

Université J.F. CHAMPOLLION Albi

Régressions



Licence d'informatique S6

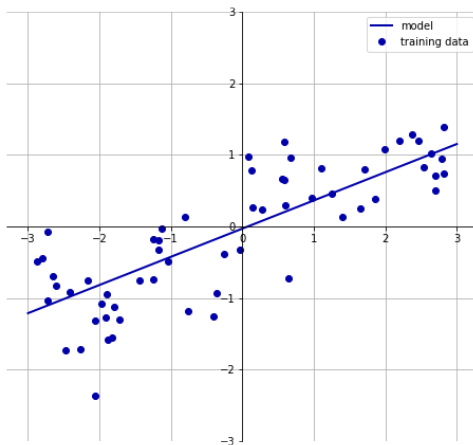
Thierry Montaut

Apprentissage supervisé d'un modèle paramétré

- On rappelle qu'on parle de problème de régression quand l'ensemble Y des valeurs prises par les étiquettes n'est plus fini mais est un intervalle de \mathbb{R} .
- Un modèle paramétré suppose que la forme analytique de la fonction de décision est connue et dépend de certains paramètres. Le but est d'apprendre à partir des données les valeurs des paramètres qui minimisent une fonction d'erreur fixée. En général, la complexité augmente avec le nombre de paramètres.
- Nous nous concentrerons cette année sur les **régressions linéaires**, c'est-à-dire les problèmes pour lesquels la fonction de décision est une fonction linéaire des caractéristiques.
- Les modèles linéaires sont les plus anciennes des méthodes de prédictions, déjà étudiées par Gauss et Legendre.

Régression linéaire

- On parle de régression linéaire lorsque la fonction de décision est une combinaison linéaire des caractéristiques X .
- Dans le cas d'une unique variable x , on cherche donc à déterminer les coefficients d'une droite d'équation $y = ax + b$ la plus proche possible des données (x_a, y_a) d'apprentissage, et qui permettra la prédiction $f(x) = ax + b$.
- Plus généralement dans \mathbb{R}^n , on cherche l'équation d'un hyperplan H d'équation $a_1x_1 + a_2x_2 + \dots + a_nx_n + b = 0$, le plus proche possible des données d'apprentissage (x_1, \dots, x_n, y) et on effectuera la prédiction $f(x) = a_1x_1 + a_2x_2 + \dots + a_nx_n + b$.

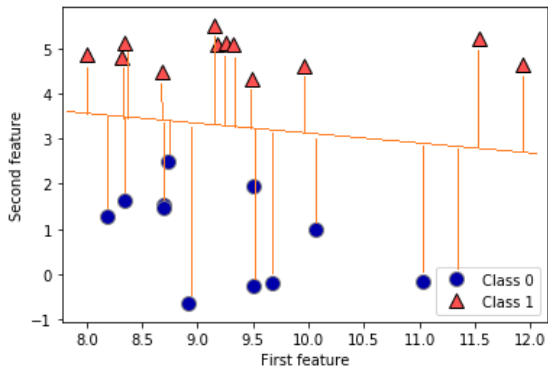


Critères de performance

- Comme nous l'avons vu précédemment, nous choisirons comme mesure de l'erreur la moyenne des erreurs quadratiques :

$$err(a_1, a_2, \dots, a_n, b) = \frac{1}{m} \sum_{i=1}^m (f(x_i) - y_i)^2$$

- Il s'agit de déterminer (a_1, \dots, a_n, b) minimisant la distance des points d'apprentissage à l'hyperplan H . On est donc ramené à la minimisation d'une fonction de plusieurs variables, ce qu'on pourra faire exactement, par approximation numérique ou en utilisant les fonctions de la bibliothèque sklearn.



Résolution exacte dans le cas d'une variable

Dans le cas d'une unique variable x $f(x) = ax + b$ (donc on cherche une droite de meilleure approximation $y = ax + b$), on sait résoudre exactement ce problème d'optimisation :

Proposition

$$\frac{\partial \text{err}}{\partial a} = \frac{2}{m} \sum_{i=1}^m x_i (ax_i + b - y_i)$$

$$\frac{\partial \text{err}}{\partial b} = \frac{2}{m} \sum_{i=1}^m (ax_i + b - y_i)$$

Résolution exacte dans le cas d'une variable

Le minimum est donc atteint sur l'unique point singulier, vérifiant $\nabla \text{err}(a, b) = 0$.

Théorème

L'erreur est minimale est obtenue pour la droite d'équation $y = a.x + b$ avec :

$$a = \frac{\text{cov}(x, y)}{\text{var}(x)}$$

et

$$b = \bar{y} - a\bar{x}$$

Résolution exacte dans le cas d'une variable

- Ce qui prouve au passage que le point moyen (\bar{x}, \bar{y}) appartient à la droite de meilleure approximation.
- Nous démontrerons ce résultat en exercice et nous utiliserons les fonctions statistiques de numpy pour déterminer des droites de régression.

Cas général

Dans le cas de n variables $f(x) = a_1 x_1 + \dots + a_n x_n + b$, on obtient :

Proposition

$$\frac{\partial \text{err}}{\partial b} = \frac{2}{m} \sum_{i=1}^m (a_1 x_{1,i} + \dots + a_n x_{n,i} + b - y_i)$$

$\forall k \in \llbracket 1, n \rrbracket :$

$$\frac{\partial \text{err}}{\partial a_k} = \frac{2}{m} \sum_{i=1}^m x_{k,i} (a_1 x_{1,i} + \dots + a_n x_{n,i} + b - y_i)$$

Forme matricielle

Soit en posant :

$$X = \begin{pmatrix} x_{1,1} & \dots & x_{n,1} & 1 \\ \vdots & & \vdots & \vdots \\ x_{1,m} & \dots & x_{n,m} & 1 \end{pmatrix} \quad A = \begin{pmatrix} a_1 \\ \vdots \\ a_n \\ b \end{pmatrix} \quad Y = \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix}$$


on obtient

Théorème

$$\nabla \text{err}(a_1, \dots, a_n, b) = \frac{2}{m} X^t (XA - Y).$$

L'erreur est minimale quand $\nabla \text{err} = 0$ donc pour

$$A = (X^t X)^{-1} \cdot (X^t Y)$$

Ce qui nous ramène à la résolution d'un système linéaire à $n+1$ inconnues. Quand n augmente, on préfère une résolution numérique 

Minimisation numérique : Descente de gradient

- La méthode de descente de gradient est une méthode numérique permettant de déterminer une valeur approchée des extrema d'une fonction d'une ou plusieurs variables réelles.
- On l'utilise dans les cas où on ne sait pas résoudre exactement le problème de minimisation et où des valeurs approchées de ce minimum suffise.
- Cette méthode est très répandue en Machine Learning notamment pour l'optimisation des problèmes de régression et la phase d'apprentissage des réseaux de neurones.

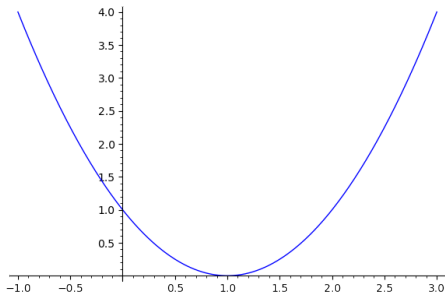
Minimisation numérique par descente de gradient

- L'idée centrale de la méthode de descente de gradient est que la dérivée (et plus généralement le vecteur gradient) donne la direction et le sens de plus grande augmentation de la fonction f . Symétriquement, l'opposé du vecteur gradient donne la direction et le sens de plus grande diminution.

Cas d'une fonction réelle

Exemple : On cherche à déterminer le minimum de la fonction réelle définie par $f(x) = x^2 - 2x + 1$.

Cette fonction est de classe \mathcal{C}^1 sur \mathbb{R} , on sait parfaitement étudier sa dérivée, ses variations et donc établir qu'elle possède un unique minimum en 1 valant 0.



Descente de gradient

Lorsque la résolution analytique est trop complexe, on cherche à utiliser une méthode numérique et itérative d'approximation de ce minimum : Si f est de classe C^1 :

Initialiser x_0 à une valeur quelconque

Tant qu'il n'y a pas convergence :

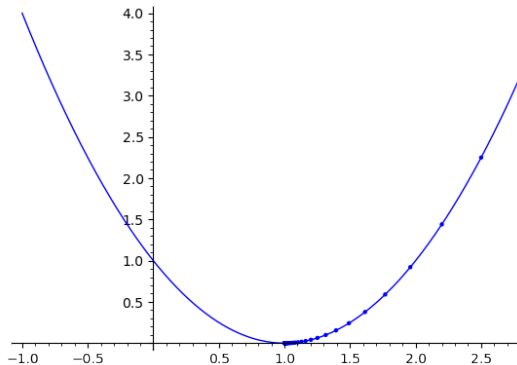
$$x_{k+1} = x_k - \alpha \cdot f'(x_k)$$

Descente de gradient

- $f'(x_k)$ donne le sens de plus grande variation. On utilise - pour une minimisation et + pour une recherche de maximum
- α est le pas de la méthode. C'est un paramètre important qui permet d'assurer la convergence et la vitesse de convergence de la méthode.
- La condition de sortie de boucle peut dépendre du nombre d'itérations, de la différence $x_{k+1} - x_k$ ou de la valeur de la dérivée $f'(x_k)$.
- Si la dérivée ne peut être calculée explicitement, elle est approchée localement par le taux d'accroissement.

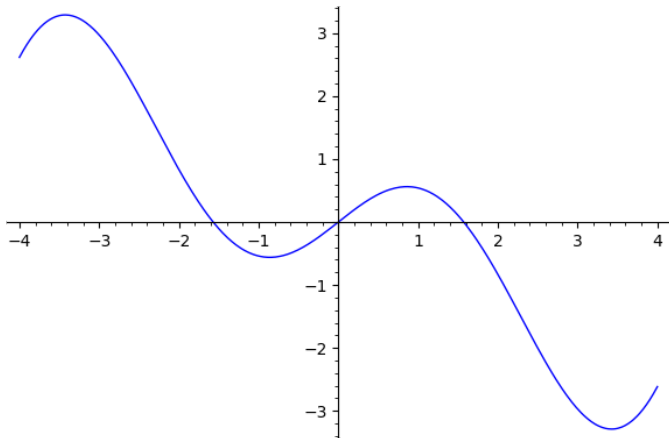
Entrée [7]: `X=grad1(f,2.5,0.1,0.001,100)`

```
1 : 2.2000000000000000
2 : 1.9600000000000000
3 : 1.7680000000000000
4 : 1.6144000000000000
5 : 1.4915200000000000
6 : 1.3932160000000000
7 : 1.3145728000000000
8 : 1.2516582400000000
9 : 1.2013265920000000
10 : 1.1610612736000000
11 : 1.1288490188800000
12 : 1.1030792151040000
13 : 1.0824633720832000
14 : 1.0659706976665600
15 : 1.0527765581332500
```



Les défauts de la méthode

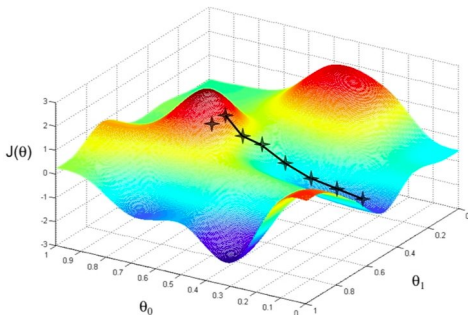
- Peut se faire piéger dans un minimum local si la fonction n'est pas convexe.
- La convergence peut être très lente dans les zones où la dérivée est très faible.



Cas d'une fonction de plusieurs variables

L'intérêt de cette méthode est qu'elle se généralise très simplement à une fonction de plusieurs variables en remplaçant la dérivée par le vecteur gradient

Gradient Descent



Descente de gradient

- Si f est de classe \mathcal{C}^1 sur \mathbb{R}^2 :

Initialiser X_0 à une valeur quelconque

Tant qu'il n'y a pas convergence :

$$X_{k+1} = X_k - \alpha \cdot \nabla f(X_k)$$

- Là encore, si les dérivées partielles ne peuvent être calculées explicitement, elle sont approchées localement par le taux d'accroissement des applications partielles.

A l'aide de scikit-learn

La bibliothèque "LinearRegression" de scikit-learn, contient toutes les fonctions nécessaires pour mener à bien une régression linéaire :

- *model = LinearRegression()* définit le modèle, puis comme d'habitude :
- *model.fit(X, y)* apprend les paramètres du modèle à l'aide des données
- *model.coef* et *model.intercept* donnent les valeurs apprises de (a_1, \dots, a_n) et de b .
- *model.score* et *model.predict* permettent alors de calculer le score de l'apprentissage et d'effectuer des prédictions sur des données inconnues.

Regardez la documentation en ligne de la bibliothèque "LinearRegression" pour plus d'informations. Vous pouvez également utiliser *SGDRegressor* de la bibliothèque *linear_model*.