Les régressions linéaires et polynomiales

Définition

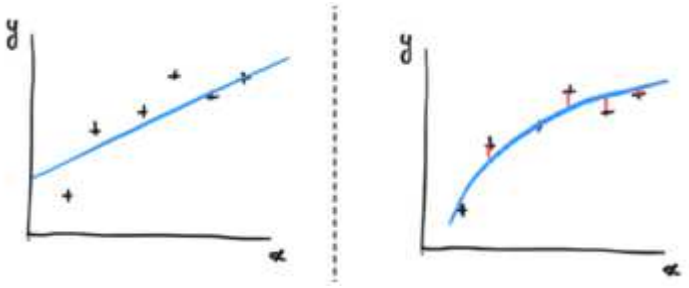
Les régressions linéaires et/ou polynomiales sont des algorithmes de type :

Apprentissage supervisé de prédiction d'une valeur continue.

Il s'agit de prédire la valeur que prendra une grandeur y en fonction d'un ou plusieurs paramètres d'entrée.

Pour cela, l'apprentissage devra définir un modèle linéaire ou polynomiale qui en fonction des paramètres d'entrée renverra une valeur de la valeur de sortie.

Voici ci-contre deux exemples de régressions univariables à un seul facteur.



linéaire
$f(x) = a.x + b$

polynomiale
$f(x) = a.x^2 + b.x + c$

Pour cela il faut disposer d'un jeu de données labélisées. C'est-à-dire un jeu de données dont on connait les valeurs des paramètres d'entrée ainsi que la valeur du paramètre de sortie.

Principe de l'apprentissage

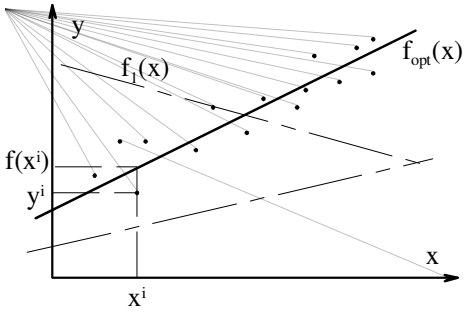
Prenons l'exemple d'une régression linéaire simple avec un seul paramètre d'entrée. On dispose pour cela d'une base de données d'apprentissage constituée d'une liste de **m couples (x,y)** où **x est la valeur du paramètre d'entrée et y la valeur de sortie**.

Objectif

Il s'agit de définir une loi de prédiction qui a une nouvelle valeur d'entrée x associera une valeur de sortie f(x) telle que f(x) = b + a.x. Il faut donc déterminer les paramètres de cette loi : a et b. Pour cela nous allons chercher les valeurs des paramètres a et b de tel sorte que pour le jeu de données d'apprentissage, la fonction f(x) est au plus proche de y.

L'apprentissage devra donc déterminer ces paramètres afin de réduire au mieux les écarts entre les m prédictions de cette loi f(x) et les m sorties y de la base de donnée d'apprentissage. C'est-à-dire réduire au mieux f(x) – y.

Prendre la somme des différences entre f(x) et y n'est pas satisfaisant. En effet, cette somme peut être très faible voir nulle alors que la prédiction est le plus souvent très éloignée de la valeur de sortie : Voir fonction $f_1(x)$ ci-contre.



Il faut, pour éviter cela, définir un écart positif. On choisit donc de prendre l'écart quadratique : $(f(x) - y)^2$. Le modèle retenu sera donc un modèle avec des paramètres a et b qui réduisent la somme des écarts quadratiques.

On définit alors le **cout de la loi de prédiction**, qui est en fait $\frac{1}{2}$ de la moyenne des carrés des écarts entre f(x) et y :

$$J(a,b) = \frac{1}{2.m} \sum_{i=1}^{m} (b + a.x^i - y^i)^2$$

Remarque : le facteur $\frac{1}{2}$ de cette fonction coût est totalement arbitraire. Il permet simplement de simplifier le calcul du gradient de descente. Ce facteur est sans importance sur notre objectif car on cherche uniquement à minimiser cette fonction coût et non pas à déterminer sa valeur.

Algorithme de la descente de gradient

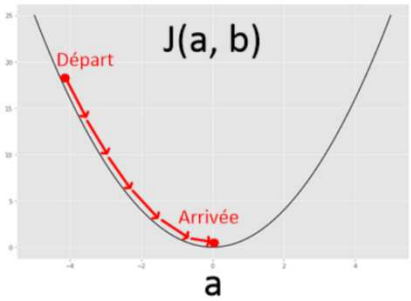
La fonction cout : J(a,b) est une fonction des deux paramètres a et b.

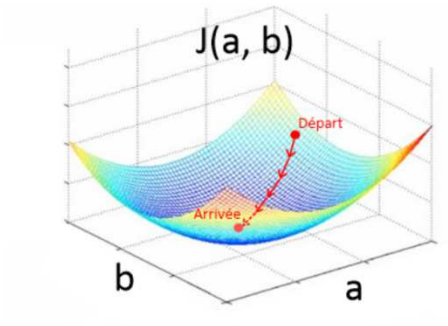
Nous allons donc dans un premier temps prendre des facteurs a et b totalement arbitrairement. Soit on prend a = b = 0 soit on choisit a et b aléatoirement. Puis nous allons faire varier les paramètres a et b sur plusieurs itérations de manière à se rapprocher du minimum de cette fonction coût.

La variation des paramètres a et b doit respecter deux critères pour atteindre au mieux le minimum de la fonction coût :

- ☞ Chaque variation doit faire diminuer la fonction cout
- ☞ Plus nous nous rapprochons du minimum plus la variation doit être fine.

Pour calculer les paramètres a_i et b_i à la $i^{ième}$ itération nous utiliserons les valeurs des paramètres à la $(i-1)^{ième}$ itération (a_{i-1} et b_{i-1}) et le gradient de descente. Ce gradient de descente sera proportionnel au taux de la variation de la fonction coût par rapport au paramètre considéré. Soit proportionnel aux dérivées partielles de la fonction coût par rapport aux paramètres a et b : $\frac{\partial J}{\partial b}$ et $\frac{\partial J}{\partial a}$

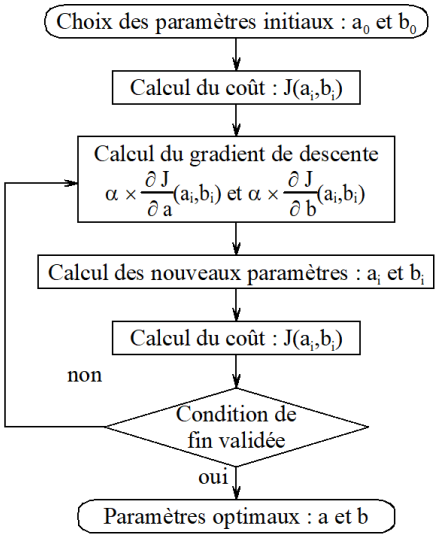




Pour faire converger les paramètres a et b vers le point où la fonction coût est minimal on utilisera les fonctions de récurrence : $a_{i+1} = a_i - \alpha \times \frac{\partial J}{\partial a}(a_i,b_i)$ et : $b_{i+1} = b_i - \alpha \times \frac{\partial J}{\partial b}(a_i,b_i)$

α : constante positive appelée vitesse d'apprentissage (learning rate)

Algorigramme de la « descente de gradient » :



La valeur de α doit être choisie de manière à faire converger les paramètres rapidement vers le minimum de la fonction coût sans que l'algorithme ne diverge.

- ☞ Si α est trop grand l'algorithme risque de diverger.
- ☞ Si α est trop petit la convergence sera trop lente et l'algorithme mettra un temps trop long pour atteindre la solution optimale.

Normalisation des données

Le choix de la vitesse d'apprentissage α est une étape difficile pour cela on modifie souvent les données d'entrée pour les « normaliser ». Ensuite la convergence pourra être assurée avec une vitesse d'apprentissage α telle que : $\alpha \in [0,1]$

Le principe de la normalisation des données est le suivant :

On calcule la moyenne et l'écart type des m données d'entrée (x^i) du jeu d'apprentissage :

$$\bar{x} = \frac{1}{m}\sum_{i=1}^{m} x^i \qquad \text{et :} \qquad \sigma_x = \sqrt{\frac{1}{m}\sum_{i=1}^{m}(x^i - \bar{x})^2}$$

On crée un nouveau jeu de m données d'entrée $\overset{*}{x}^i$ (données centrées réduites) :

On a alors un jeu de données dont la moyenne est nulle et l'écart type égal à 1.

L'algorithme de la descente de gradient fournit alors les paramètre $\overset{*}{b}$ et $\overset{*}{a}$ de la fonction de prédiction $\overset{*}{f}(\overset{*}{x}) = \overset{*}{b} + \overset{*}{a}.\overset{*}{x}$ qui minimise le cout. Cette fonction étant telle que quelque soit x^i on a $\overset{*}{f}(\overset{*}{x}^i) = f(x^i)$.

Soit : $\forall\ x^i$ et $\overset{*}{x}^i$ tel que : $\overset{*}{x}^i = \dfrac{x^i - \bar{x}}{\sigma_x}$ et $\overset{*}{b} + \overset{*}{a}.\overset{*}{x}^i = b + a.x^i$ $\Leftrightarrow$ $a = \dfrac{\overset{*}{a}}{\sigma_x}$; $b = \overset{*}{b} - \dfrac{\overset{*}{a}.\bar{x}}{\sigma_x}$

Notation matricielle

Imaginons un jeu de m données d'apprentissage normalisées : m couples (x^i,y^i) on définit alors :

Le vecteur cible : $Y = \begin{bmatrix} y^1 \\ y^2 \\ .. \\ y^m \end{bmatrix}$ La matrice d'entrée : $X = \begin{bmatrix} 1 & x^1 \\ 1 & x^2 \\ .. & .. \\ 1 & x^m \end{bmatrix}$ le vecteur paramètres $\theta = \begin{bmatrix} b \\ a \end{bmatrix}$

On remarque alors que les prédictions peuvent se mettre dans un vecteur : $F(X) = \begin{bmatrix} b + a.x^1 \\ b + a.x^2 \\ ..... \\ b + a.x^m \end{bmatrix}$

Ce vecteur prédiction peut se calculer par le produit matriciel : $\mathbf{F(X) = X.\theta}$

Le coût de la fonction de prédiction est : $J(a,b) = \dfrac{1}{2.m} \sum_{i=1}^{m} (b + a.x^i - y^i)^2$

On définit le vecteur du gradient de descente : $\dfrac{\partial J(\theta)}{\partial \theta} = \begin{bmatrix} \dfrac{\partial J(\theta)}{\partial b} \\ \dfrac{\partial J(\theta)}{\partial a} \end{bmatrix} = \begin{bmatrix} \dfrac{1}{m} \sum_{i=1}^{m} 1.(b + a.x^i - y^i) \\ \dfrac{1}{m} \sum_{i=1}^{m} x^i.(b + a.x^i - y^i) \end{bmatrix}$

On a la transposée du vecteur X : $X^T = \begin{bmatrix} 1 & 1 & \dots & 1 \\ x^1 & x^2 & \dots & x^m \end{bmatrix}$ et le vecteur : $(X.\theta - Y) = \begin{bmatrix} b + a.x^1 - y^1 \\ b + a.x^2 - y^2 \\ ..... \\ b + a.x^m - y^m \end{bmatrix}$

Dont le produit matriciel est : $X^T.(X.\theta - Y) = \begin{bmatrix} \sum_{i=1}^{m} 1.(b + a.x^i - y^i) \\ \sum_{i=1}^{m} x^i.(b + a.x^i - y^i) \end{bmatrix}$

On en déduit alors la formule qui est le cœur de l'algorithme du gradient de descente :

$$\mathbf{\frac{\partial J(\theta)}{\partial \theta} = \frac{1}{m} . X^T . (X.\theta - Y)}$$

$-\alpha . \dfrac{\partial J(\theta)}{\partial \theta}$ étant le gradient de descente

La mise à jour du vecteur paramètres à la $i^{ième}$ itération s'obtient alors à l'aide de la relation matricielle :

$$\mathbf{\theta_{i+1} = \theta_i - \alpha . \frac{\partial J(\theta)}{\partial \theta} = \theta_i - \alpha . \frac{1}{m} . X^T . (X.\theta - Y)}$$

Régression multifacteurs

Nous avons maintenant une régression linéaire avec n paramètres d'entrée.

On dispose pour cela d'une base de données d'apprentissage constituée d'une liste de m « n+1-uplets » : $(x_1, x_2, \dots, x_n, y)$ où x_j sont les valeurs des n paramètres d'entrée et y la valeur cible.

La fonction de prédiction s'écrit alors :

$$f(x^i) = \omega_0 + \omega_1.x_1^i + \omega_2.x_2^i + \dots + \omega_n.x_n^i$$

D'où le vecteur paramètre : $\theta = \begin{bmatrix} \omega_0 \\ \omega_1 \\ \omega_2 \\ \dots \\ \omega_n \end{bmatrix}$ de dimension $(n+1) \times 1$

Vecteur des Données cibles	Matrice des données d'entrée On ajoute aux données d'entrainement une colonne de 1.	Vecteur des prédictions ($f(x^i)$) Fonction des paramètres de la régression vecteur de dimension (n + 1)	Vecteur des erreurs quadratiques Fonction de X, W et Y
y^1	$1 \quad x_1^1 \quad x_2^1 \quad \dots \quad x_n^1$	$\omega_0 + \omega_1.x_1^1 + \omega_2.x_2^1 + \dots + \omega_n.x_n^1$	$\left(f(x^1) - y^1\right)^2$
y^2	$1 \quad x_1^2 \quad x_2^2 \quad \dots \quad x_n^2$	$\omega_0 + \omega_1.x_1^2 + \omega_2.x_2^2 + \dots + \omega_n.x_n^2$	$\left(f(x^2) - y^2\right)^2$
…	… … … … …	……	……
y^m	$1 \quad x_1^m \quad x_2^m \quad \dots \quad x_n^m$	$\omega_0 + \omega_1.x_1^m + \omega_2.x_2^m + \dots + \omega_n.x_n^m$	$\left(f(x^m) - y^m\right)^2$
Y	X	$F(X) = X.\theta$	$E(\theta) = (X.\theta - Y)^2$
$m \times 1$	$m \times (n+1)$	$m \times 1$	$m \times 1$

Fonction cout

C'est la somme des erreurs quadratiques, pour les m données du jeu de données d'entrainement, entre les prédictions du modèle et les m valeurs cibles du jeu de données d'entrainement.

$$J(\theta) = \frac{1}{2.m} \sum_{i=1}^{m} \left(\omega_0 + \omega_1.x_1^i + \omega_2.x_2^i + \dots + \omega_n.x_n^i - y^i\right)^2 = \frac{1}{2.m} \sum_{i=1}^{m} \left[(X.\theta - Y)^2 \right]^i$$

Où les $\left[(X.\theta - Y)^2 \right]^i$ sont les valeurs du vecteur colonne des erreurs quadratiques $E(\theta)$.

Gradient de descente

C'est un vecteur dont les coordonnées sont les taux de variation de la fonction cout en fonction de chacun des paramètres du modèle de prédiction. Ces coordonnées sont donc les dérivées partielles de la fonction coût en fonction des (n+1) paramètres du modèle de prédiction.

$$\frac{\partial J(\theta)}{\partial \theta} = \begin{bmatrix} \frac{\partial J(\theta)}{\partial \omega_0} = \frac{1}{m} . \sum_{i=1}^{m} 1 . \left(\omega_0 + \omega_1.x_1^i + \omega_2.x_2^i + \dots + \omega_n.x_n^i - y^i\right) \\ \frac{\partial J(\theta)}{\partial \omega_1} = \frac{1}{m} . \sum_{i=1}^{m} x_1^i . \left(\omega_0 + \omega_1.x_1^i + \omega_2.x_2^i + \dots + \omega_n.x_n^i - y^i\right) \\ \frac{\partial J(\theta)}{\partial \omega_2} = \frac{1}{m} . \sum_{i=1}^{m} x_2^i . \left(\omega_0 + \omega_1.x_1^i + \omega_2.x_2^i + \dots + \omega_n.x_n^i - y^i\right) \\ \dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots \\ \frac{\partial J(\theta)}{\partial \omega_n} = \frac{1}{m} . \sum_{i=1}^{m} x_n^i . \left(\omega_0 + \omega_1.x_1^i + \omega_2.x_2^i + \dots + \omega_n.x_n^i - y^i\right) \end{bmatrix}$$

Finalement on a : $\frac{\partial J(\theta)}{\partial \theta} = \frac{1}{m} . X^T . (X.\theta - Y)$ Vecteur gradient de dimension $(n+1) \times 1$

Puis la mise à jour des paramètres de la régression à la $i^{ième}$ itération :

$$\theta_{i+1} = \theta_i - \alpha . \frac{\partial J(\theta)}{\partial \theta} = \theta_i - \alpha . \frac{1}{m} . X^T . (X.\theta - Y)$$

Régression polynômiale

Nous avons maintenant une régression avec n paramètres d'entrée.

On dispose pour cela d'une base de données d'apprentissage constituée d'une liste de m couples : (x,y) où x_j est la valeur du paramètre d'entrée et y la valeur cible.

L'observation de la répartition des points du jeu de données d'entrainement s'apparente davantage à une répartition polynomiale de degré n.

C'est-à-dire que la fonction de prédiction la mieux adaptée serait du type :

$$f(x^i) = \omega_0 + \omega_1.(x^i) + \omega_2.(x^i)^2 + \ldots + \omega_n.(x^i)^n$$

On se ramène alors à une régression linéaire à n paramètres d'entrée où le $j^{ième}$ paramètre d'entré de la $i^{ième}$ donnée est la puissance $j^{ième}$ du $i^{ième}$ paramètre d'entrée du jeu de données d'entrainement :

$$x_j^i = (x^i)^j$$

On utilise donc la matrice de données d'entrée X

1	(x^1)	$(x^1)^2$	…	$(x^1)^n$
1	(x^2)	$(x^2)^2$	…	$(x^2)^n$
…	…	…	…	…
1	(x^m)	$(x^m)^2$	…	$(x^m)^n$
X				
$m \times (n+1)$				

Le reste de l'algorithme reste identique

à celui de la régression linéaire multi-facteurs