Royaume du Maroc
Université Mohammed Premier
École Supérieure De Technologie-Oujda
Département : Informatique
Filière : LP Informatique Décisionnelle

Analyse De Données

La régression linéaire ‘TP5’

Réalisé par : HNIOUA Abdessamad

Encadré par : M. Mounir Grari

Année Universitaire : 2023/2024

I. Introduction

La régression linéaire, une méthode fondamentale en statistique et en analyse de données, permet de modéliser les relations entre variables. En traçant une ligne droite ou un hyperplan, elle cherche à expliquer et prédire les variations dans une variable cible en fonction de variables explicatives. Ce rapport explore les principes de la régression linéaire, ses applications et son importance dans divers domaines, allant de la finance à la science des données.

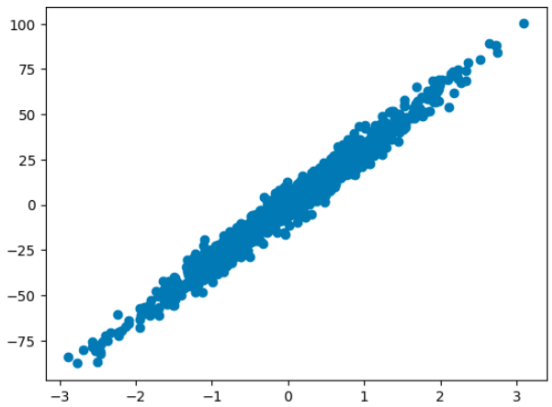
II. Implémention

- ## Dataset

```
import numpy as np
from sklearn.datasets import make_regression
import matplotlib.pyplot as plt
```

sklearn.datasets est un module dans la bibliothèque scikit-learn (ou sklearn) qui contient plusieurs ensembles de données intégrés et des fonctions pour charger et générer des ensembles de données. Ces ensembles de données sont souvent utilisés pour expérimenter et tester des algorithmes d'apprentissage automatique.


```
x, y = make_regression(n_samples=1000, n_features=1, noise=5)
plt.scatter(x, y) # afficher les résultats.
```



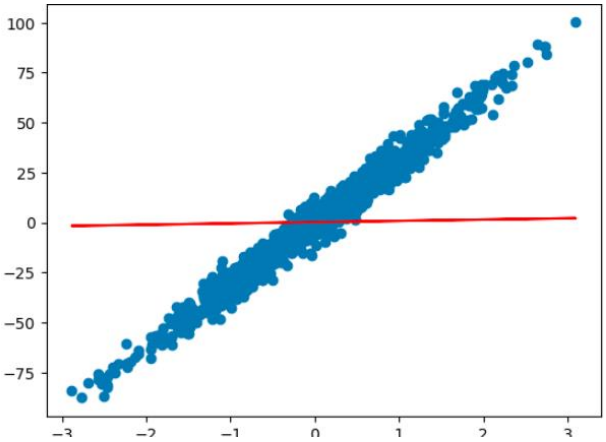
Nous obtiendrons un graphique avec un nuage de points où les valeurs de x sont sur l'axe des abscisses et les valeurs de y sont sur l'axe des ordonnées,

```
y = y.reshape(y.shape[0], 1)
print(y.shape)
X = np.hstack((x, np.ones(x.shape)))
print(X.shape)
```

Nous redimensionnons le vecteur y pour qu'il ait les dimensions (1000, 1) à l'aide de la fonction np.reshape. Ensuite, nous construisons la matrice X en empilant les caractéristiques x avec une colonne de un à l'aide de np.hstack, pour former une matrice avec les dimensions (1000, 2).

- **Modèle Linéaire**

```
theta = np.random.randn(2, 1)
theta
def cost_function(X, y, theta):
    m = len(y)
    return 1/(2*m) * np.sum((model(X, theta) - y)**2)
def model(X, theta):
    return X.dot(theta)
plt.scatter(x, y)
plt.plot(x, model(X, theta), c='r')
```



Ce code initialise un vecteur de paramètres aléatoire, calcule le coût à partir des prédictions du modèle, définit un modèle de régression linéaire, et visualise les données et la droite de régression correspondante.

- **Fonction Cout**

  On mesure les erreurs du modèle sur le Dataset X, y en implémenter l'erreur quadratique moyenne, (MSE) en anglais. Ensuite, on teste notre fonction, pour voir s'il n'y a pas de bug.

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

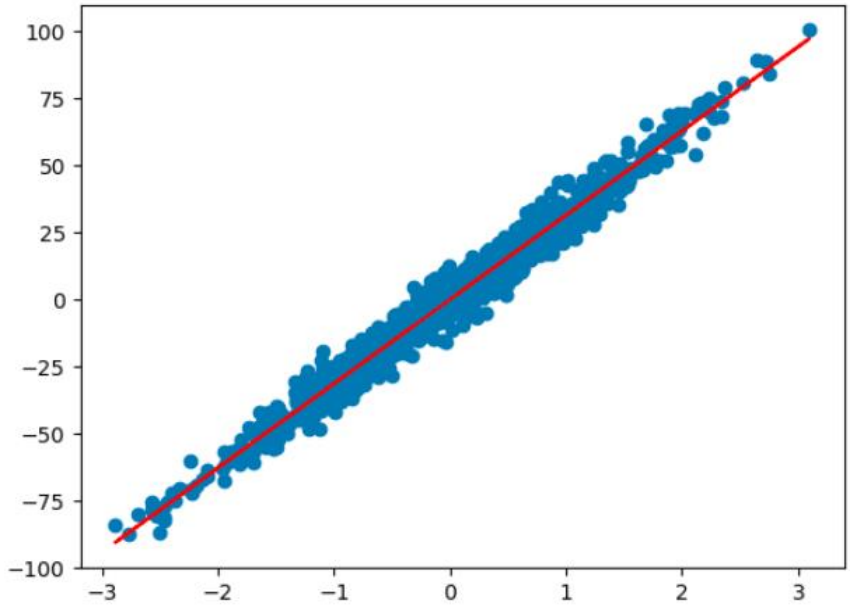
- **Gradients et Descente de Gradient.**

```
def grad(X, y, theta):
    m = len(y)
    XT = np.transpose(X)
    return 1/m * XT.dot(model(X, theta) - y)
```

```
def gradient_descent(X, y, theta, learning_rate, n_iterations):
    cost_history = np.zeros(n_iterations
    for i in range(0, n_iterations):
        theta = theta - learning_rate * grad(X, y, theta
        cost_history[i] = cost_function(X, y, theta
```

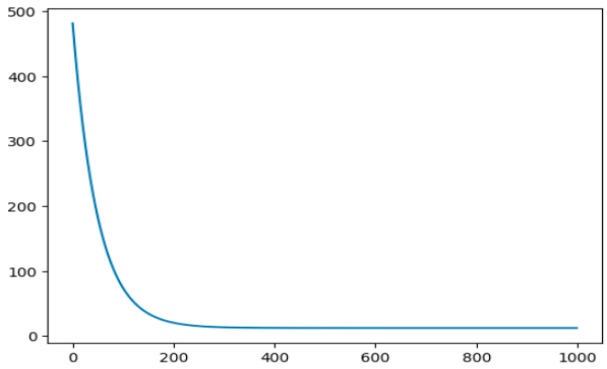
- **Phase d'entraînement**

```
theta_final, cost_history = gradient_descent(X, y, theta,
learning_rate=0.01, n_iterations=1000)
predictions = model(X, theta_final)
plt.scatter(x, y)
plt.plot(x, predictions, c='r')
```



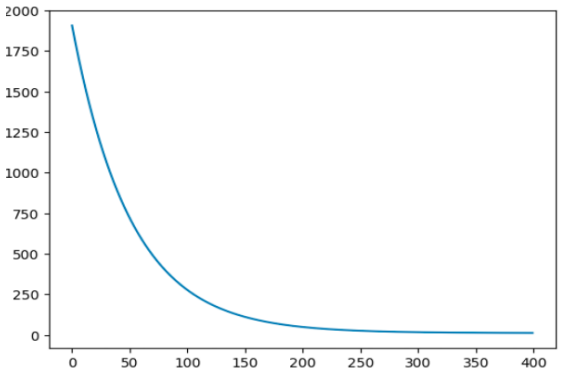
Le code entraîne un modèle de régression linéaire en utilisant la descente de gradient, fait des prédictions sur les données d'entraînement à l'aide des paramètres finaux du modèle, puis trace les données d'entraînement ainsi que la droite de régression obtenue.

- **Courbes d'apprentissage**



La courbe du coût reste constante après 400 itérations sur un total de 1000 itérations, cela peut indiquer plusieurs problèmes potentiels dans votre algorithme de descente de gradient.

Il est important de noter que l'algorithme de descente de gradient peut être sensible à différents paramètres et configurations, et il peut nécessiter un ajustement expérimental pour trouver la meilleure convergence possible.



L'algorithme s'arrêtera dès qu'il aura atteint l'itération 400 et que la différence entre le coût actuel et le coût précédent est inférieure à un seuil défini (cost_threshold).

- **Evaluation finale**

```
def coef_determination(y, pred):
    u = ((y - pred)**2).sum()
    v = ((y - y.mean())**2).sum()
    return 1 - u/v
```

```
0.9755274033673896
```

Un coefficient de détermination de **0.97** suggère que le modèle explique **97 %** de la variance des données, ce qui est considéré comme un excellent ajustement dans de nombreux contextes. Cela signifie que le modèle est capable de prédire efficacement les valeurs cibles à partir des caractéristiques d'entrée.