Université Sultan Moulay Slimane Faculté Polydisciplinaire Béni Mellal
Département INFORMATIQUE (MIP)

Filière : Science de données et sécurité des systèmes d'information
A.U : 2023-2024

Sujet

Compte Rendu de TP02 – Apprentissage Automatique

Présenté Par :
MAFTOUH Omar

Encadré Par :
A. MAARIR

Introduction :

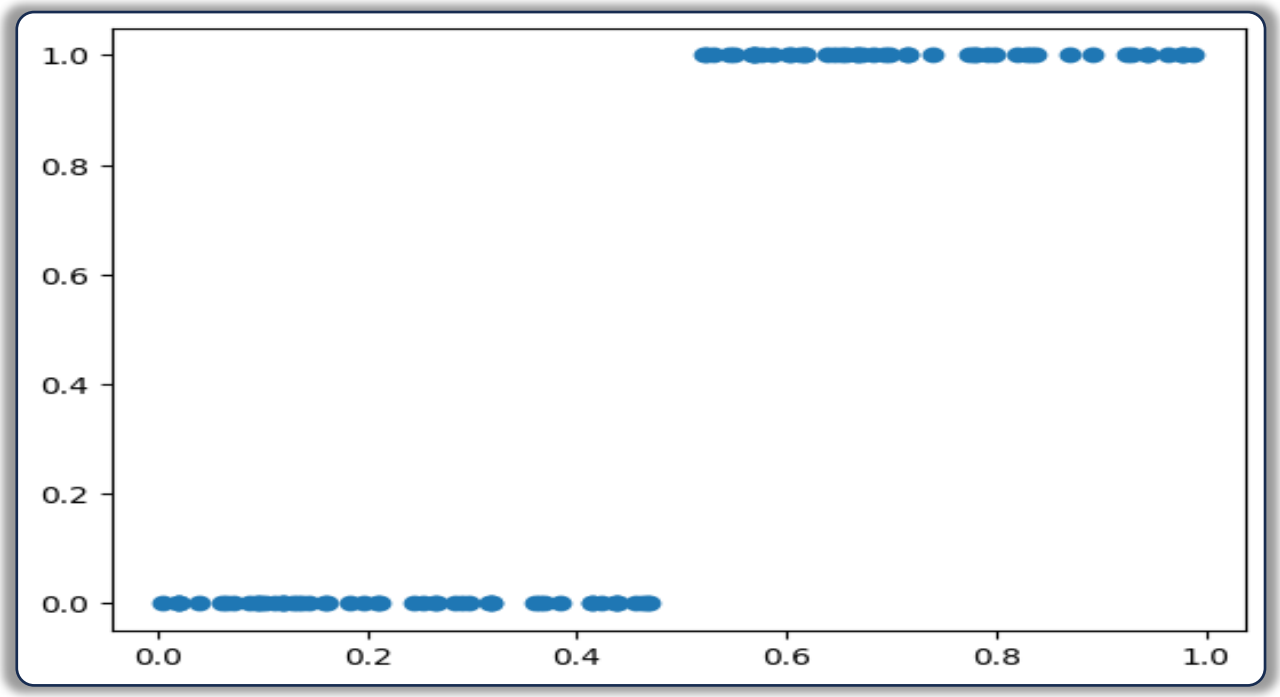
Le présent compte rendu porte sur la mise en œuvre pratique de la régression logistique, une technique essentielle en machine learning, visant à modéliser la probabilité qu'une observation appartienne à une classe spécifique. Ce travail consiste à comprendre et à implémenter les étapes fondamentales de la régression logistique, telles que la préparation des données, la fonction sigmoïde, la fonction de coût, l'entraînement du modèle et son évaluation. Nous explorerons également la comparaison des résultats obtenus avec notre implémentation et celle de la bibliothèque scikit-learn.

1. Préparation et Visualisation des Données :

Nous avons commencé par générer un ensemble de données synthétiques adaptées à la classification binaire, comprenant 100 observations. En utilisant la fonction train_test_split de la bibliothèque scikit-learn, nous avons divisé ces données en ensembles d'entraînement et de test, avec une proportion de 80% pour l'entraînement et 20% pour les tests. Enfin, nous avons visualisé graphiquement les données pour avoir un aperçu visuel de la distribution des points dans l'espace.

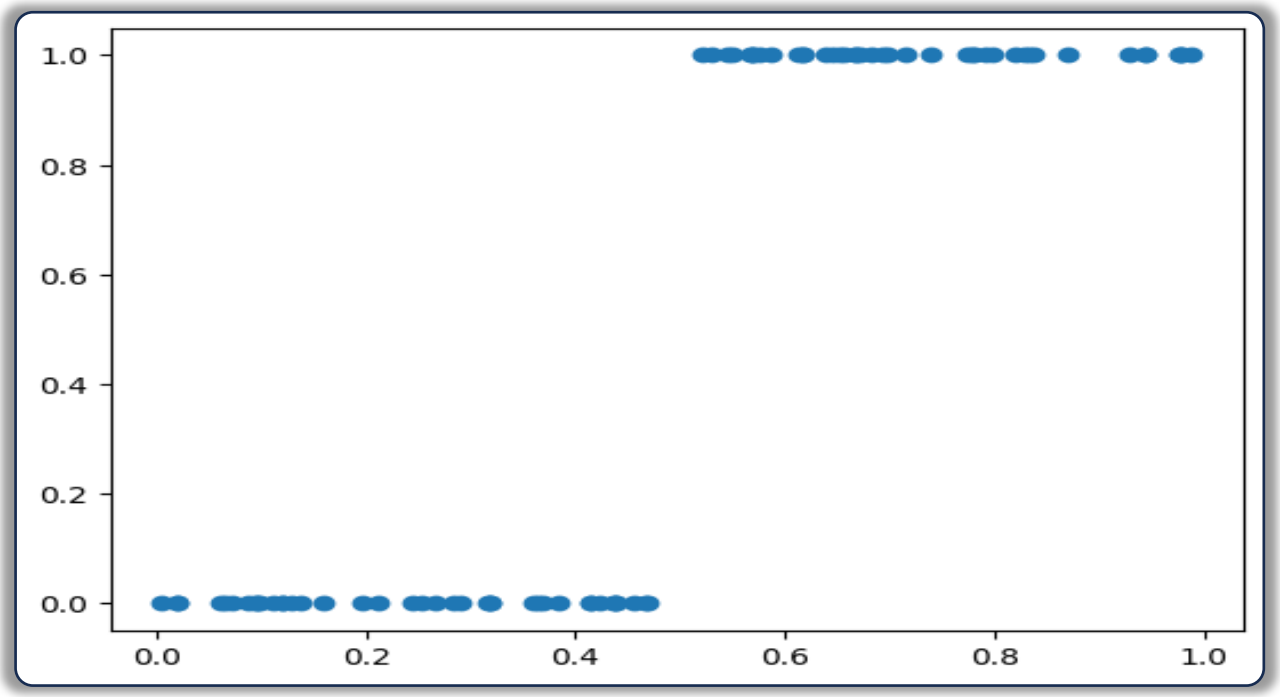
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split

# 1
np.random.seed(0)
n_observation = 100
X = np.random.randn(n_observation , 1)
y = (X > 0.5).astype(int).ravel()
plt.scatter(X , y)
```

Résultat du programme :



```
# 2
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2,
random_state=0)

# 3
plt.scatter(X_train, Y_train)
plt.show()
```

Résultat du programme :



2.Implémentation de la Régression Logistique :

Fonction Sigmoïde :

La première étape de notre implémentation a été la définition de la fonction sigmoïde, qui est cruciale dans la régression logistique pour transformer les valeurs d'entrée en valeurs comprises entre 0 et 1, représentant des probabilités.

```
def sigmoide(z):
    return 1 / (1 + np.exp(-z))
```

Fonction de Coût et Descente de Gradient :

Nous avons ensuite implémenté la fonction de coût pour évaluer les performances de notre modèle ainsi que la descente de gradient pour optimiser les paramètres du modèle. La fonction de coût est essentielle pour mesurer l'erreur entre les prédictions du modèle et les valeurs réelles, tandis que la descente de gradient permet de mettre à jour itérativement les paramètres du modèle pour minimiser cette erreur.

```
def fonction_cout(X, Y, theta):
    m = len(Y)
    h = sigmoide(X.dot(theta))
    cout = (1 / m) * (-np.dot(Y.T, np.log(h)) - (1 - Y).T.dot(np.log(1-h)) )
    return cout
def descent_gradient(X, Y, theta, alpha, n_iteration):
    m = len(Y)
    cout_historique = np.zeros(n_iteration)
    for i in range(n_iteration):
        h = sigmoide(X.dot(theta))
        gradient = (1 / m)*X.T.dot(h-Y)
        theta -= alpha*gradient
        cout_historique[i] = fonction_cout(X, Y, theta)
    return theta, cout_historique
```

Entraînement du Modèle :

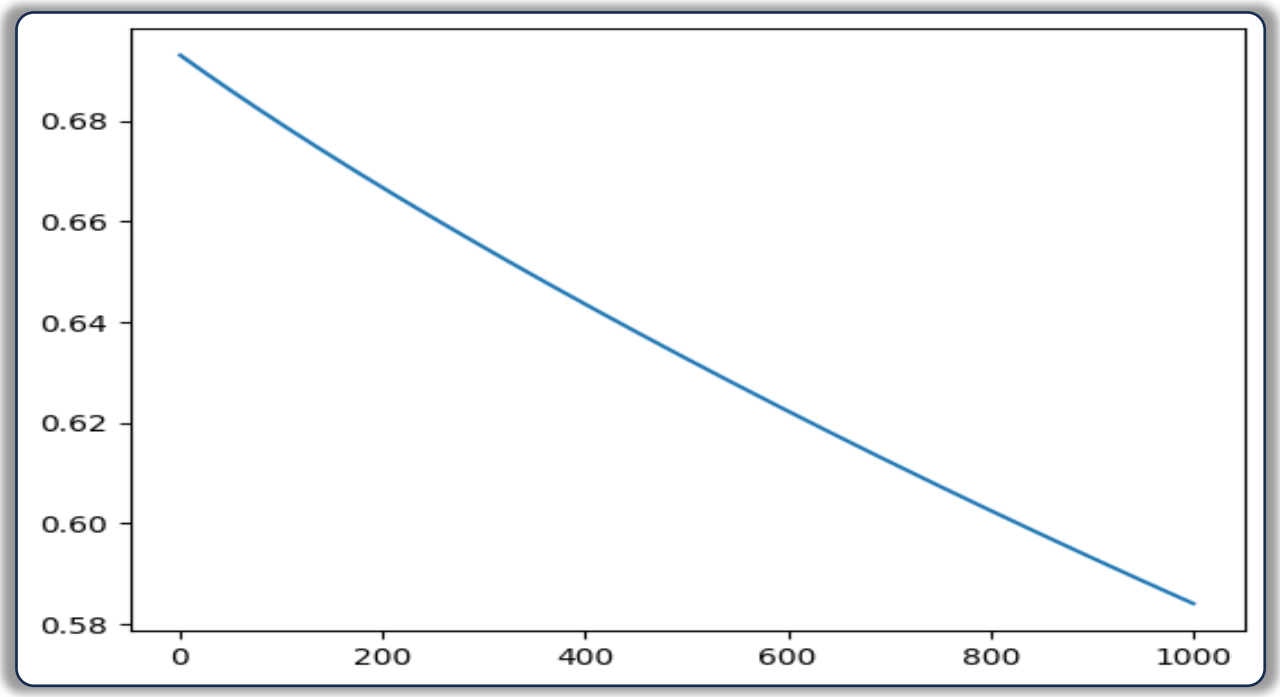
En utilisant les fonctions définies précédemment, nous avons entraîné notre modèle de régression logistique sur l'ensemble de données d'entraînement. Nous avons surveillé la convergence de la fonction de coût pendant l'entraînement pour nous assurer que notre modèle apprend de manière efficace.

```
X = np.c_[np.ones(n_observation), X]
alpha = 0.01
n_iteration = 1000
theta = np.zeros(X.shape[1])

theta_final, cout_historique = descent_gradient(X, Y, theta, alpha,
n_iteration)

theta_final # array([-0.33846446,  0.9804234 ])
```

Évaluation du Modèle :

Une fois le modèle entraîné, nous l'avons évalué en utilisant l'ensemble de données de test. Nous avons calculé des métriques d'évaluation telles que la précision, qui mesure la proportion d'observations correctement prédites par le modèle.

```
plt.plot(range(1, n_iteration + 1), cout_historique)
plt.show()
```



Comparaison avec scikit-learn :

Enfin, nous avons comparé les résultats obtenus avec notre implémentation à ceux de la bibliothèque scikit-learn en utilisant sa classe LogisticRegression. Cette comparaison nous permet de valider notre implémentation en la confrontant à une solution éprouvée et largement utilisée dans la communauté du machine learning.

```
def Eval_precision (y_Vrai , y_pred) :
    vrais_positive = 0
    faux_positive = 0

    for vrai , pred in zip(y_Vrai , y_pred) :
        if pred == 1 and vrai == 1 :
            vrais_positive += 1
        elif pred == 1 and vrai != 0 :
            faux_positive +=1

    if faux_positive + vrais_positive == 0 :
        return 0

    return vrais_positive / ( vrais_positive + faux_positive )
```

Conclusion :

Ce TP sur la régression logistique nous a permis de maîtriser les fondements pratiques de cet algorithme clé en apprentissage automatique. En suivant les étapes, de la préparation des données à l'évaluation du modèle, nous avons consolidé notre compréhension des concepts essentiels tels que la fonction sigmoïde, la fonction de coût et la descente de gradient.

L'analyse des résultats obtenus, associée à une comparaison avec la solution de référence de scikit-learn, nous a fourni des insights précieux sur les performances de notre implémentation. Ce TP constitue ainsi une base solide pour explorer davantage de techniques en machine learning et aborder des problèmes plus complexes dans ce domaine dynamique.