

# Chapitre 3: Principe de la descente de gradient pour l'apprentissage supervisé: Application régression linéaire et régression logistique

Mahdi LOUATI

3 Ingénierie des Données et Systèmes Décisionnels  
Sciences de Données 3 IDSD-SD  
Ecole Nationale d'Electronique et des télécommunications de Sfax

04 Octobre 2022



# Plan

- 1 Introduction
- 2 Algorithme de gradient: Démarche itérative pour minimiser une fonction
  - Optimisation d'une fonction différentiable et convexe
  - Algorithme du gradient (descente de gradient)
- 3 Rapport avec apprentissage automatique: Cas régression linéaire multiple
- 4 Descente de gradient stochastique (Approche incrémentale pour le traitement des bases très grandes)
  - Correction par observation (online)
  - Stratégies - Gradient stochastique
- 5 Choix du taux d'apprentissage (Taux fixe ou taux décroissant au fil du processus d'apprentissage)
- 6 Rapport avec apprentissage automatique: Cas régression logistique
  - Régression logistique binaire
  - Régression logistique multinomiale
- 7 Logiciels: Quelques packages pour Python et R

## Motivation

La descente de gradient (stochastique) n'est pas un concept nouveau (ADALINE, 1960), mais elle connaît un très grand intérêt aujourd'hui,

- Pour l'entraînement des réseaux de neurones profonds (deep learning).
- Elle permet aussi de revisiter des approches statistiques existantes (exp. régression).

# Plan

- 1 Introduction
- 2 Algorithme de gradient: Démarche itérative pour minimiser une fonction
  - Optimisation d'une fonction différentiable et convexe
  - Algorithme du gradient (descente de gradient)
- 3 Rapport avec apprentissage automatique: Cas régression linéaire multiple
- 4 Descente de gradient stochastique (Approche incrémentale pour le traitement des bases très grandes)
  - Correction par observation (online)
  - Stratégies - Gradient stochastique
- 5 Choix du taux d'apprentissage (Taux fixe ou taux décroissant au fil du processus d'apprentissage)
- 6 Rapport avec apprentissage automatique: Cas régression logistique
  - Régression logistique binaire
  - Régression logistique multinomiale
- 7 Logiciels: Quelques packages pour Python et R

Maximiser ou minimiser une fonction est un problème usuel dans de nombreux domaines.

### Exemple

- On désire minimiser la fonction

$$f(x) = x^2 - x + 1$$

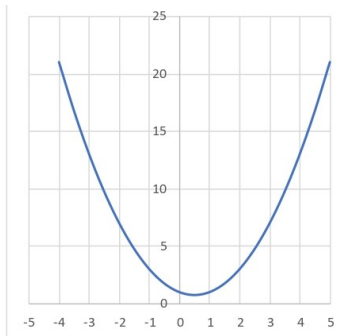
par rapport à  $x$ .

- $f$  est la fonction à minimiser et  $x$  joue le rôle de paramètre ici, i.e., on cherche la valeur de  $x$  qui minimise  $f$ .

- 

$$\min_{x \in \mathbb{R}} f(x) = \min_{x \in \mathbb{R}} (x^2 - x + 1).$$

## Exemple



La solution analytique passe par :

$$f'(x) = 0$$

En s'assurant que  $f''(x) > 0$



$$f'(x) = 2x - 1 = 0 \Rightarrow x^* = \frac{1}{2}$$

## Remarque

Parfois la résolution analytique n'est pas possible, parce que

- le nombre de paramètres est élevé par exemple
- le calcul serait trop coûteux.

⇒ **Approximation avec une approche itérative.**

## Algorithme du gradient

- 1 Initialiser avec  $x_0$  (au hasard).
- 2 Répéter

$$x_{t+1} = x_t - \eta \nabla f(x_t).$$

- 3 Jusqu'à convergence.

## Remarques (Algorithme du gradient)

- 1 **Initialisation**: Quasiment impossible de suggérer des valeurs "intelligentes".
- 2  $\nabla f$ : Le "gradient" généralisation multidimensionnelle de la dérivée.
- 3  $\eta$ : Un paramètre qui permet de moduler la correction ( $\eta$  trop faible  $\rightarrow$  lenteur de convergence et  $\eta$  trop élevé  $\rightarrow$  oscillation).
- 4  $-$ : puisqu'on cherche à minimiser  $f$ .
- 5 **Jusqu'à convergence**: Nombre d'itérations fixé, ou différence entre deux valeurs successives  $x_t$ , ou  $\nabla f(x_t)$  très petit.



## Exemple

$$\begin{cases} f(x) &= x^2 - x - 1 \\ \nabla f(x) = \frac{\partial f(x)}{\partial x} &= 2x - 1. \end{cases}$$

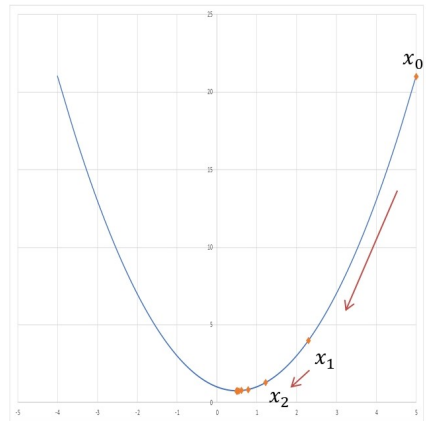
Il n'y a qu'un seul paramètre, la dérivée partielle est égale à la dérivée.

eta 0.3  $\eta = 0.3$

$x_0 = 5$

x	$f'(x_t)$	$f(x)$
5.0000		21.0000
2.3000	9.0000	3.9900
1.2200	3.6000	1.2684
0.7880	1.4400	0.8329
0.6152	0.5760	0.7633
0.5461	0.2304	0.7521
0.5184	0.0922	0.7503
0.5074	0.0369	0.7501
0.5029	0.0147	0.7500
0.5012	0.0059	0.7500
0.5005	0.0024	0.7500
0.5002	0.0009	0.7500
0.5001	0.0004	0.7500
0.5000	0.0002	0.7500

$x_{t+1} = x_t - \eta \times \nabla f(x_t)$



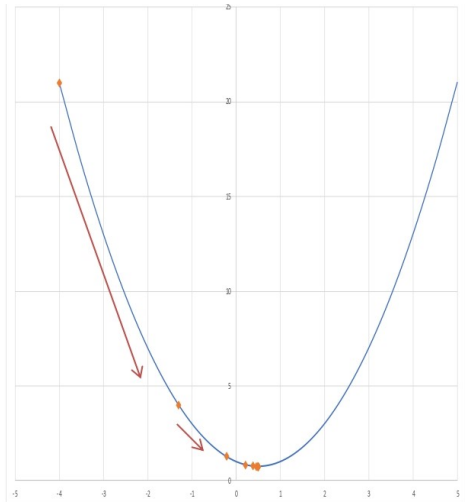
$\dots x_2 < x_1 < x_0$

## Remarque

On aurait pu partir de l'autre côté

eta 0.3

x	f'(x <sub>t</sub> )	f(x)
-4.0000		21.0000
-1.3000	-9.0000	3.9900
-0.2200	-3.6000	1.2684
0.2120	-1.4400	0.8329
0.3848	-0.5760	0.7633
0.4539	-0.2304	0.7521
0.4816	-0.0922	0.7503
0.4926	-0.0369	0.7501
0.4971	-0.0147	0.7500
0.4988	-0.0059	0.7500
0.4995	-0.0024	0.7500
0.4998	-0.0009	0.7500
0.4999	-0.0004	0.7500
0.5000	-0.0002	0.7500



$$x_0 < x_1 < x_2 \dots$$

# Plan

- 1 Introduction
- 2 Algorithme de gradient: Démarche itérative pour minimiser une fonction
  - Optimisation d'une fonction différentiable et convexe
  - Algorithme du gradient (descente de gradient)
- 3 **Rapport avec apprentissage automatique: Cas régression linéaire multiple**
- 4 Descente de gradient stochastique (Approche incrémentale pour le traitement des bases très grandes)
  - Correction par observation (online)
  - Stratégies - Gradient stochastique
- 5 Choix du taux d'apprentissage (Taux fixe ou taux décroissant au fil du processus d'apprentissage)
- 6 Rapport avec apprentissage automatique: Cas régression logistique
  - Régression logistique binaire
  - Régression logistique multinomiale
- 7 Logiciels: Quelques packages pour Python et R

Objectif: Modéliser  $y$  (quantitative) à partir de  $p$  variables explicatives  $X = (x_1, x_2, \dots, x_p)$  quantitatives

$$y_i = a_0 + a_1x_{i1} + a_2x_{i2} + \dots + a_px_{ip} + \varepsilon_i, \text{ pour tout } i \in \{1, 2, \dots, n\}.$$



Sur un échantillon de taille  $n$ , on cherche à minimiser le critère

$$S = \frac{1}{2n} \sum_{i=1}^n \varepsilon_i^2 \quad \text{Critère des moindres carrés}$$



On a un problème de minimisation par rapport aux paramètres  $a = (a_0, a_1, \dots, a_p)$

$$\min_{a_0, a_1, \dots, a_p} S = \sum_{i=1}^n (y_i - \langle a, x_i \rangle)^2$$

Où  $x_i = (x_{0i}, x_{i1}, \dots, x_{ip})$  et (constante)  $x_{0i} = 1, \forall i$



Il existe une solution « exacte »

$$\hat{a} = (X'X)^{-1}X'Y$$

## L'estimateur

$$\hat{a} = (X'X)^{-1}X'Y$$

nécessite la manipulation de la matrice  $(X'X)$  de taille  $(p+1, p+1)$ , ingérable dès que  $p$  est élevé (millier de variables, grandes dimensions).

D'autres approches existent, mais elle nécessitent toujours la manipulation d'une matrice de taille  $(p+1, p+1)$  durant les calculs.

Descente de gradient



$$a^{t+1} = a^t - \eta \times \nabla S^t$$



Taux d'apprentissage ( $\eta$ , learning rate)

Où  $\nabla S^t =$

$$\begin{cases} \frac{\partial S^t}{\partial a_0} = \frac{1}{n} \sum_{i=1}^n (-1) \times (y_i - \langle a^t, x_i \rangle) \\ \frac{\partial S^t}{\partial a_1} = \frac{1}{n} \sum_{i=1}^n (-x_{i1}) \times (y_i - \langle a^t, x_i \rangle) \\ \vdots \\ \frac{\partial S^t}{\partial a_p} = \frac{1}{n} \sum_{i=1}^n (-x_{ip}) \times (y_i - \langle a^t, x_i \rangle) \end{cases}$$



Le vecteur gradient de taille  $(p+1, 1)$  est composé des dérivées partielles de  $S$  par rapport à chaque paramètre du modèle.

## Remarques

- Manipulation d'un vecteur de taille  $(p + 1, 1)$  plutôt qu'une matrice  $(p + 1, p + 1)$ , voilà tout l'intérêt de la descente de gradient pour les grandes dimensions.
- Nécessite de parcourir plusieurs fois la base (avec  $n$  observations).

## Exemple

Il est préférable d'harmoniser les données (les mettre sous la même échelle), on peut utiliser la standardisation ou la normalisation, afin d'éviter les problèmes d'échelles.

x0	x1	x2	y
1	0.72	0.32	6.93
1	0.75	0.12	5.99
1	0.53	0.65	1.46
1	0.27	0.82	1.44
1	0.49	0.15	4.51
1	0.02	0.19	1.25
1	0.35	0.87	2.53
1	0.99	0.71	6.88
1	0.98	0.92	6.25
1	0.73	0.19	6.36

On pose

$$a^0 = (0.1, 0.1, 0.1) \text{ et } \eta = 1.05.$$

eta		1.05					
t	a0	a1	a2	S	dS/d ao	dS/d a1	dS/d a2
0	0.100	0.100	0.100	224.72	-4.152	-3.003	-1.889
1	4.460	3.253	2.083	127.71	3.026	1.483	1.892
2	1.283	1.697	0.097	77.53	-2.040	-1.633	-0.825
3	3.425	3.411	0.963	50.95	1.529	0.609	1.045
4	1.819	2.771	-0.135	36.36	-0.992	-0.931	-0.315
5	2.860	3.749	0.196	27.95	0.783	0.193	0.606
6	2.039	3.546	-0.440	22.79	-0.472	-0.564	-0.078
7	2.534	4.138	-0.358	19.39	0.410	0.002	0.373
8	2.104	4.135	-0.750	17.00	-0.216	-0.367	0.026
9	2.330	4.521	-0.778	15.22	0.222	-0.079	0.245
10	2.097	4.604	-1.035	13.83	-0.090	-0.257	0.067
11	2.191	4.874	-1.105	12.72	0.127	-0.108	0.171
12	2.058	4.987	-1.284	11.80	-0.029	-0.191	0.078
13	2.088	5.188	-1.365	11.04	0.078	-0.112	0.125
14	2.006	5.306	-1.497	10.41	0.000	-0.149	0.075
15	2.006	5.463	-1.576	9.87	0.052	-0.106	0.096
16	1.951	5.574	-1.676	9.42	0.013	-0.121	0.068
17	1.938	5.701	-1.747	9.03	0.038	-0.096	0.075
18	1.898	5.802	-1.826	8.71	0.018	-0.100	0.059
19	1.879	5.907	-1.888	8.43	0.030	-0.085	0.060
20	1.847	5.996	-1.951	8.20	0.019	-0.084	0.050
21	1.827	6.084	-2.003	8.00	0.025	-0.074	0.048
22	1.801	6.161	-2.054	7.83	0.019	-0.071	0.042
23	1.782	6.236	-2.098	7.68	0.021	-0.064	0.039
24	1.759	6.303	-2.139	7.56	0.018	-0.061	0.035
25	1.741	6.367	-2.175	7.46	0.018	-0.055	0.032
26	1.722	6.425	-2.209	7.37	0.016	-0.052	0.029
27	1.705	6.479	-2.239	7.29	0.016	-0.047	0.026
28	1.688	6.529	-2.267	7.23	0.015	-0.044	0.024
29	1.673	6.575	-2.292	7.17	0.014	-0.041	0.022
30	1.658	6.618	-2.314	7.12	0.013	-0.038	0.019



## Exemple



$$a^0 = (0.1, 0.1, 0.1)$$



$$a^{30} = (1.658, 6.618, -2.314)$$



$$a^{solution} = (1.424, 7.173, -2.523).$$

- La convergence est lente puisqu'on a un petit effectif ( $n = 10$ ).

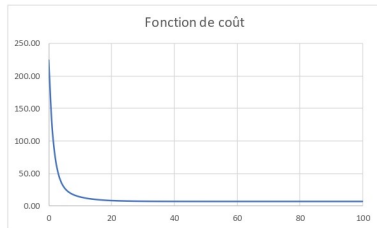
eta		1.05					
t	a0	a1	a2	S	dS/d ao	dS/d a1	dS/d a2
0	0.100	0.100	0.100	224.72	-4.152	-3.003	-1.889
1	4.460	3.253	2.083	127.71	3.026	1.483	1.892
2	1.283	1.697	0.097	77.53	-2.040	-1.633	-0.825
3	3.425	3.411	0.963	50.95	1.529	0.609	1.045

$$a^0 = (0.1, 0.1, 0.1) \Rightarrow \nabla S^0 = \begin{pmatrix} -4.152 \\ -3.003 \\ -1.889 \end{pmatrix}$$

$$\begin{pmatrix} 0.1 \\ 0.1 \\ 0.1 \end{pmatrix} - 1.05 \times \begin{pmatrix} -4.152 \\ -3.003 \\ -1.889 \end{pmatrix} \rightarrow \begin{pmatrix} 4.460 \\ 3.253 \\ 2.083 \end{pmatrix} = a^1$$

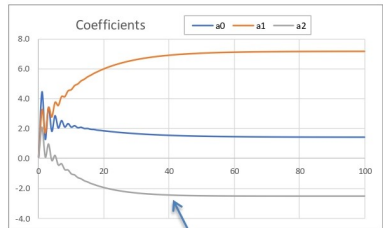
## Exemple

Evolution de S au fil des itérations (t)



Baisse constante de la  
fonction de coût.

Evolution des coefficients au fil des itérations (t)



Oscillations au départ parce  
 $\eta=1.05$  choisi très élevé

Solution acceptable dès  $t \approx 40$

## La fonction de coût

$$J(t) = \frac{1}{2n} \sum_{i=1}^n (Y_i - \hat{Y}_i(t))^2.$$

# Plan

- 1 Introduction
- 2 Algorithme de gradient: Démarche itérative pour minimiser une fonction
  - Optimisation d'une fonction différentiable et convexe
  - Algorithme du gradient (descente de gradient)
- 3 Rapport avec apprentissage automatique: Cas régression linéaire multiple
- 4 Descente de gradient stochastique (Approche incrémentale pour le traitement des bases très grandes)
  - Correction par observation (online)
  - Stratégies - Gradient stochastique
- 5 Choix du taux d'apprentissage (Taux fixe ou taux décroissant au fil du processus d'apprentissage)
- 6 Rapport avec apprentissage automatique: Cas régression logistique
  - Régression logistique binaire
  - Régression logistique multinomiale
- 7 Logiciels: Quelques packages pour Python et R

## Principe

Gradient stochastique est une approximation de la descente de gradient, applicable lorsque la fonction objectif s'écrit comme une somme de fonctions dérivables: c'est très souvent le cas en apprentissage supervisé.

Exemple de la régression linéaire multiple via les moindres carrés

$$S = \sum_{i=1}^n (y_i - \langle a, x_i \rangle)^2$$



$$(y_i - \langle a, x_i \rangle)^2$$

Est dérivable par rapport au paramètres ( $a_j$ )

Il est possible de corriger les paramètres estimés pour le passage de chaque observation



$$a := a - \eta \times \nabla S_i$$

$$\text{Où } \frac{\partial S_i}{\partial a_j} = (-x_{ij}) \times (y_i - \langle a, x_i \rangle)$$

## Exemple

eta 0.5

x0	x1	x2	y
1	0.72	0.32	6.93
1	0.75	0.12	5.99
1	0.53	0.65	1.46
1	0.27	0.82	1.44
1	0.49	0.15	4.51
1	0.02	0.19	1.25
1	0.35	0.87	2.53
1	0.99	0.71	6.88
1	0.98	0.92	6.25
1	0.73	0.19	6.36

i	a0	a1	a2	S	dS/d_ao	dS/d_a1	dS/d_a2
0	0.100	0.100	0.100	224.72	-6.726	-4.843	-2.152
1	3.463	2.521	1.176	47.83	-0.495	-0.371	-0.059
2	3.710	2.707	1.206	56.63	4.469	2.369	2.905
3	1.476	1.523	-0.247	81.22	0.245	0.066	0.201
4	1.354	1.490	-0.347	89.71	-2.479	-1.215	-0.372
5	2.593	2.097	-0.161	35.50	1.354	0.027	0.257
6	1.916	2.083	-0.290	50.15	-0.137	-0.048	-0.119
7	1.984	2.107	-0.230	47.19	-2.973	-2.943	-2.111
8	3.471	3.579	0.825	51.62	1.487	1.457	1.368
9	2.727	2.850	0.141	27.12	-1.526	-1.114	-0.290
10	3.490	3.407	0.286	39.93	-0.896	-0.645	-0.287
11	3.938	3.729	0.429	61.57	0.796	0.597	0.096
12	3.540	3.431	0.382	43.13	4.146	2.197	2.695
13	1.467	2.332	-0.966	68.10	-0.136	-0.037	-0.111
14	1.534	2.350	-0.910	63.90	-1.960	-0.961	-0.294
15	2.515	2.831	-0.763	27.57	1.176	0.024	0.223
16	1.927	2.819	-0.875	39.08	-0.378	-0.132	-0.329
17	2.116	2.885	-0.710	32.38	-2.413	-2.389	-1.713
18	3.322	4.079	0.146	39.90	1.204	1.180	1.108
19	2.720	3.489	-0.408	21.36	-1.170	-0.854	-0.222
20	3.305	3.917	-0.297	30.72	-0.900	-0.648	-0.288
21	3.755	4.241	-0.153	50.09	0.927	0.695	0.111
22	3.291	3.893	-0.208	31.51	3.759	1.992	2.444
23	1.412	2.897	-1.430	61.38	-0.419	-0.113	-0.343
24	1.621	2.953	-1.258	49.38	-1.631	-0.799	-0.245
25	2.436	3.353	-1.136	22.86	1.038	0.021	0.197
26	1.918	3.342	-1.235	32.14	-0.517	-0.181	-0.450
27	2.176	3.433	-1.010	24.43	-2.023	-2.002	-1.436
28	3.187	4.434	-0.292	32.71	1.014	0.994	0.933
29	2.680	3.937	-0.758	18.09	-0.950	-0.693	-0.180

## Remarques

- La décroissance de  $S$  au passage de chaque observation n'est pas assurée mais, en moyenne, sur l'ensemble des observations, sa convergence est effective.
- Dans l'exemple précédent (descente de gradient), après 3 passages sur les observations nous avons  $S = 50.95$ . Ici nous obtenons  $S = 18.09$  ( $\eta$  n'est pas le même non plus).

## Stratégies

- **Descente de Gradient classique (batch gradient descent)**. On fait passer la totalité des observations, le gradient est calculé, les coefficients sont corrigés.
- **Online**. Gradient calculé pour chaque observation, correction des coefficients.
- **Mini-batch (traitement par lots)**. On fait passer un groupe (effectif = paramètre de l'algorithme) d'observations. Calcul du gradient. Correction des coefficients.

## Remarques

- Le traitement par lots permet **d'améliorer la convergence en réduisant le nombre de passage sur la base entière.**

## Stratégies

- **Descente de Gradient classique (batch gradient descent)**. On fait passer la totalité des observations, le gradient est calculé, les coefficients sont corrigés.
- **Online**. Gradient calculé pour chaque observation, correction des coefficients.
- **Mini-batch (traitement par lots)**. On fait passer un groupe (effectif = paramètre de l'algorithme) d'observations. Calcul du gradient. Correction des coefficients.

## Remarques

- Le traitement par lots permet **d'améliorer la convergence en réduisant le nombre de passage sur la base entière**.
- Il permet également **de se contenter de charger partiellement les données en mémoire au fur et à mesure**.



# Plan

- 1 Introduction
- 2 Algorithme de gradient: Démarche itérative pour minimiser une fonction
  - Optimisation d'une fonction différentiable et convexe
  - Algorithme du gradient (descente de gradient)
- 3 Rapport avec apprentissage automatique: Cas régression linéaire multiple
- 4 Descente de gradient stochastique (Approche incrémentale pour le traitement des bases très grandes)
  - Correction par observation (online)
  - Stratégies - Gradient stochastique
- 5 Choix du taux d'apprentissage (Taux fixe ou taux décroissant au fil du processus d'apprentissage)
- 6 Rapport avec apprentissage automatique: Cas régression logistique
  - Régression logistique binaire
  - Régression logistique multinomiale
- 7 Logiciels: Quelques packages pour Python et R

## Importance du taux d'apprentissage (learning rate)

- $\eta$  détermine la vitesse de convergence du processus d'apprentissage.
- Améliorer le dispositif en faisant évoluer  $\eta$  au fil des itérations
  - Fort au début  $\implies$  accélérer la convergence
  - Faible à la fin  $\implies$  améliorer la précision.

## Exemple: SGDRegressor du package "scikit-learn" (Stochastic Gradient Descent Python)

```
class sklearn.linear_model. SGDRegressor (loss='squared_loss', penalty='l2', alpha=0.0001, l1_ratio=0.15,
fit_intercept=True, max_iter=None, tol=None, shuffle=True, verbose=0, epsilon=0.1, random_state=None,
learning_rate='invscaling', eta0=0.01, power_t=0.25, warm_start=False, average=False, n_iter=None)
```

### **loss : str, default='squared\_error'**

The loss function to be used. The possible values are 'squared\_error', 'huber', 'epsilon\_insensitive', or 'squared\_epsilon\_insensitive'

The 'squared\_error' refers to the ordinary least squares fit. 'huber' modifies 'squared\_error' to focus less on getting outliers correct by switching from squared to linear loss past a distance of epsilon. 'epsilon\_insensitive' ignores errors less than epsilon and is linear past that; this is the loss function used in SVR. 'squared\_epsilon\_insensitive' is the same but becomes squared loss past a tolerance of epsilon.

More details about the losses formulas can be found in the [User Guide](#).

*Deprecated since version 1.0:* The loss 'squared\_loss' was deprecated in v1.0 and will be removed in version 1.2. Use `loss='squared_error'` which is equivalent.

### **penalty : {'l2', 'l1', 'elasticnet'}, default='l2'**

The penalty (aka regularization term) to be used. Defaults to 'l2' which is the standard regularizer for linear SVM models. 'l1' and 'elasticnet' might bring sparsity to the model (feature selection) not achievable with 'l2'.

### **alpha : float, default=0.0001**

Constant that multiplies the regularization term. The higher the value, the stronger the regularization. Also used to compute the learning rate when set to `learning_rate` is set to 'optimal'.

### **l1\_ratio : float, default=0.15**

The Elastic Net mixing parameter, with  $0 \leq \text{l1\_ratio} \leq 1$ . `l1_ratio=0` corresponds to L2 penalty, `l1_ratio=1` to L1. Only used if `penalty` is 'elasticnet'.

## Exemple: SGDRegressor du package "scikit-learn" (Stochastic Gradient Descent Python)

```
class sklearn.linear_model. SGDRegressor (loss='squared_loss', penalty='l2', alpha=0.0001, l1_ratio=0.15,
fit_intercept=True, max_iter=None, tol=None, shuffle=True, verbose=0, epsilon=0.1, random_state=None,
learning_rate='invscaling', eta0=0.01, power_t=0.25, warm_start=False, average=False, n_iter=None)
```

### **fit\_intercept : bool, default=True**

Whether the intercept should be estimated or not. If False, the data is assumed to be already centered.

### **max\_iter : int, default=1000**

The maximum number of passes over the training data (aka epochs). It only impacts the behavior in the `fit` method, and not the `partial_fit` method.

*New in version 0.19.*

### **tol : float, default=1e-3**

The stopping criterion. If it is not None, training will stop when  $(\text{loss} > \text{best\_loss} - \text{tol})$  for `n_iter_no_change` consecutive epochs. Convergence is checked against the training loss or the validation loss depending on the `early_stopping` parameter.

*New in version 0.19.*

### **shuffle : bool, default=True**

Whether or not the training data should be shuffled after each epoch.

**learning\_rate** : string, optional

The learning rate schedule:

- 'constant': eta = eta0
- 'optimal': eta = 1.0 / (alpha \* (t + t0)) [default]
- 'invscaling': eta = eta0 / pow(t, power\_t)

where t0 is chosen by a heuristic proposed by Leon Bottou.

**eta0** : double, optional

The initial learning rate [default 0.01].

**power\_t** : double, optional

The exponent for inverse scaling learning rate [default 0.25].

(t<sub>0</sub>???) La documentation n'est pas très  
disserte à ce sujet.

$$\eta = \frac{\eta_0}{t^{0.25}} \quad 0.25 \text{ étant lui-même modifiable}$$

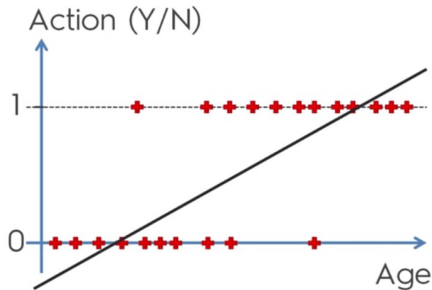
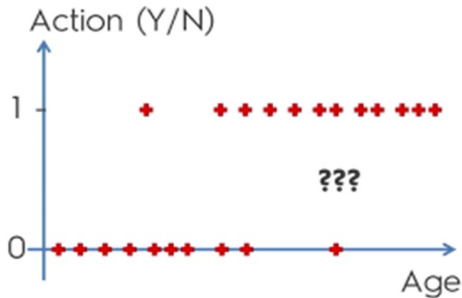
## Remarque

En tous les cas, il est acquis que évoluer  $\eta$  au fil des itérations sur la base  $t$  améliore l'efficacité du dispositif.

# Plan

- 1 Introduction
- 2 Algorithme de gradient: Démarche itérative pour minimiser une fonction
  - Optimisation d'une fonction différentiable et convexe
  - Algorithme du gradient (descente de gradient)
- 3 Rapport avec apprentissage automatique: Cas régression linéaire multiple
- 4 Descente de gradient stochastique (Approche incrémentale pour le traitement des bases très grandes)
  - Correction par observation (online)
  - Stratégies - Gradient stochastique
- 5 Choix du taux d'apprentissage (Taux fixe ou taux décroissant au fil du processus d'apprentissage)
- 6 Rapport avec apprentissage automatique: Cas régression logistique
  - Régression logistique binaire
  - Régression logistique multinomiale
- 7 Logiciels: Quelques packages pour Python et R

Nous sommes dans le cadre de l'apprentissage supervisé où la variable cible  $y$  est binaire i.e.,  $y \in \{0; 1\}$ .





$$Y = b_0 + b_1 X$$



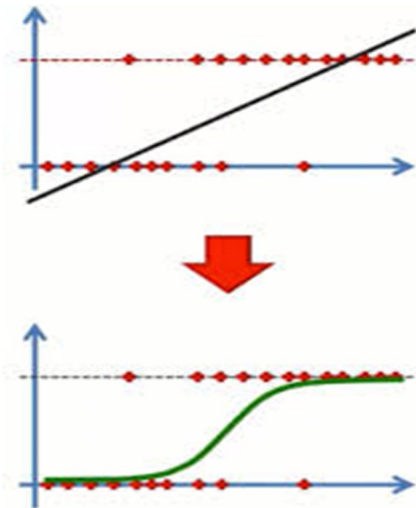
$$P = \frac{1}{1 + e^{-Y}}$$

Sigmoid function



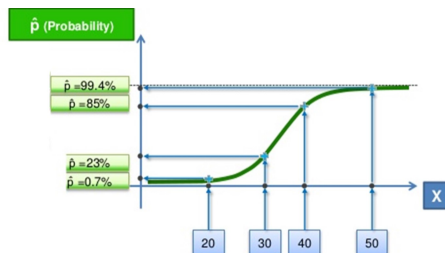
$$\ln\left(\frac{P}{1-P}\right) = b_0 + b_1 X$$

Equation of the Logistic Regression



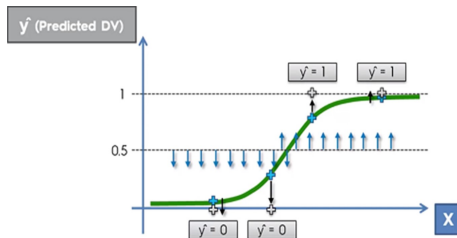
## Remarque

On transforme la régression linéaire en une courbe de régression logistique.



## Remarque

Calculer la proba. d'acheter le produit  $\implies$  prédire la décision du client.



## Fonction de perte: Binary cross-entropy

$$J(a) = - \underbrace{\frac{1}{n} \sum_{i=1}^n y_i \ln(p_i) + (1 - y_i) \ln(1 - p_i)}_{\text{la log-vraisemblance du modèle binomial}},$$

$p_i$  proba. condi.  $\mathbb{P}(Y | X)$  modélisée avec la régression logistique

$$p_i = \frac{1}{1 + e^{-(a_0 + a_1 x_{i1} + a_2 x_{i2} + \dots + a_p x_{ip})}}.$$

## Gradient

$$\frac{\partial J}{\partial a_j} = \frac{1}{n} \sum_{i=1}^n x_{ij} (y_i - p_i).$$

## Gradient stochastique

La fonction de perte s'écrit comme une somme de fonctions dérivables

$\implies$  l'approche du gradient stochastique peut s'appliquer.

```
import pandas as pd
dataset = pd.read_csv('Social_Network_Ads.csv')
```

Index	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0
5	15728773	Male	27	58000	0
6	15598044	Female	27	84000	0
7	15694829	Female	32	150000	1
8	15600575	Male	25	33000	0
9	15727311	Female	35	65000	0
10	15570769	Female	26	80000	0
11	15606274	Female	26	52000	0
12	15746139	Male	20	86000	0
13	15704987	Male	32	18000	0
14	15628972	Male	18	82000	0
15	15697686	Male	29	80000	0

```

X = dataset.iloc[:, [2, 3]]
y = dataset.iloc[:, 4].values
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25)
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression()
classifier.fit(X_train, y_train)
Y_pred=classifier.predict(X_test)

```

y_pred	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	1	0	0	1	.	.	1	0	1	0	0	0	1	0	0	0	0	0	1	1
y_test	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	1	.	.	1	0	1	1	0	0	1	0	0	0	1	0	1	1

## Matrice de Confusion

Dans l'analyse prédictive, une matrice de confusion est un tableau à deux lignes et deux colonnes qui indique le nombre de faux positifs (FP), de faux négatifs (FN), de vrais positifs (VP) et de vrais négatifs (VN).

		Actual	
		Positive	Negative
Predicted	Positive	<b>True Positive</b>	<b>False Positive</b>
	Negative	<b>False Negative</b>	<b>True Negative</b>

- **VP** nombre de cas prédits positifs correctement identifiés.
- **FP** nombre de cas prédits positifs classés de manière incorrecte.
- **VN** nombre de cas prédits négatifs correctement classés.
- **FN** nombre de cas prédits négatifs classés de manière incorrecte.

## Mesures de performance

- **Sensibilité** (Rappel, Recall, Sensitivity, True Positive Rate (TPR)) représente la proportion des vrais positifs si la réponse du client est réellement positive. Elle est donnée par

$$\text{Sensibilité} = \frac{VP}{VP + FN} = \frac{VP}{P}.$$

- **Spécificité** (specificity, selectivity, True Negative Rate (TNR)) représente la proportion des vrais négatifs si la réponse du client est réellement négative. Elle est donnée par

$$\text{Spécificité} = \frac{VN}{VN + FP} = \frac{VN}{N}.$$

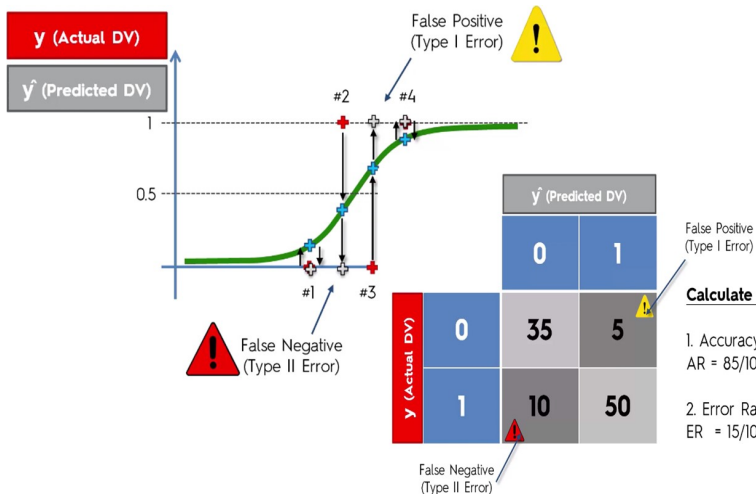
- **Précision** (Positive Predictive Value (PPV))

$$\text{Précision} = \frac{VP}{VP + FP}.$$

## Mesures de performance

- Accuracy** représente la proportion des prédictions correctes.

$$\text{Accuracy} = \frac{VP + VN}{VP + FP + VN + FN}$$





```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
```

y_pred	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	1	0	0	1	.	.	1	0	1	0	0	0	1	0	0	0	0	0	1	1
y_test	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	1	.	.	1	0	1	1	0	0	1	0	0	0	1	0	1	1

$$cm = \begin{pmatrix} 63 & 5 \\ 7 & 25 \end{pmatrix}$$

## Remarques

- Après avoir exécuté le code, nous obtenons  $63 + 25 = 88$  prédictions correctes et  $5 + 7 = 12$  prédictions incorrectes.
- L'Accuracy du modèle est donc 88%.

## Remarque

L'Accuracy n'est pas une mesure fiable des performances réelles d'un classificateur, car elle produira des résultats trompeurs si l'ensemble de données est déséquilibré (i.e., lorsque le nombre d'observations dans différentes classes varie considérablement). Par exemple, s'il y avait 95 chats et seulement 5 chiens dans les données, un classificateur particulier pourrait classer toutes les observations comme des chats.

## Visualisation

```
from matplotlib.colors import ListedColormap
X_set, y_set = X_test, y_test
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set[:, 0].max() + 1, step = 0.01),
                     np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].max() + 1, step = 0.01))
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),
             alpha = 0.4, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
               c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('Logistic Regression ( Test set )')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
```



La régression logistique est une régression linéaire, c'est pourquoi le séparateur est une droite.

$y$  est catégorielle et peut prendre  $K$  modalités i.e.,  $y \in \{y_1, y_2, \dots, y_k\}$ .

## Codage

Codage en  $K$  indicatrices 0/1

Ex.

Y	Y_A	Y_B	Y_C
A	1	0	0
A	1	0	0
B	0	1	0
A	1	0	0
C	0	0	1
A	1	0	0

Fonction de classement:  $a$  est une matrice de dimension  $K \times p + 1$


Fonction de perte: Categorical cross-entropy

$$J(a) = -\frac{1}{n} \sum_{i=1}^n \sum_{k=1}^K y_{ik} \ln(p_{ik}).$$

Gradient: Le vecteur gradient est un vecteur de dimension  $K \times p + 1$

# Plan

- 1 Introduction
- 2 Algorithme de gradient: Démarche itérative pour minimiser une fonction
  - Optimisation d'une fonction différentiable et convexe
  - Algorithme du gradient (descente de gradient)
- 3 Rapport avec apprentissage automatique: Cas régression linéaire multiple
- 4 Descente de gradient stochastique (Approche incrémentale pour le traitement des bases très grandes)
  - Correction par observation (online)
  - Stratégies - Gradient stochastique
- 5 Choix du taux d'apprentissage (Taux fixe ou taux décroissant au fil du processus d'apprentissage)
- 6 Rapport avec apprentissage automatique: Cas régression logistique
  - Régression logistique binaire
  - Régression logistique multinomiale
- 7 Logiciels: Quelques packages pour Python et R



[Home](#)
[Installation](#)
[Documentation](#)
[Examples](#)

Previous  
1.4. Support

Next  
1.6. Nearest

Up  
1. Supervised...

scikit-learn v0.19.1  
Other versions

Please **cite us** if you use the software.

## 1.5. Stochastic Gradient Descent

**Stochastic Gradient Descent (SGD)** is a simple yet very efficient approach to discriminative learning of linear classifiers under convex loss functions such as (linear) [Support Vector Machines](#) and [Logistic Regression](#). Even though SGD has been around in the machine learning community for a long time, it has received a considerable amount of attention just recently in the context of large-scale learning.

SGD has been successfully applied to large-scale and sparse machine learning problems often encountered in text classification and natural language processing. Given that the data is sparse, the classifiers in this module easily scale to problems with more than  $10^4$  training examples and more than  $10^4$  features.

The advantages of Stochastic Gradient Descent are:

- Efficiency.
- Ease of implementation (lots of opportunities for code tuning).

The disadvantages of Stochastic Gradient Descent include:

- SGD requires a number of hyperparameters such as the regularization parameter and the number of iterations.
- SGD is sensitive to feature scaling.

### 1.5. Stochastic Gradient Descent

**1.5.1. Classification**

**1.5.2. Regression**

**1.5.3. Stochastic Gradient Descent for sparse data**

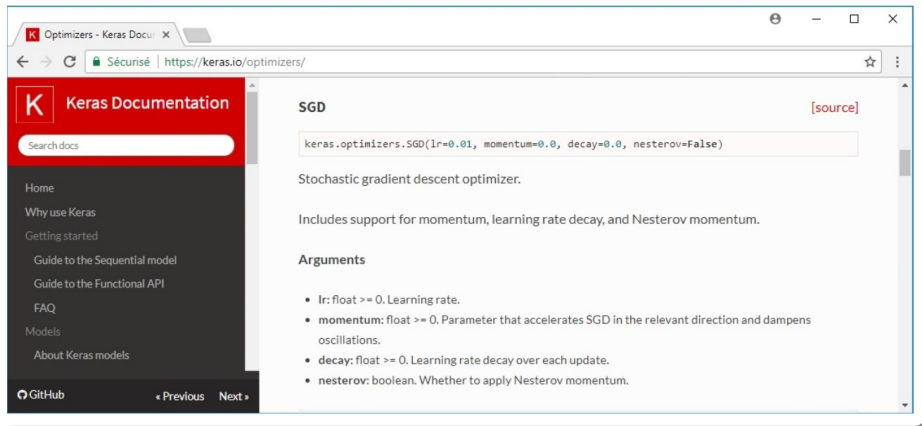
**1.5.4. Complexity**

**1.5.5. Tips on Practical Use**

**1.5.6. Mathematical formulation**

- 1.5.6.1. SGD

**1.5.7. Implementation details**



The screenshot shows a web browser window displaying the Keras Documentation page for the SGD optimizer. The browser's address bar shows the URL `https://keras.io/optimizers/`. The page has a dark sidebar on the left with the Keras logo and a search bar. The main content area is white and contains the following information:

- SGD** (Stochastic gradient descent optimizer) with a [\[source\]](#) link.
- A code snippet: `keras.optimizers.SGD(lr=0.01, momentum=0.0, decay=0.0, nesterov=False)`
- Description: "Stochastic gradient descent optimizer."
- Features: "Includes support for momentum, learning rate decay, and Nesterov momentum."
- Arguments** section with a bulleted list:
  - lr**: float  $\geq 0$ . Learning rate.
  - momentum**: float  $\geq 0$ . Parameter that accelerates SGD in the relevant direction and dampens oscillations.
  - decay**: float  $\geq 0$ . Learning rate decay over each update.
  - nesterov**: boolean. Whether to apply Nesterov momentum.



CRAN - Package gradDescent

← → ↻ Sécursé | <https://cran.r-project.org/web/packages/gradDescent/index.html> ☆ ⋮

### gradDescent: Gradient Descent for Regression Tasks

An implementation of various learning algorithms based on Gradient Descent for dealing with regression tasks. The variants of gradient descent algorithm are : Mini-Batch Gradient Descent (MBGD), which is an optimization to use training data partially to reduce the computation load. Stochastic Gradient Descent (SGD), which is an optimization to use a random data in learning to reduce the computation load drastically. Stochastic Average Gradient (SAG), which is a SGD-based algorithm to minimize stochastic step to average. Momentum Gradient Descent (MGD), which is an optimization to speed-up gradient descent learning. Accelerated Gradient Descent (AGD), which is an optimization to accelerate gradient descent learning. Adagrad, which is a gradient-descent-based algorithm that accumulate previous cost to do adaptive learning. Adadelata, which is a gradient-descent-based algorithm that use hessian approximation to do adaptive learning. RMSprop, which is a gradient-descent-based algorithm that combine Adagrad and Adadelata adaptive learning ability. Adam, which is a gradient-descent-based algorithm that mean and variance moment to do adaptive learning. Stochastic Variance Reduce Gradient (SVRG), which is an optimization SGD-based algorithm to accelerates the process toward converging by reducing the gradient. Semi Stochastic Gradient Descent (SSGD), which is a SGD-based algorithm that combine GD and SGD to accelerates the process toward converging by choosing one of the gradients at a time. Stochastic Recursive Gradient Algorithm (SARAH), which is an optimization algorithm similarly SVRG to accelerates the process toward converging by accumulated stochastic information. Stochastic Recursive Gradient Algorithm+ (SARAHPlus), which is a SARAH practical variant algorithm to accelerates the process toward converging provides a possibility of earlier termination.

Version: 3.0  
 Published: 2018-01-25  
 Author: Galih Praja Wijaya, Dendi Handian, Imam Fachmi Nasrulloh, Lala Septem Riza, Rani Megasari, Enjun Junaeti  
 Maintainer: Lala Septem Riza <lala.s.riza at upi.edu>  
 License: [GPL-2](#) | [GPL-3](#) | file [LICENSE](#) [expanded from: GPL (≥ 2) | file LICENSE]  
 URL: <https://github.com/drizzersilverberg/gradDescentR>  
 NeedsCompilation: no  
 In views: [MachineLearning](#)  
 CRAN checks: [gradDescent results](#)  
 Downloads:

Reference manual: [gradDescent.pdf](#)  
 Package source: [gradDescent\\_3.0.tar.gz](#)  
 Windows binaries: r-prerelease: [gradDescent\\_3.0.zip](#), r-release: [gradDescent\\_3.0.zip](#), r-older: [gradDescent\\_3.0.zip](#)  
 OS X binaries: r-prerelease: [gradDescent\\_3.0.tgz](#), r-release: [gradDescent\\_3.0.tgz](#)  
 Old sources: [gradDescent archive](#)

Linking:

Please use the canonical form <https://CRAN.R-project.org/package=gradDescent> to link to this page.

## Importance de la Descente de Gradient en apprentissage supervisé?

- Approches et surtout implémentations classiques des méthodes de data mining impuissantes par rapport aux très grandes volumétries.
- L'algorithme du gradient / gradient stochastique permet de les appréhender sans nécessiter de ressources machines prohibitives.
- Possibilité de parallélisation des algorithmes.

## Cependant Attention!

- Grand nombre de paramètres pas toujours faciles à appréhender, qui influent sur le comportement de l'algorithme.
- Toujours ramener les variables sur la même échelle (normalisation, standardisation) pour éviter que les disparités faussent le déroulement de l'optimisation.