Atelier : Régression Logistique avec Python

Exercice 1 : Comprendre la Régression Logistique

Explorer la relation entre les probabilités et la fonction logistique.

Tâche :

Implémentez la **fonction logistique** en Python : σ(z)=1/1+e−z

Tracez la courbe de la fonction logistique pour des valeurs de z comprises entre -10 et 10.

Code de départ :

```
 import numpy as np
import matplotlib.pyplot as plt
# Définir la fonction logistique
def sigmoid(z):
    return 1 / (1 + np.exp(-z))
# Générer des valeurs de z
z = np.linspace(-10, 10, 100)
sigma = sigmoid(z)
# Tracer la courbe
plt.plot(z, sigma)
plt.title("Courbe de la fonction logistique")
plt.xlabel("z")
plt.ylabel("Sigmoid(z)")
plt.grid()
plt.show()
```

Questions :

Quelle est la plage des valeurs que peut prendre la fonction sigmoid ?

Pourquoi cette fonction est-elle utile pour modéliser une probabilité ?

Exercice 2 : Application à un Jeu de Données Synthétique

Appliquer une régression logistique pour un jeu de données simple à deux classes.

Tâche :
Créez un jeu de données synthétique avec deux classes. Appliquez une régression logistique pour séparer ces deux classes.

Illustration Python :

```
 from sklearn.datasets import make_classification
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
# Générer des données
X, y = make_classification(n_samples=200, n_features=2, n_classes=2,
n_clusters_per_class=1, random_state=42)
# Diviser en ensembles d'entraînement et de test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
# Appliquer une régression logistique
model = LogisticRegression()
model.fit(X_train, y_train)
# Prédire sur l'ensemble de test
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Précision du modèle : {accuracy:.2f}")
```

Questions :
Quelles sont les limites d'une régression logistique avec deux classes ?

Comment améliorer la précision dans ce cas ?

Exercice 3 : Visualisation de la Frontière de Décision

Visualiser la frontière de décision générée par la régression logistique.

Tâche :
Ajoutez un graphique pour visualiser la **frontière de décision** de la régression logistique.

Illustration Python :

```
import matplotlib.pyplot as plt
# Fonction pour tracer la frontière de décision
def plot_decision_boundary(model, X, y):
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.01),
                         np.arange(y_min, y_max, 0.01))
    Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    plt.contourf(xx, yy, Z, alpha=0.8, cmap=plt.cm.Paired)
    plt.scatter(X[:, 0], X[:, 1], c=y, edgecolor='k', cmap=plt.cm.Paired)
    plt.show()
# Appeler la fonction
plot_decision_boundary(model, X, y)
```

Questions :
La frontière de décision est-elle linéaire ? Pourquoi ?

Que se passe-t-il si les données sont non linéairement séparables ?

Exercice 4 : Interprétation des Coefficients

Affichez les coefficients de la régression logistique et interprétez leur rôle.

Code de départ :

```
 # Afficher les coefficients
print("Coefficients :", model.coef_)
print("Intercept :", model.intercept_)
# Interprétation
for i, coef in enumerate(model.coef_[0]):
    print(f"Importance de la caractéristique {i+1} : {coef:.2f}")
```

Questions à discuter :

Comment interpréter les coefficients d'une régression logistique ?

Que représente l’intercept ?

Exercice 5 : Utilisation sur un Jeu de Données Réel (Titanic)

Appliquer une régression logistique pour prédire la survie sur le Titanic.

Tâche :
Téléchargez le jeu de données Titanic depuis Seaborn.

Prétraiter les données pour les rendre utilisables.

Appliquez une régression logistique pour prédire la survie (y=1y = 1).

Illustration Python :

```
 import seaborn as sns
import pandas as pd
from sklearn.preprocessing import StandardScaler
# Charger les données
data = sns.load_dataset("titanic")
data = data[["age", "fare", "sex", "class", "survived"]].dropna()
# Encodage des variables catégoriques
data["sex"] = data["sex"].map({"male": 0, "female": 1})
data["class"] = data["class"].map({"Third": 0, "Second": 1, "First": 2})
# Séparer les caractéristiques et la cible
X = data[["age", "fare", "sex", "class"]]
y = data["survived"]
# Normalisation des caractéristiques
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
# Régression logistique
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.3,
random_state=42)
model = LogisticRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
print(f"Précision sur les données réelles : {accuracy_score(y_test, y_pred):.2f}")
```

Questions :
Quels facteurs influencent le plus la survie ?

Comment améliorer les performances en ajustant les hyperparamètres ?