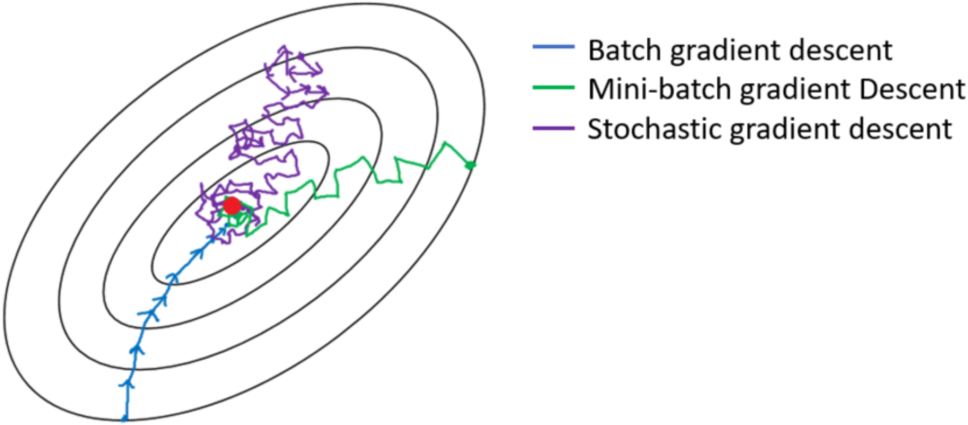
Le calcul de $\nabla_{\mathbf{w}} C$ est coûteux... Il est possible de décomposer le problème :

$$C = \sum_{i=1}^{N} C_i, \qquad C_i = (\mathbf{x}_i \mathbf{w} - y_i)^2$$

Algorithme stochastique (Cas MC : ADALINE) :

1. Initialiser $\mathbf{w}_0$
2. En boucle (avec mise à jour du gradient) :
   - Tirage aléatoire d'un échantillon i
   - Calcul de $\nabla_{\mathbf{w}} C_i$ (cas MC : $\nabla_{\mathbf{w}} C_i = 2\mathbf{x}_i^T(\mathbf{x}_i \mathbf{w} - y_i)$)
   - MAJ : $\mathbf{w}^{t+1} = \mathbf{w}^t - \varepsilon \nabla_{\mathbf{w}} C_i$

Stochastic, batch... Ou mini-batch

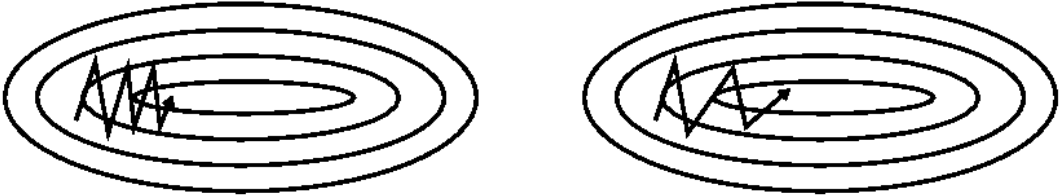


Le nombre d'itérations n'est pas le coût !

Plein de variantes très efficaces pour le gradient

Sebastian Ruder :

https://ruder.io/optimizing-gradient-descent/



Francis Bach : https://francisbach.com/

Perceptron

Perceptron

Algorithme de classification binaire des années 60 : toujours très efficace aujourd'hui

$$C = \sum_{i=1}^{N} (-y_i \mathbf{x}_i \mathbf{w})_+$$

Algorithme stochastique (Cas charnière : Perceptron) :

1. Initialiser $\mathbf{w}_0$
2. En boucle (avec mise à jour du gradient) :
   - Tirage aléatoire d'un échantillon i
   - Si $y_i \mathbf{x}_i \mathbf{w} \leq 0$
     - Calcul de $\nabla_{\mathbf{w}} C_i = -y_i x_i^T$
     - MAJ : $\mathbf{w}^{t+1} = \mathbf{w}^t - \varepsilon \nabla_{\mathbf{w}} C_i$