

Cours 2 - La régression linéaire et polynomiale

Neila Mezghani

5 janvier 2022

Plan du cours

- 1 La régression linéaire
 - Le modèle de régression linéaire
 - La fonction coût
 - Entraînement du modèle de régression linéaire
 - Méthode analytique
 - Descente de gradient
- 2 Les modèles de régression polynomiaux
 - Principe de régression polynomiale
 - Régression polynomiale de haut degré
- 3 Courbe d'apprentissage

La régression linéaire

Plan du cours

- 1 La régression linéaire
 - Le modèle de régression linéaire
 - La fonction coût
 - Entraînement du modèle de régression linéaire
 - Méthode analytique
 - Descente de gradient
- 2 Les modèles de régression polynomiaux
 - Principe de régression polynomiale
 - Régression polynomiale de haut degré
- 3 Courbe d'apprentissage

Le modèle de régression linéaire (1/5)

Un modèle de régression linéaire effectue une prédiction en calculant simplement une somme pondérée des variables d'entrée tout en y ajoutant un terme constant (intercept) :

$$\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_n x_n \quad (1)$$

avec

- \hat{y} est la valeur prédite et x_i est la valeur de la i ème variable
- n est le nombre de variables
- θ_j est le j ème paramètre du modèle (terme constant θ_0 et coefficients de pondération des variables $\theta_1, \theta_2, \dots, \theta_n$).

Le modèle de régression linéaire (2/5)

L'équation (1) peut s'écrire de manière générale :

$$\hat{y} = h_{\theta}(\mathbf{x}) = \theta \cdot \mathbf{x} \quad (2)$$

Dans cette équation :

- θ est le vecteur des paramètres du modèle. Il regroupe à la fois θ_0 et les coefficients de pondération θ_1 à θ_n des variables.
- \mathbf{x} est le vecteur des valeurs d'une observation, contenant les valeurs x_0 à x_n , **où x_0 est toujours égal à 1.**
- $\theta \cdot \mathbf{x}$ est le produit scalaire de θ et \mathbf{x} .
- h_{θ} est la fonction hypothèse, utilisant les paramètres du modèle θ .

Le modèle de régression linéaire (3/5)

Souvent, on réalise plusieurs prédictions simultanément \Rightarrow on regroupe dans une même matrice \mathbf{X} toutes les observations pour lesquelles on souhaite faire des prédictions

$$\mathbf{X} = \begin{pmatrix} \mathbf{x}^{(1)} \\ \mathbf{x}^{(2)} \\ \vdots \\ \mathbf{x}^{(m)} \end{pmatrix} = \begin{pmatrix} x_1^{(1)} & x_2^{(1)} & \cdots & x_n^{(1)} \\ x_1^{(2)} & x_2^{(2)} & \cdots & x_n^{(2)} \\ \vdots & \ddots & \cdots & \vdots \\ x_1^{(m)} & x_2^{(m)} & \cdots & x_n^{(m)} \end{pmatrix} \quad (3)$$

m étant le nombre d'observations et n le nombre de variable.


$$\mathbf{x}^{(1)} = (x_1^{(1)}, x_2^{(1)}, \cdots, x_n^{(1)})$$

Le modèle de régression linéaire (4/5)

Pour réaliser simultanément une prédiction pour toutes les observations, on peut alors simplement utiliser l'équation suivante :

$$\hat{\mathbf{y}} = \mathbf{X} \boldsymbol{\theta} \quad (4)$$

avec $\hat{\mathbf{y}}$ est le vecteur des prédictions. Son i ème élément correspond à la prédiction du modèle pour la i ème observation.

 Dans l'Éq (2) : $\hat{y} = \boldsymbol{\theta} \cdot \mathbf{x} = \mathbf{x} \cdot \boldsymbol{\theta}$ parce que nous avons des produits scalaires.

Par contre dans l'Éq (4) $\hat{\mathbf{y}} = \mathbf{X} \boldsymbol{\theta}$ nous avons un produit matriciel \implies l'ordre est important.

(La méthode `dot()` permet de calculer les produits matriciels)

Le modèle de régression linéaire (5/5)

- L'application de Eq 4 à l'ensemble des données permet d'écrire :

$$\begin{cases} \hat{y}_1 = \theta_0 + \theta_1 x_1^{(1)} + \theta_2 x_2^{(1)} + \dots + \theta_n x_n^{(1)} \\ \hat{y}_2 = \theta_0 + \theta_1 x_1^{(2)} + \theta_2 x_2^{(2)} + \dots + \theta_n x_n^{(2)} \\ \dots \\ \hat{y}_m = \theta_0 + \theta_1 x_1^{(m)} + \theta_2 x_2^{(m)} + \dots + \theta_n x_n^{(m)} \end{cases} \quad (5)$$

- Entraîner un modèle de régression consiste à définir ses paramètres de telle sorte que le modèle s'ajuste au mieux aux données d'entraînement \implies Besoin d'une **mesure** de performance pour évaluer si le modèle s'ajuste bien.

Plan du cours

- 1 La régression linéaire
 - Le modèle de régression linéaire
 - **La fonction coût**
 - Entraînement du modèle de régression linéaire
 - Méthode analytique
 - Descente de gradient
- 2 Les modèles de régression polynomiaux
 - Principe de régression polynomiale
 - Régression polynomiale de haut degré
- 3 Courbe d'apprentissage

La fonction coût : Erreur quadratique moyenne (1/3)

- On utilise généralement une mesure de l'erreur commise par le modèle sur l'ensemble des données d'entraînement.
- On évalue, pour chaque individu i , le **résidu** ou **l'erreur** ϵ_i qui est la différence entre une valeur observée de y_i et la valeur \hat{y}_i prédite par l'équation de régression \implies c'est ce qu'on appelle la fonction coût.

$$\epsilon_i = y_i - \hat{y}_i \quad (6)$$

- La fonction de coût la plus courante pour un modèle de régression est la racine carrée de l'erreur quadratique moyenne (en anglais, root mean square error ou RMSE)

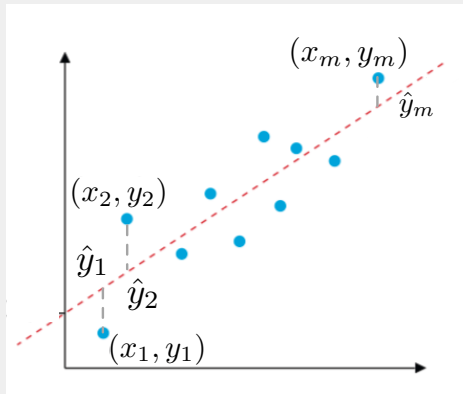
La fonction coût : Erreur quadratique moyenne (2/3)

Exemple : si nous avons un ensemble de m observations, l'équation de régression est donnée par :

$$\hat{\mathbf{y}} = \theta_0 + \theta_1 \mathbf{x}$$

La RMSE est donnée par :

$$RMSE = \sqrt{\frac{1}{m} \sum_{i=1}^m (\hat{y}_i - y_i)^2}$$



La fonction coût : Erreur quadratique moyenne (3/3)

- Plus généralement, la RMSE s'écrit :

$$RMSE(X, h_{\theta}) = \sqrt{\frac{1}{m} \sum_{i=1}^m (h_{\theta}(\mathbf{x}^{(i)}) - y_i)^2} \quad (7)$$

m étant le nombre d'observations.

- Entraîner un modèle de régression linéaire \implies trouver le vecteur θ qui minimise la RMSE.
- En pratique, il est plus simple et plus rapide de minimiser l'erreur quadratique moyenne (MSE) que la RMSE.

Plan du cours

- 1 La régression linéaire
 - Le modèle de régression linéaire
 - La fonction coût
 - Entraînement du modèle de régression linéaire
 - Méthode analytique
 - Descente de gradient
- 2 Les modèles de régression polynomiaux
 - Principe de régression polynomiale
 - Régression polynomiale de haut degré
- 3 Courbe d'apprentissage

Entrainement du modèle de régression linéaire

- L'entraînement du modèle de régression linéaire a pour objectif de déterminer la valeur de θ qui minimise la fonction coût MSE.
- Il existe deux façons très différentes d'entraînement :

1- Une méthode analytique

- Calcule directement les valeurs des paramètres du modèle donnant les meilleurs résultats

2- Une approche d'optimisation itérative

- Basée sur la descente de gradient

1 - Méthode analytique

- Afin de déterminer la valeur θ qui minimise la fonction coût, on utilise une méthode analytique (qui se base sur une formule mathématique) :

$$\hat{\theta} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y} \quad (8)$$

avec $\hat{\theta}$ la valeur de θ qui minimise la fonction coût.

$^{-1}$ indique l'inverse de la matrice $(\mathbf{X}^T \mathbf{X})$ (On peut utiliser **inv**)

\mathbf{y} est le vecteur des valeurs à prédire.

- L'équation (8) porte le nom d'Équation normale.

Exemple : Entraînement basé sur l'équation normale (1/4)

La détermination de $\hat{\theta}$ peut être obtenue par l'équation normale :

$$\hat{\theta} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$$

```
# Ajouter x0 = 1 a chaque observation
```

```
X_b = np.c_[np.ones((100, 1)), X]
```

```
theta_hat= np.linalg.inv(X_b.T.dot(X_b)).dot(X_b.T).dot(y)
```

On obtient à la sortie la valeur de $\hat{\theta}$.

Exemple : Entraînement basé sur l'équation normale (2/4)

On peut également procéder à un entraînement en utilisant la librairie Scikit-Learn

```
from sklearn.linear_model import LinearRegression  
lin_reg = LinearRegression()  
lin_reg.fit(X, y)  
lin_reg.intercept_, lin_reg.coef_
```

$\theta_0 = \text{lin_reg.intercept_}$

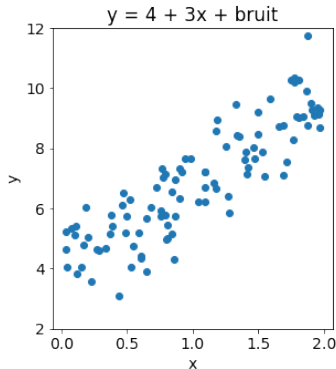
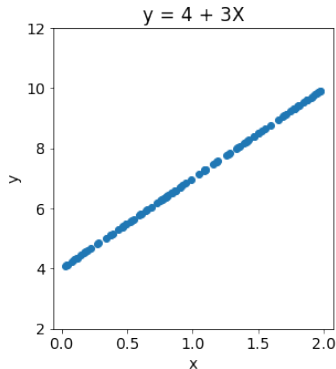
$\theta_1 = \text{lin_reg.coef_}$

La classe `LinearRegression` de Scikit-Learn utilise une décomposition en valeurs singulières pour le calcul matriciel.

Exemple : Entraînement basé sur l'équation normale (3/4)

On considère une variable aléatoire \mathbf{x} de dimension 100 et une variable \mathbf{y} tel que : $y = 4 + 3x$

On ajoute un bruit gaussien à la variable \mathbf{y}

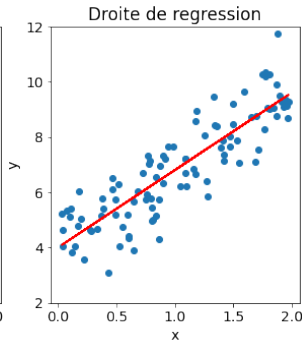
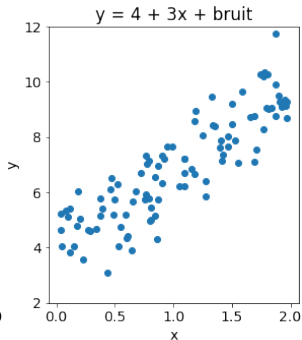
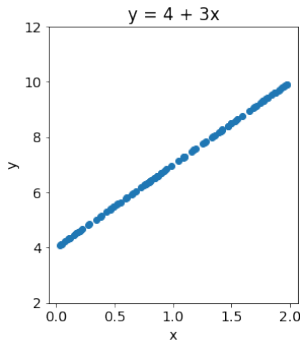


Exemple : Entraînement basé sur l'équation normale (4/4)

L'exécution de l'équation normale ou de `LinearRegression()` donne

$$\theta_0 = 4.0031 \text{ et } \theta_1 = 2.8026$$

L'équation de régression obtenue est donc : $\hat{y} = 4.0031 + 2.8026x$
comparée à l'équation réelle : $y = 4 + 3x$



Limites de l'entraînement basé sur l'équation normale

- L'équation normale calcule l'inverse de $\mathbf{X}^T \mathbf{X}$, qui est une matrice $(n + 1) \times (n + 1)$ (où n est le nombre de variables) \Rightarrow La complexité algorithmique d'une inversion de matrice est de l'ordre de $O(n^3)$, selon l'algorithme d'inversion utilisé.
- Si on double le nombre de variables, le temps de calcul est multiplié par un facteur de $2^3 = 8$
 \Rightarrow Besoin de méthode d'entraînement mieux adaptée au cas où il y a beaucoup de variables ou d'observations \Rightarrow La descente de gradient.

2- La descente de gradient

1- Une méthode analytique

- Calcule directement les valeurs des paramètres du modèle donnant les meilleurs résultats

2- Une approche d'optimisation itérative

- Basée sur la descente de gradient

2- La descente de gradient : principe (1/4)

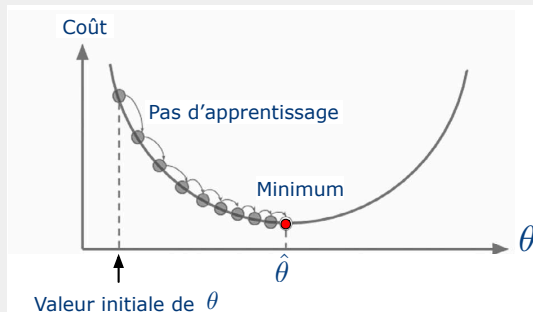
- La descente de gradient est un algorithme d'optimisation très général et qui est capable de trouver des solutions optimales à un grand nombre de problèmes.
- Cet algorithme d'optimisation est utilisé pour calculer les paramètres (coefficients et biais) pour des algorithmes comme la régression linéaire, la régression logistique, les réseaux de neurones, etc.
- L'idée générale de la descente de gradient est d'ajuster, pendant la phase d'apprentissage, petit à petit les paramètres dans le but de minimiser la fonction de coût.

2- La descente de gradient : principe (2/4)

La taux d'apprentissage est un élément important dans l'algorithme de descente de gradient.

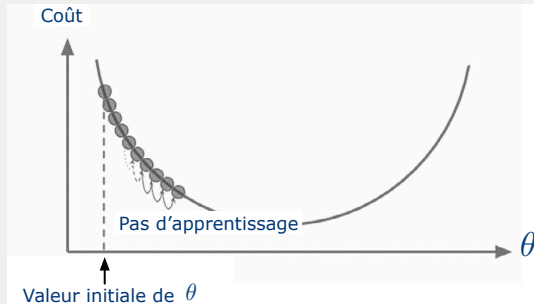
Ce taux correspond à la dimension des pas que l'on détermine par l'intermédiaire de l'hyperparamètre `learning_rate`.

$\hat{\theta}$ est la valeur minimale de la fonction coût.



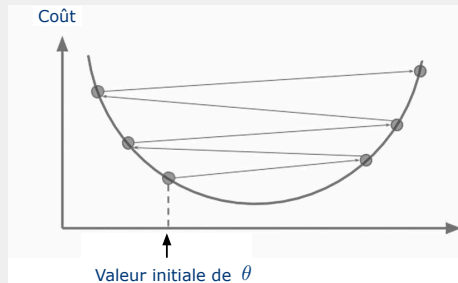
2- La descente de gradient : principe (3/4)

Si le taux d'apprentissage est trop petit, l'algorithme devra effectuer un grand nombre d'itérations pour converger et prendra beaucoup de temps.



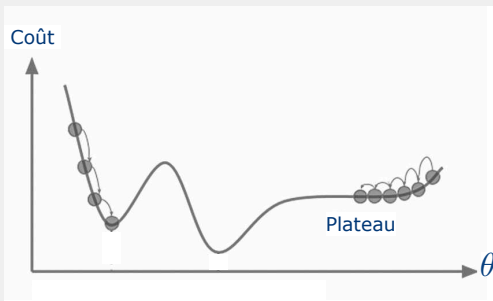
2- La descente de gradient : principe (4/4)

Si le taux d'apprentissage est trop élevé, on risque de dépasser le point le plus bas et de se retrouver de l'autre côté de la valeur optimale de $\theta \implies$ L'algorithme peut diverger, ce qui empêcherait de trouver une bonne solution



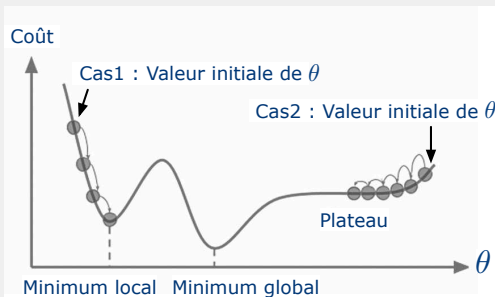
2- La descente de gradient : Limites (1/2)

- Les fonctions de coût n'ont pas toujours une forme convexe (la forme d'une jolie cuvette régulière)
- On retrouve des trous, des crêtes, des plateaux \Rightarrow ce qui complique la convergence vers le minimum.



2- La descente de gradient : Limites (2/2)

- **Cas 1** : La valeur initiale de θ entraîne la convergence de l'algorithme vers un minimum local \neq du minimum global.
- **Cas 2** : La valeur initiale de θ est très loin \Rightarrow l'algorithme prendra longtemps pour traverser le plateau afin d'atteindre le minimum global



2- La descente de gradient : La fonction de coût MSE (1/2)

La fonction de coût MSE du modèle de régression linéaire est :

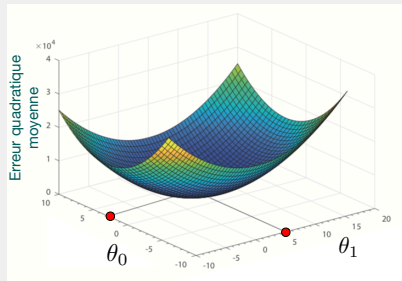
(1) une fonction convexe

(2) une fonction continue dont la pente ne varie jamais abruptement

⇒ il est garanti que la descente de gradient s'approche suffisamment du minimum global

2- La descente de gradient : La fonction de coût MSE (2/2)

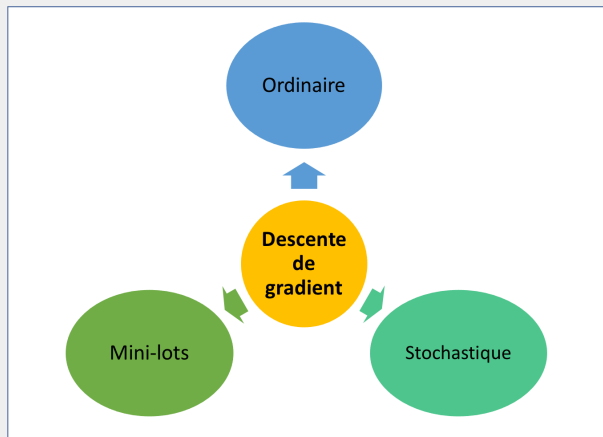
Si nous avons deux variables x_0 et x_1 , la fonction coût dépend des paramètres θ_0 et $\theta_1 \implies$ Elle a l'allure d'une fonction convexe dont les paramètres sont θ_0 et θ_1 (Les points rouges correspondent aux valeurs optimales)



2- La descente de gradient : Implémentation (1/2)

- Rappelons que, pour entraîner un modèle de régression linéaire, il faut trouver le vecteur θ qui minimise la RMSE.
- Pour implémenter une descente de gradient, on doit calculer le gradient de la fonction de coût MSE par rapport à chaque paramètre θ_j du modèle \implies rechercher une combinaison de paramètres du modèle minimisant une fonction de coût (sur le jeu d'entraînement).

2- La descente de gradient : Implémentation (2/2)



(a) La descente de gradient ordinaire (1/4)

- Pour implémenter une descente de gradient, on doit calculer le gradient de la fonction de coût MSE par rapport à chaque paramètre θ_j du modèle
- Déterminer quelle est la modification de la fonction de coût lorsque qu'on modifie θ_j , c'est ce qu'on appelle une dérivée partielle.

(a) La descente de gradient ordinaire (2/4)

On a :

$$MSE(\boldsymbol{\theta}) = \frac{1}{m} \sum_{i=1}^m (h_{\boldsymbol{\theta}}(\mathbf{x}_i) - y_i)^2$$

La dérivée partielle de la fonction de coût MSE par rapport à θ_j est donnée par :

$$\frac{\partial}{\partial \theta_j} MSE(\boldsymbol{\theta}) = \frac{2}{m} \sum_{i=1}^m (\boldsymbol{\theta}^T \mathbf{x}^{(i)} - y^{(i)}) x_j^{(i)} \quad (9)$$

(a) La descente de gradient ordinaire (3/4)

Au lieu de calculer individuellement chaque dérivée partielle, on peut utiliser la formulation vectorielle suivante pour les calculer toutes ensemble. Le vecteur gradient, noté $\nabla_{\theta}MSE(\theta)$, est composé de toutes les dérivées partielles de la fonction de coût (une pour chaque paramètre du modèle) :

$$\nabla_{\theta}MSE(\theta) = \begin{pmatrix} \frac{\partial}{\partial \theta_1}MSE(\theta) \\ \frac{\partial}{\partial \theta_2}MSE(\theta) \\ \vdots \\ \frac{\partial}{\partial \theta_n}MSE(\theta) \end{pmatrix} = \frac{2}{m}\mathbf{X}^T(X\theta - \mathbf{y}) \quad (10)$$

(a) La descente de gradient ordinaire (4/4)

- Une fois le gradient $\nabla_{\theta}MSE(\theta)$ calculé, on met à jour la valeur de θ :

$$\theta^{(\text{Étape suivante})} = \theta^{(\text{Étape actuelle})} - \eta \nabla_{\theta}MSE(\theta) \quad (11)$$

η étant le taux d'apprentissage = facteur de multiplication du vecteur gradient pour fixer le pas de progression.

Exemple : Descente de gradient ordinaire (1/2)

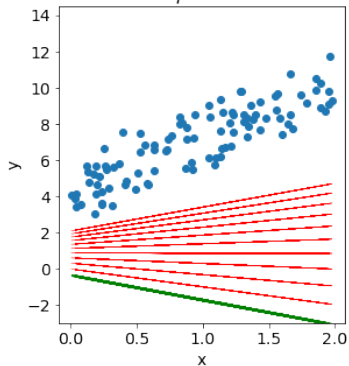
La descente de gradient peut être implémentée via les lignes de codes suivants :

```
m = 100
# Initialisation des hyperparametres
eta = 0.1
N = 1000
# Initialisations des parametres
theta = np.random.randn(2,1)
#Implementation de l'algorithme de descente de gradient
for iteration in range(N):
    gradients = 2/m * X_b.T.dot(X_b.dot(theta) - y)
    theta = theta - eta * gradients
```

Exemple : Descente de gradient ordinaire (2/2)

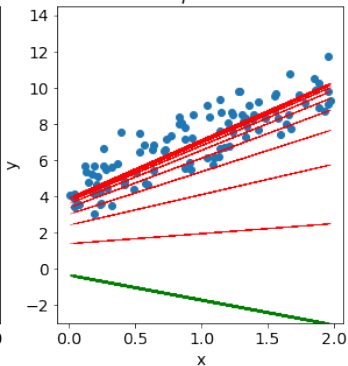
η est trop faible : l'algorithme aboutira au bout du compte à la solution mais cela prendra très longtemps.

$\eta = 0.02$



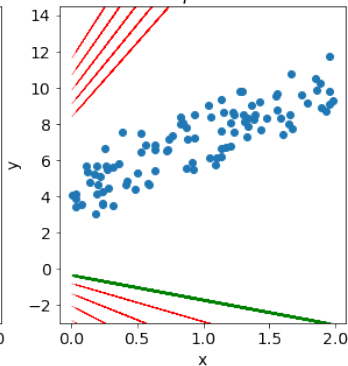
η est assez bon : l'algorithme a convergé en quelques itérations

$\eta = 0.1$



η est trop grand : l'algorithme diverge et s'éloignant de plus en plus de la solution à chaque étape.

$\eta = 0.5$



Comment choisir la pas d'apprentissage η ?

Comment choisir le taux d'apprentissage η ?

- Si l'algorithme met trop de temps à converger, il faudra augmenter la valeur de η .
- Si l'algorithme diverge, il faudra diminuer la valeur de η pour avoir de moins grand pas.

Comment choisir le nombre d'itérations N ?

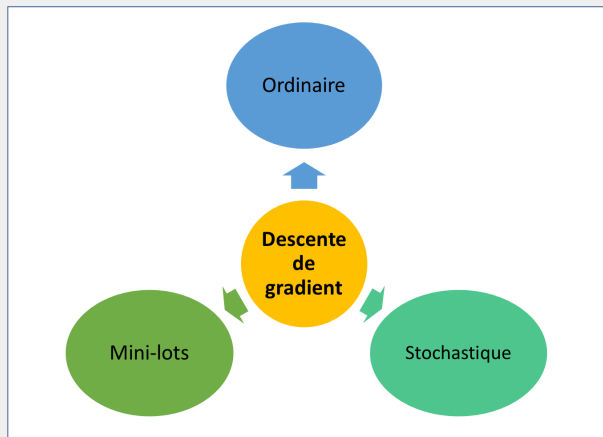
Comment choisir le nombre d'itérations N ?

- Si N est trop faible \implies On sera très loin de la solution optimale lorsque l'algorithme s'arrêtera.
- Si N est trop élevé \implies On perd du temps alors que les paramètres du modèle n'évolueront plus.
- Une solution simple consiste à choisir un très grand nombre d'itérations, mais à interrompre l'algorithme lorsque le vecteur de gradient devient très petit, c'est-à-dire quand sa norme devient inférieure à un très petit nombre tol appelé tolérance

Limites de la descente de gradient ordinaire

- La descente de gradient ordinaire = La descente de gradient groupée = *batch gradient descent* = *full gradient descent*
- L'équation des dérivées partielles de la fonction de coût (Eq. 10), implique des calculs sur l'ensemble des données d'entraînement \mathbf{X} à chaque étape de la descente de gradient \Rightarrow Algorithme lent sur des ensemble de données d'entraînement très volumineux \Rightarrow Besoin d'une approche d'implémentation plus rapide.

2- La descente de gradient : Implémentation



(b) Descente de gradient stochastique (1/3)

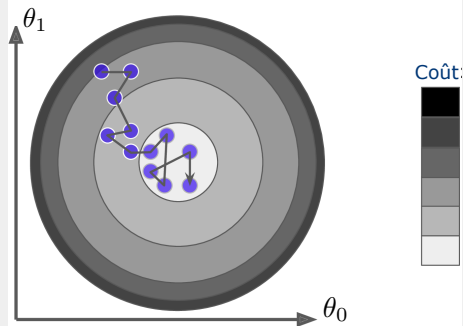
- Le principal problème de la descente de gradient ordinaire, c'est qu'elle utilise à chaque étape la totalité de l'ensemble d'entraînement pour calculer les gradients, ce qui la rend très lente lorsque sa taille est grande.
- La descente de gradient stochastique choisit à chaque étape une observation prise au hasard dans l'ensemble d'entraînement et calcule les gradients en se basant uniquement sur cette seule observation.

(b) Descente de gradient stochastique (2/3)

- Par contre, étant donné sa nature stochastique (c'est-à-dire aléatoire), cet algorithme est beaucoup moins régulier qu'une descente de gradient ordinaire.
- Au lieu de décroître doucement jusqu'à atteindre le minimum, la fonction de coût va effectuer des sauts vers le haut et vers le bas et ne décroîtra qu'en moyenne.
- À la fin, elle arrivera très près du minimum, mais continuera à effectuer des sauts aux alentours sans s'arrêter.
- Pour effectuer une régression linéaire par descente de gradient stochastique avec Scikit-Learn, on utilise **SGDRegressor**.

(b) Descente de gradient stochastique (3/3)

La fonction de coût va effectuer des sauts vers le haut et vers le bas
⇒ lorsque l'algorithme est stoppé, les valeurs finales des paramètres sont bonnes, mais pas nécessairement les plus optimales.



Exemple : La descente de gradient stochastique

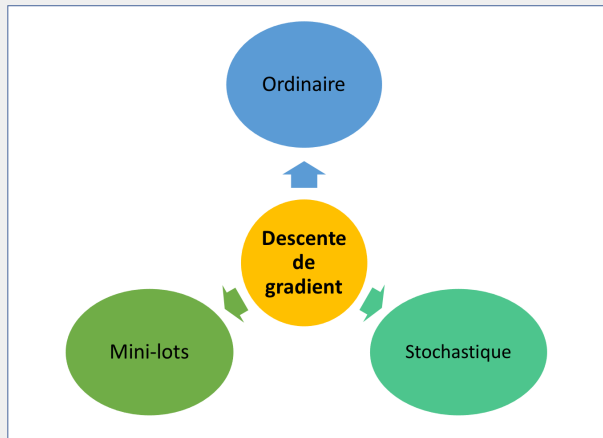
```
from sklearn.linear_model import SGDRegressor  
sgd_reg = SGDRegressor(max_iter=1000, tol=1e-3, penalty=None, eta0=0.1)  
sgd_reg.fit(X, y.ravel())
```

L'exécution de `SGDRegressor()` permet d'identifier les valeurs de $\theta_0 = 4.2436$ et $\theta_1 = 2.8250$

L'équation de régression obtenue est donc : $\hat{y} = 4.2436 + 2.8250x$
comparée à l'équation réelle : $y = 4 + 3x$

(Rappel : L'équation de régression obtenue la méthode analytique est : $\hat{y} = 4.0031 + 2.8026x$)

2- La descente de gradient : Implémentation



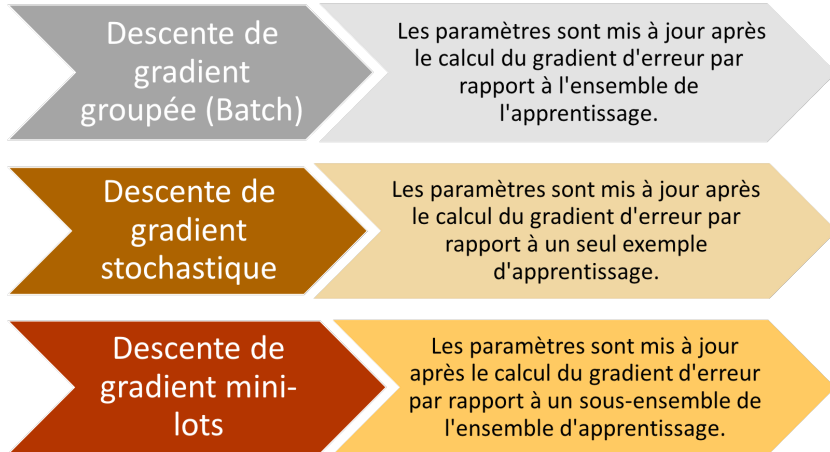
(c) Descente de gradient par mini-lots (1/2)

- La descente de gradient par mini-lots (en anglais, mini-batch gradient descent) vient remédier aux limites précédentes.
- Calcule le gradient sur de petits sous-ensembles d'observations sélectionnées aléatoirement qu'on appelle mini-lots.
- La progression de l'algorithme dans l'espace des paramètres est moins désordonnée que dans le cas de la descente de gradient stochastique

(c) Descente de gradient par mini-lots (2/2)

- La descente de gradient par mini-lots aboutit à une solution un peu plus proche du minimum qu'une descente de gradient stochastique.
- Par contre elle aura plus de mal à sortir d'un minimum local (mais le problème ne se pose dans le cas de la régression linéaire).

En résumé...



Les modèles de régression polynomiaux

Plan du cours

- 1 La régression linéaire
 - Le modèle de régression linéaire
 - La fonction coût
 - Entraînement du modèle de régression linéaire
 - Méthode analytique
 - Descente de gradient
- 2 Les modèles de régression polynomiaux
 - Principe de régression polynomiale
 - Régression polynomiale de haut degré
- 3 Courbe d'apprentissage

La régression polynomiale

- La majorité des phénomènes physique régis par des équations non-linéaires. Que faire pour les modéliser ?
- On peut utiliser un modèle linéaire pour ajuster des données non linéaires.
- L'un des moyens d'y parvenir consiste à ajouter les puissances de chacune des variables comme nouvelles variables, puis d'entraîner un modèle linéaire sur ce nouvel ensemble de données \implies régression polynomiale.

Du modèle linéaire au modèle polynomial ...

Si nous disposons d'une seule caractéristique, nous pouvons écrire pour une observation x :

- Dans le cas d'un modèle linéaire :

$$y = \theta_1 x + \theta_0$$

- Dans le cas d'un modèle polynomial de second degré :

$$y = \theta_2 x^2 + \theta_1 x + \theta_0$$

Du modèle linéaire au modèle polynomial ...

On peut donc généraliser pour m observations et structurer les données dans une matrice \mathbf{X}

$$\mathbf{X} = \begin{pmatrix} x_1^2 & x_1 & 1 \\ x_2^2 & x_2 & 1 \\ \vdots & \dots & \vdots \\ x_m^2 & x_m & 1 \end{pmatrix} \quad (12)$$

et aussi garder :

$$\hat{\mathbf{y}} = \mathbf{X} \boldsymbol{\theta}$$

avec $\hat{\mathbf{y}}$ le vecteur des prédictions et \mathbf{X} la matrice qui regroupe tous les vecteurs d'observations.

Du modèle linéaire au modèle polynomial ...

et donc déduire ces équations

$$\begin{cases} \hat{y}_1 = \theta_2 x_1^2 + \theta_1 x_1 + \theta_0 \\ \hat{y}_2 = \theta_2 x_2^2 + \theta_1 x_2 + \theta_0 \\ \dots \\ \hat{y}_m = \theta_1 x_m^2 + \theta_2 x_m + \theta_0 \end{cases} \quad (13)$$

Exemple : Modélisation polynomiale (1/4)

On désire générer un ensemble de données non-linéaires.

Soit la fonction polynomiale de second degré : $y = 0.5x^2 + x + 2$ à laquelle on ajoute un bruit gaussien ϵ

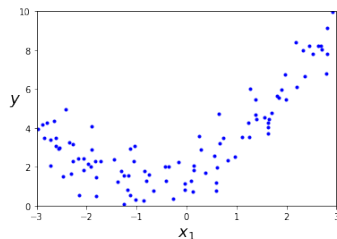
On génère les variables **X** et **y** moyennant ces lignes de codes :

```
#Generation d'une fonction polynomiale
```

```
m = 100
```

```
X = 6 * np.random.rand(m, 1) - 3
```

```
y = 0.5 * X**2 + X + 2 + np.random.randn(m,1)
```



Exemple : Modélisation polynomiale (2/4)

On peut utiliser la classe `PolynomialFeatures` de Scikit-Learn pour transformer nos données d'apprentissage, en ajoutant les carrés (polynômes du second degré) de la variable X (`degree=2`)

```
from sklearn.preprocessing import PolynomialFeatures
poly_features = PolynomialFeatures(degree=2, include_bias=False)
X_poly = poly_features.fit_transform(X)
```

L'affichage de la première observation donne :

`X[0]=[-0.75275929]`

`X_poly[0]=[[[-0.75275929, 0.56664654]]]` (la valeur de x et x^2)

Exemple : Modélisation polynomiale (3/4)

On développe par la suite un modèle de régression linéaire à l'ensemble des données

```
from sklearn.linear_model import LinearRegression  
lin_reg = LinearRegression()  
lin_reg.fit(X_poly, y)
```

Les valeurs obtenues de θ sont :

$$\theta_0 = 1.781$$

$$\theta_1 = 0.933$$

$$\theta_2 = 0.564$$

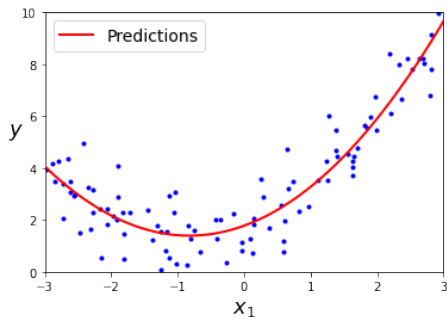
Exemple : Modélisation polynomiale (4/4)

A partir des valeurs $\theta = (\theta_0, \theta_1, \theta_2)$,
on peut donc écrire l'éq. de régres-
sion polynomiale :

$$\hat{y} = 0.564x^2 + 0.933x + 1.781$$

La fonction polynomiale de second
degré étant :

$$y = 0.5x^2 + x + 2$$

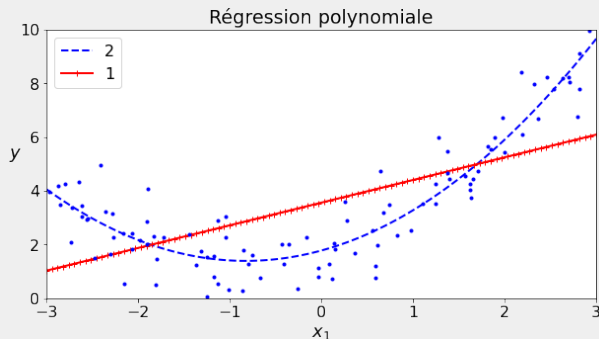


Plan du cours

- 1 La régression linéaire
 - Le modèle de régression linéaire
 - La fonction coût
 - Entraînement du modèle de régression linéaire
 - Méthode analytique
 - Descente de gradient
- 2 Les modèles de régression polynomiaux
 - Principe de régression polynomiale
 - Régression polynomiale de haut degré
- 3 Courbe d'apprentissage

Régression polynomiale de haut degré (1/3)

- Le passage d'une régression polynomiale de degré 1 (modèle linéaire) à une régression polynomiale de degré 2 (modèle quadratique) a permis un meilleur ajustement aux données d'entraînement.

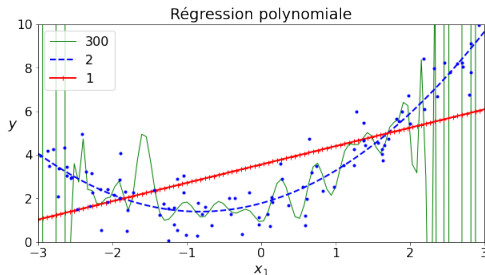


Régression polynomiale de haut degré (2/3)

- On peut donc penser au fait qu'une régression polynomiale de haut degré, permettra un meilleur ajustement aux données d'entraînement.
- C'est fort probable mais le modèle polynomial de haut degré peut surajuster considérablement les données d'entraînement \Rightarrow On obtient un surapprentissage

Exemple : Régression polynomiale de haut degré

- (1) : Modèle linéaire sous-ajuste les données d'entraînement.
- (2) : Modèle quadratique généralise le mieux.
- (3) : Modèle polynomial de degré 300 : surajuste considérablement les données d'entraînement.



Régression polynomiale de haut degré (3/3)

Comment peut-on éviter un sous- et un sur-apprentissage ?

- Entraîner plusieurs fois le modèle avec des degrés polynomiaux différents, jusqu'à trouver le degré qui produit le meilleur modèle, évalué sur les données de validation.
- Examiner les courbes d'apprentissage.

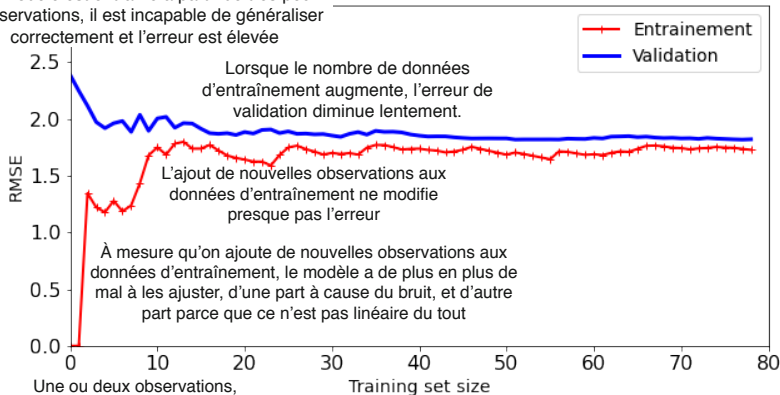
Courbe d'apprentissage

Courbe d'apprentissage

- Courbe d'apprentissage = diagramme représentant les résultats obtenus par le modèle sur l'ensemble des données d'entraînement et l'ensemble des données de validation en fonction de la taille des données d'entraînement ou de l'itération d'entraînement.
- Pour générer ces graphiques, entraînez le modèle plusieurs fois sur des sous-ensembles de différentes tailles de l'ensemble d'entraînement.

Exemple : Courbe d'apprentissage d'un simple modèle linéaire - sous-ajustement

Le modèle est entraîné à partir de très peu d'observations, il est incapable de généraliser correctement et l'erreur est élevée



Une ou deux observations,
le modèle peut les ajuster parfaitement

Exemple : Courbe d'apprentissage pour le modèle polynomial de degré 10 - surajustement

