

L'année dernière, nous avons été introduit au machine learning, plus précisément les catégories d'apprentissage supervisé avec la linear et logistic regression, ainsi que l'apprentissage non supervisé avec la méthode des K-nearest neighbors (KNN).

Nous avons aussi découvert la méthode de descente de Gradient pour minimiser une fonction de coût dans le but de trouver de bon paramètres pour un modèle.

Ce tp m'a permis de découvrir de manière détaillée cet algorithme, même si le vocabulaire était différent de notre cours de l'an dernier.

Ci-dessous l'exercice 1.

tpml1

September 14, 2023

```
[92]: import pandas as pd
import numpy as np

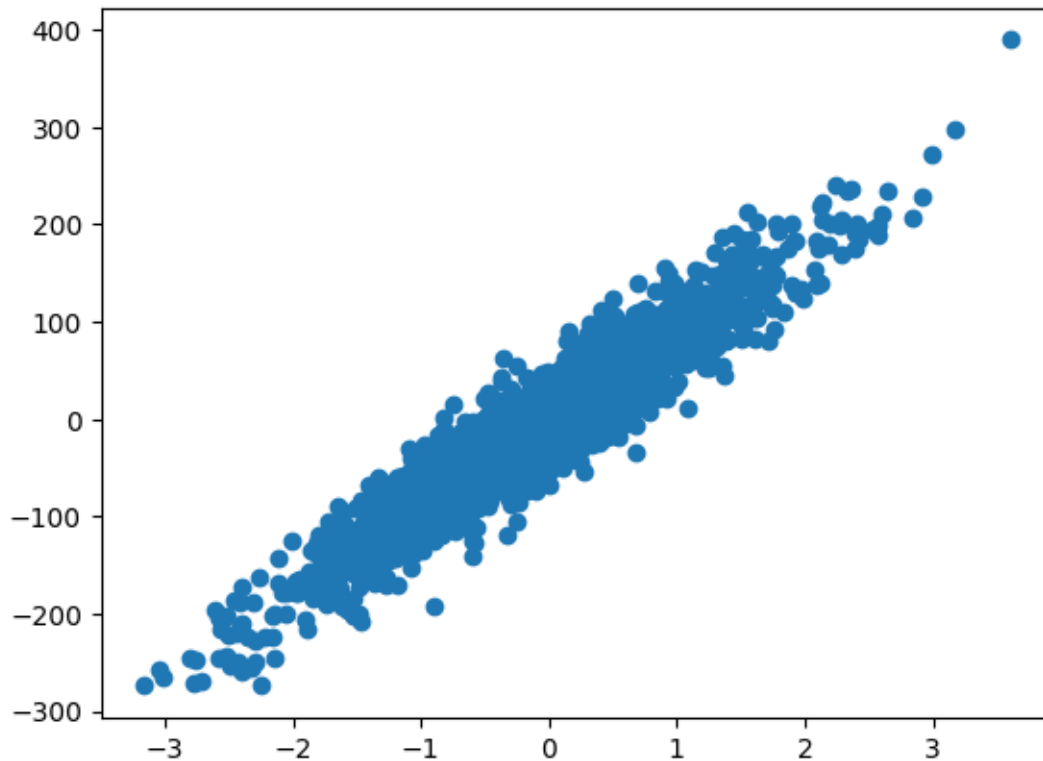
from sklearn.datasets import make_regression
from sklearn.model_selection import train_test_split
from sklearn.datasets import make_regression

import seaborn as sns
from matplotlib import pyplot as plt
```

## 0.1 Etape 1 : Réalisation d'un dataset

```
[93]: x, y = make_regression (n_samples=1500, n_features=1, noise=30) #génère un
      ↪ problème de régression
      plt.scatter(x,y) #on crée un graphe
```

```
[93]: <matplotlib.collections.PathCollection at 0x2a91abec6d0>
```



```
[94]: #on vérifie la validité des données
print(x.shape)
print(y.shape)

print(x, y)
#le y.shape n'est pas à la bonne dimension
y=y.reshape(y.shape[0], 1)
print(y.shape)
```

```
(1500, 1)
(1500,)
[[ 1.35654054]
 [ 1.01081209]
 [-0.52822065]
 ...
 [-0.94996755]
 [-1.79955866]
 [ 0.35981874]] [ 55.22405351  38.99489415 -37.06128537 ... -91.53573255
-119.36782723
 39.69126307]
(1500, 1)
```

```
[95]: #on crée une matrice de dataset contenant la colonne x et une autre colonne de 1
X=np.hstack((x, np.ones (x.shape)))
print(X)
print(X.shape)
```

```
[[ 1.35654054  1.         ]
 [ 1.01081209  1.         ]
 [-0.52822065  1.         ]
 ...
 [-0.94996755  1.         ]
 [-1.79955866  1.         ]
 [ 0.35981874  1.         ]]
(1500, 2)
```

```
[96]: #on initialise les paramètres a et b de la regression lineaire
theta=np.random.randn(2,1)
print(theta)
print(theta.shape)
```

```
[[ 0.01109034]
 [-1.87926288]]
(2, 1)
```

## 1 Etape 2 : Définition de modèle et son affichage

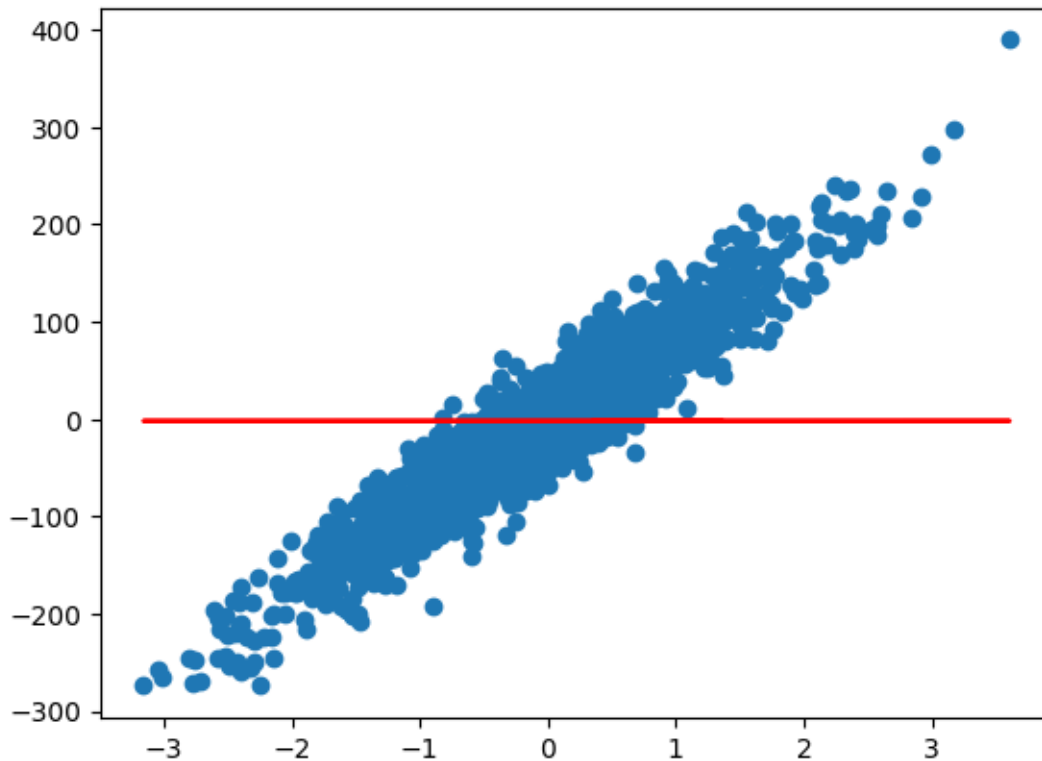
```
[97]: #on définit le modèle régression linéaire, multiplication de matrices
def model (X, theta):
    return X.dot(theta)
```

```
[98]: modele = model(X, theta)
print(modele)
print(modele.shape)
```

```
[[ -1.86421838]
 [ -1.86805263]
 [ -1.88512102]
 ...
 [ -1.88979834]
 [ -1.89922059]
 [ -1.87527237]]
(1500, 1)
```

```
[99]: plt.scatter(x,y)
plt.plot(x, model(X,theta), c='r')
#on voit que le modèle ne répond pas à nos attentes
```

```
[99]: [<matplotlib.lines.Line2D at 0x2a91acbb190>]
```



```
[100]: def cout(X,y,theta):
        m=len(y)
        return 1/(2*m)*np.sum((model(X,theta)-y)**2)
```

```
[101]: cout = cout(X, y, theta)
        print(cout.shape)
```

()

## 2 Quatrième étape : Algorithme de descente de gradient

```
[102]: #on crée une fonction descente_gradient pour réduire la fonction cout
def grad(X, y, theta):
    m=len(y)
    return (1/m)*X.T.dot(model(X,theta)-y)

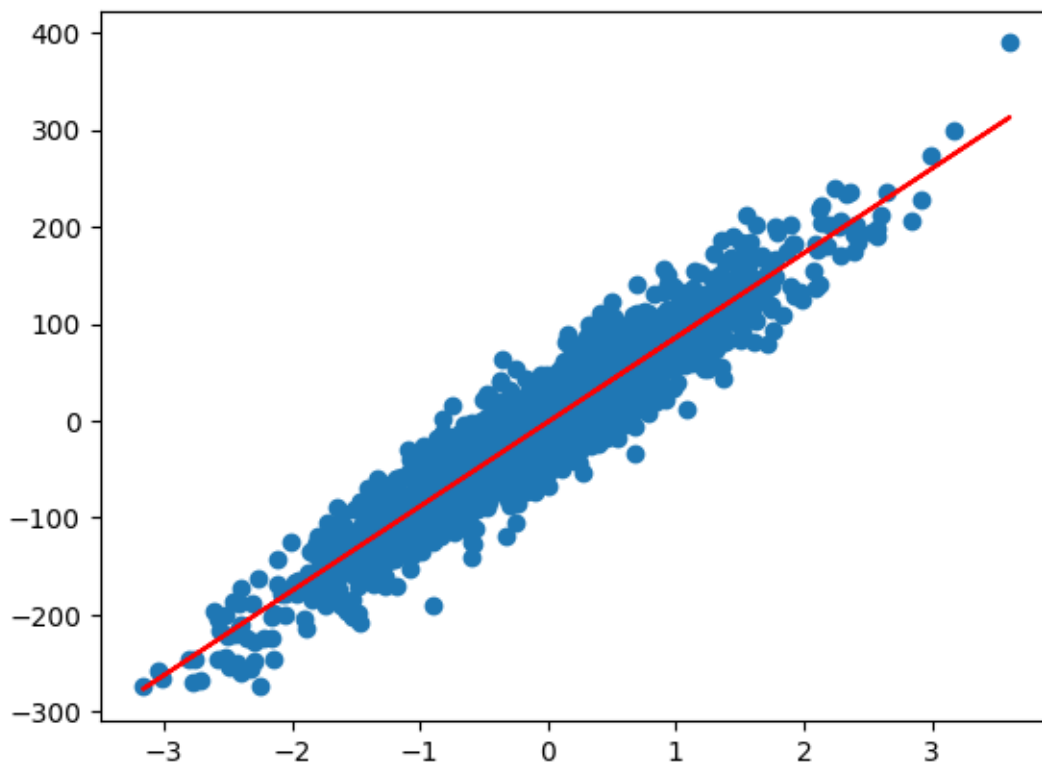
def DG (X,y, theta, learning_rate, n_iterations):
    for i in range(0, n_iterations):
        theta=theta-learning_rate*grad(X,y, theta)
    return theta
```

```
[103]: #on teste la descente des gradients sur notre modèle  
thetaF = DG(X, y, theta, 0.5, 1000)  
print(thetaF)
```

```
[[87.10523149]  
 [-1.49853517]]
```

```
[104]: prediction=model(X, thetaF)  
plt.scatter(x,y)  
plt.plot(x, prediction, c='r')
```

```
[104]: [<matplotlib.lines.Line2D at 0x2a91af67b90>]
```



```
[107]: def DG (X,y, theta, learning_rate, n_iterations):  
        histCout=np.zeros(n_iterations)  
        for i in range(0, n_iterations):  
            theta=theta-learning_rate*grad(X,y, theta)  
            histCout[i]= cout(X,y,theta)  
        return theta, histCout
```

```
[ ]: thetaF, histCout=DG(X,y, theta, learning_rate =0.01, n_iterations=1000)  
thetaF, histCout
```

```
[ ]: plt.plot(range(1000),histCout)
```

```
[ ]: def coefDet(y, prediction):  
    u=((y-prediction)**2).sum()  
    v=((y-y.mean())**2).sum()  
    return 1-u/v
```

```
[ ]: print(coefDet(y, prediction))
```