

MLWEEK - Formation Machine Learning

Cours 2 : Régressions Linéaire et Logistique

Jeff Abrahamson

François-Marie Giraud

23 Janvier 2019



<https://www.ml-week.com/>

Machine Learning

Cours 2 : Régressions Linéaire et Logistique

Machine Learning

Rappels

Apprentissage supervisé

Prédire une valeur numérique ou l'appartenance à une classe
Données d'entraînement **annotées** !

Ex : prédiction CAC40, classification d'image/texte/...

À l'intérieur du **Modèle** :

- **Algèbre linéaire**
- Théorie de l'Optimisation
- Calcul différentiel
- Probabilités
- Statistiques

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} ax + by \\ cx + dy \end{bmatrix}$$

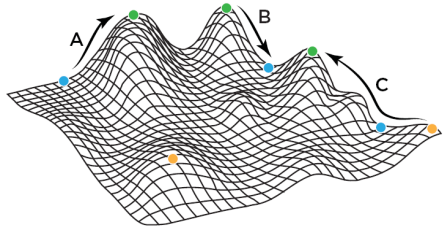
$$\begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} ax + by + c \\ dx + ey + f \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} ax + by + cz \\ dx + ey + fz \\ gx + hy + iz \end{bmatrix}$$

$$\begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} ax + by + cz + d \\ ex + fy + gz + h \\ ix + jy + kz + l \\ 1 \end{bmatrix}$$

À l'intérieur du **Modèle** :

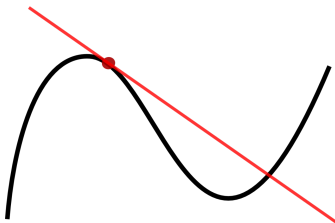
- Algèbre linéaire
- **Théorie de l'Optimisation**
- Calcul différentiel
- Probabilités
- Statistiques



By Max Olson for FutureBlind

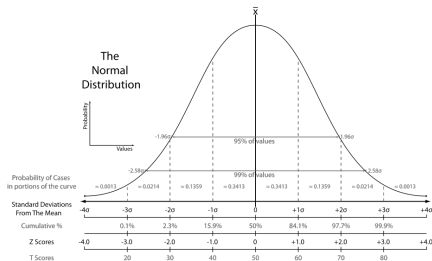
À l'intérieur du **Modèle** :

- Algèbre linéaire
- Théorie de l'Optimisation
- **Calcul différentiel**
- Probabilités
- Statistiques



À l'intérieur du **Modèle** :

- Algèbre linéaire
- Théorie de l'Optimisation
- Calcul différentiel
- **Probabilités**
- **Statistiques**



Machine Learning

Regression Linéaire

Définition du problème :

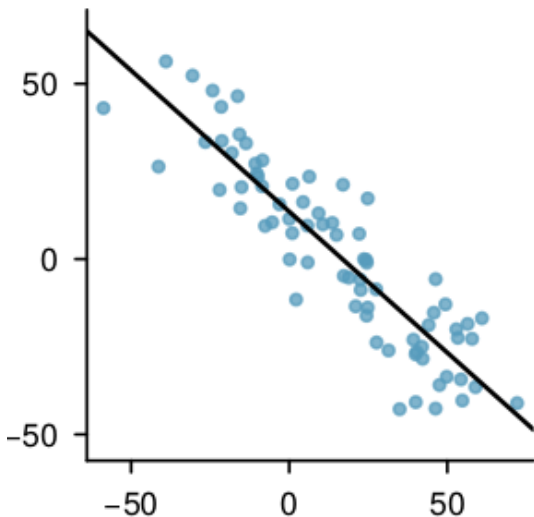
Soit $\{(x_i, y_i)\}_{i \in \mathbb{R}}$ un ensemble de données tel que $\forall i, x_i \in \mathbb{R}$ et $y_i \in \mathbb{R}$

Trouver $\phi^*(x_i) = y_i^*$ telle que

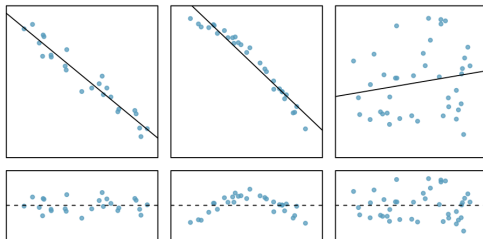
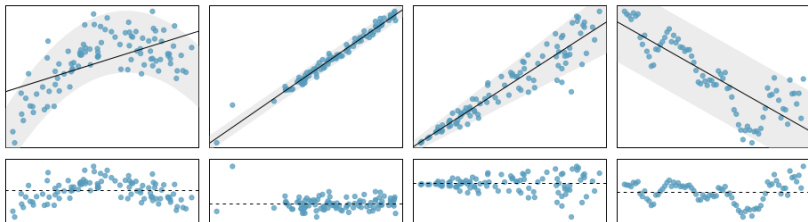
$$\forall i, y_i^* - y_i \rightarrow 0$$

sous la contrainte que ϕ^* soit une fonction linéaire (affine)

Regression Linéaire



Regression Linéaire



Machine Learning

Fonction de Coût/Erreur

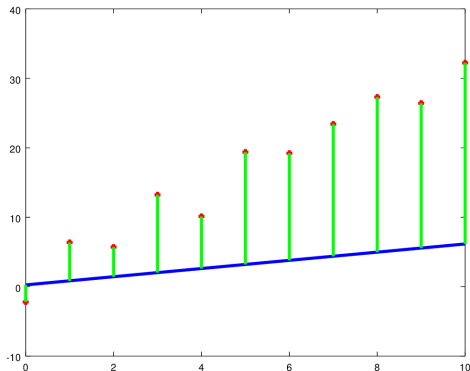
Fonction de Coût/Erreur

Erreur moyenne :

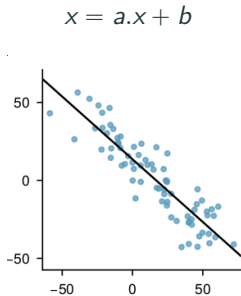
$$\frac{1}{n} \sum_{i=[1..n]} \sqrt{(y_i^* - y_i)^2}$$

Critère des moindres carrés :

$$\frac{1}{n} \sum_{i=[1..n]} (y_i^* - y_i)^2$$



Fonction de Coût/Erreur



$$E_{\Omega} = \frac{1}{2n} \sum_{i=[1..n]} (y_i^* - y_i)^2$$

Machine Learning

Optimisation

Des solutions analytiques existent !

mais ...

Optimisation

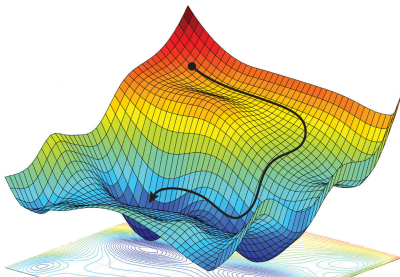
Calcul du gradient de l'erreur par rapport aux paramètres :

$$\frac{\partial Err}{\partial w_i}$$

Mise à jour :

$$w'_i = w_i - \gamma * grad$$

où : $0 < \gamma < 1$ (learning rate)



- 1 - initialisation aléatoire du modèle
- 2 - Tant que(critère arrêt == 0)
 - Selection aléatoire d'un **batch** de données
 - **Forward** : Passe avant du **batch** dans le modèle
 - Calcul de l'erreur par rapport aux sorties attendues
 - **Backward** : Rétropropagation du gradient de l'erreur en fonction des paramètres dans le modèle (mise à jour du modèle)
 - Calcul critère arrêt

Machine Learning

Gradient de l'erreur

$$y = a.x + b$$

$$E_{\Omega} = \frac{1}{2n} \sum_{i=[1..n]} (y_i^* - y_i)^2$$

$$E_{\Omega} = \frac{1}{2n} \sum_{i=[1..n]} (y_i^* - (a.x_i + b))^2$$

...

$$\frac{\partial E_{\Omega}}{\partial a} = \frac{1}{n} \sum_{i=[1..n]} (a.x_i + b - y_i^*).x_i$$

$$\frac{\partial E_{\Omega}}{\partial b} = \frac{1}{n} \sum_{i=[1..n]} (a.x_i + b - y_i^*)$$

$$U^2' = 2U' * U$$

M.A.J :

$$a \leftarrow a - \gamma \cdot \frac{\partial E_{\Omega}}{\partial a}$$

$$b \leftarrow b - \gamma \cdot \frac{\partial E_{\Omega}}{\partial b}$$

où $1 > \gamma > 0$ (learning rate)

Machine Learning

TP2 : Régression Linéaire

Deux Options :

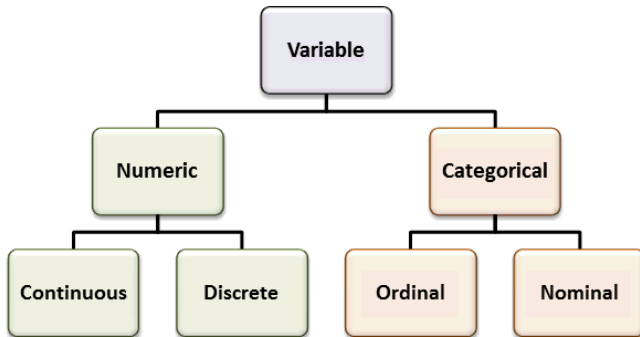
`$ jupyter notebook regression-lineaire.ipynb`

OU

Suivre à l'écran et programmer dans le shell ipython

Machine Learning

Régression Logistique



0	1	2	3	4	5	6	7	8	9
0	1	2	3	4	5	6	7	8	9
0	1	2	3	4	5	6	7	8	9
0	1	2	3	4	5	6	7	8	9
0	1	2	3	4	5	6	7	8	9
0	1	1	4	1	3	1	8	0	8
0	1	0	5	5	5	7	8	4	3
0	1	0	4	3	3	1	9	1	8
0	0	6	8	5	4	1	8	1	2
0	1	2	9	5	0	2	8	8	5

Régression Logistique



[001.ak47](#)



[002.american-flag](#)



[003.backpack](#)



[004.baseball-bat](#)



[005.baseball-glove](#)



[006.basketball-hoop](#)



[007.bat](#)



[008.bathtub](#)



[009.bear](#)



[010.beer-mug](#)



[011.billiards](#)



[012.binoculars](#)

\approx Distance entre la sortie et la cible ?

Sortie :

0.0	0.1	0.4	0.0	0.0	0.2	0.1	0.0	0.2	0.0
-----	-----	-----	-----	-----	-----	-----	-----	-----	-----

Cible :

0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
-----	-----	-----	-----	-----	-----	-----	-----	-----	-----

Machine Learning

Réseau de Neurones

$$\sigma \left(\begin{bmatrix} x_1 & x_2 & \dots & x_d \end{bmatrix} * \begin{bmatrix} ?_{11} & ?_{12} & \dots & ?_{1n} \\ ?_{21} & ?_{22} & \dots & ?_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ ?_{d1} & ?_{d2} & \dots & ?_{dn} \end{bmatrix} \right) = \begin{bmatrix} o_1 & o_2 & \dots & o_n \end{bmatrix}$$

où :

X est une donnée en entrée de dimension \mathbf{d} ,

$?_{ij}$ sont les paramètres à trouver des \mathbf{n} neurones de notre modèle.

σ la fonction d'activation et

O la sortie du réseau

Calcul de l'erreur

$$\frac{1}{2} \left(\begin{bmatrix} o_1^* & o_2^* & \dots & o_n^* \end{bmatrix} - \begin{bmatrix} o_1 & o_2 & \dots & o_n \end{bmatrix} \right)^2 = \begin{bmatrix} e_1 & e_2 & \dots & e_n \end{bmatrix}$$

où :

O^* représente la sortie attendue du réseau,

O la sortie du réseau et

E l'erreur commise par chaque neurone de sortie.

Mise à jour des poids (**backward**)

$$\Delta w_i = -\gamma * \frac{\partial E}{\partial w_i}$$

où :

Δw_i est l'update du paramètre w_i et

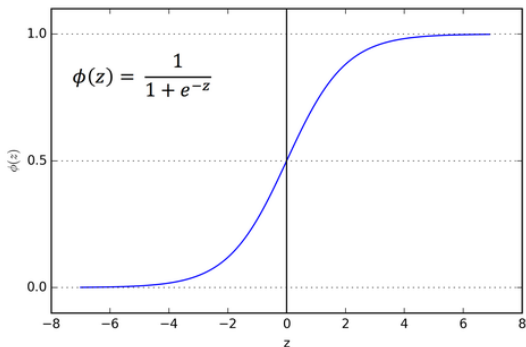
γ est un méta-paramètre du modèle (learning rate)

Réseaux de neurones : Fonction d'activation σ

- Sigmoidé
- Tanh
- Softmax
- ReLU
- ...

Réseaux de neurones : Fonction d'activation σ

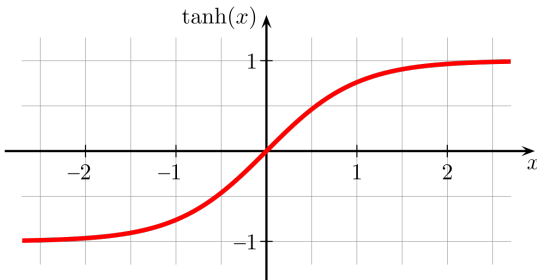
Sigmoïde



$$\frac{\partial \phi(x)}{\partial x} = \phi(x) * (1 - \phi(x))$$

Réseaux de neurones : Fonction d'activation σ

$$\tanh(x) = \frac{1 - \exp -2 * x}{1 + \exp -2 * x}$$



$$\frac{\partial \tanh(x)}{\partial x} = 1 - \tanh^2(x)$$

$$\text{Softmax}(x_j) = \frac{\exp x_j}{\sum_{i=1}^n \exp x_i}$$

donc :

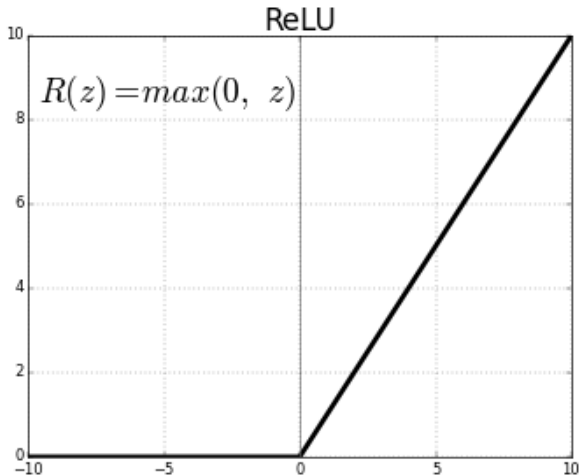
$$\sum_{j=1}^n \text{Softmax}(x_j) = 1$$

dérivée (ou jacobien car le softmax est une fonction de $\mathbb{R}^n \rightarrow \mathbb{R}^n$) :

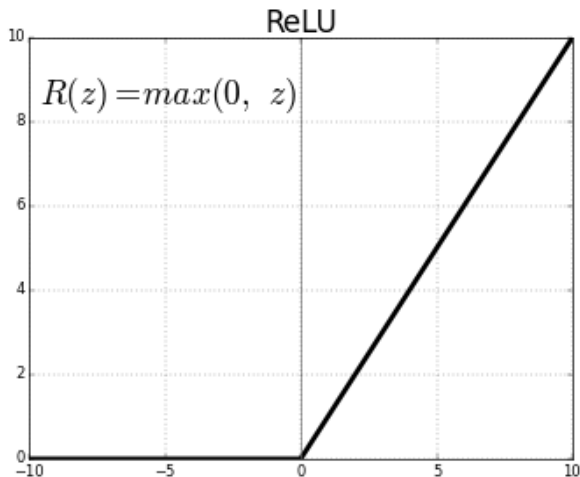
$$D_j S_i = S_i(\delta_{ij} - S_j)$$

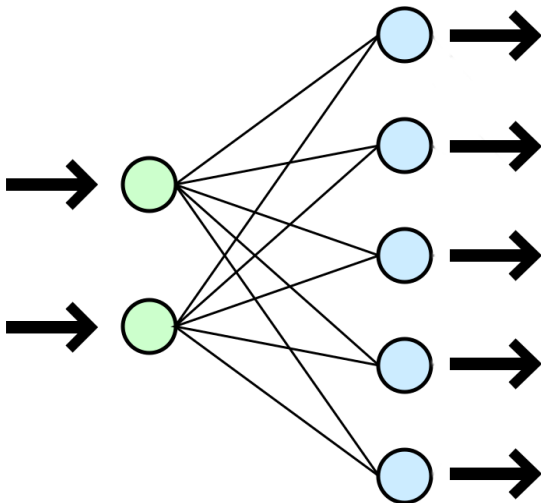
où $D_j S_i$ est la dérivée partielle de la i-ième sortie par rapport à la j-ième entrée

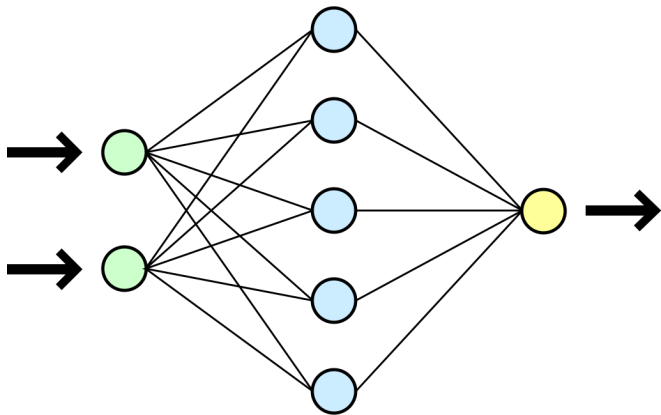
δ_{ij} est le delta de Kronecker



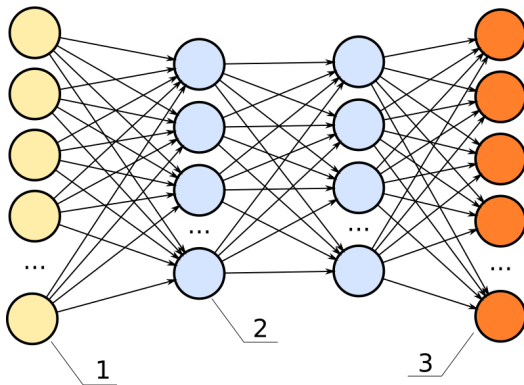
Réseaux de neurones : Fonction d'activation σ



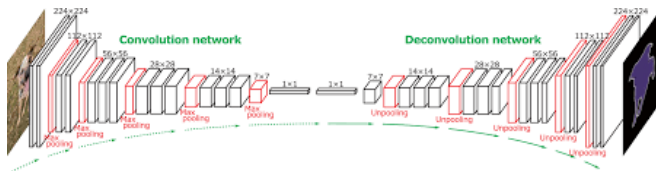




Réseau de Neurones



Réseau de Neurones



Machine Learning

TP 2 : Régression Logistique

Deux Options :

```
$ jupyter notebook logistic_regression.ipynb
```

OU

Suivre à l'écran et programmer dans le shell ipython

requirements : python3 python3-dev python3-tk installées.