TP : Perceptron en Python

Dans ce TP, nous allons créer un perceptron en utilisant la bibliothèque `numpy` pour les opérations numériques et `matplotlib` pour la visualisation. Le perceptron sera entraîné sur un jeu de données bidimensionnel pour effectuer une classification binaire.

Étape 1 : Importation des bibliothèques

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

Étape 2 : Définition de la classe Perceptron

Vous pouvez télécharger et compléter le fichier Perceptron.py disponible sur Nextcloud.

Constructeur

Le constructeur de la classe `Perceptron` initialise les attributs nécessaires pour créer et entraîner un perceptron. Les paramètres sont :

- `self.poids` : Un vecteur de poids initialisé aléatoirement. Chaque dimension de ce vecteur correspond aux poids associés à chaque caractéristique d'entrée du modèle.
- `self.bias` : Le e biais initialisé aléatoirement. Il s'agit d'une constante qui influence la sortie du perceptron indépendamment des caractéristiques d'entrée.
- `self.alpha` : Le taux d'apprentissage, qui contrôle les ajustements effectués aux poids et au biais lors de l'apprentissage. Il est initialement fixé à 0.01.
- `self.epochs` : Le nombre d'itérations sur l'ensemble des données d'entraînement. Une `epoch` correspond à une passe complète à travers toutes les données d'entraînement. Ce paramètre est initialement fixé à 100.

```
def __init__(self, n, alpha=0.01, epochs=100):
  self.poids = np.random.rand(n)
  self.bias = np.random.rand()
```

```
self.alpha = alpha
self.epochs = epochs
```

0.1 Fonction d'activation

La méthode `fonction_activation` est responsable de définir la fonction d'activation utilisée par le perceptron. Pour ce TP, on utilise la fonction *threshold* vu en cours.

```
def fonction_activation(self, x):
  #TODO
```

0.2 Prédiction

La méthode `prediction` prend un vecteur d'entrée X en argument et renvoie la sortie prédite par le perceptron.

```
def prediction(self, X):
    #TODO
```

0.3 Entraînement

La méthode `entrainement` permet d'ajuster les poids du Perceptron en utilisant les données d'entrainements. L'ajustement des poids est effectué itérativement selon les étapes suivantes

- À chaque époque, le perceptron parcourt tous les exemples du jeu de données d'entraînement.
- Pour chaque donnée X dans X_train une prédiction y_pred est calculée. Si la prédiction est différente de la sortie attendue y, les poids et le biais sont ajustés pour corriger l'erreur. Les ajustements sont effectués en utilisant la règle de mise à jour du perceptron vu en cours.
- Le processus est répété pour le nombre d'époques spécifié lors de la création de l'instance du perceptron.

```
def entrainement(self, X_train, Y_train):
  #TODO
```

Affichage du séparateur linéaire

La méthode `affichage_separateur` dans la classe `Perceptron` est responsable de l'affichage du séparateur linéaire (la droite de décision) appris par le perceptron après l'entraînement.

```
def affichage_seperateur(self, X_train, Y_train):
  #TODO
```

Test et paramètrages

- Testez votre peceptron sur le jeu de données disponible sur Nextcloud.
- Evaluer sa performance sur les données de tests.
- Faites varier le nombre d'itérations (*epoch*) et étudier son impact sur la performance de votre perceptron.

```
# Charger les donnees depuis un fichier CSV
data_train = pd.read_csv("data_train.csv")
X_train = data_train[['X1','X2']]
Y_train = data_train['Y']
print(Y_train)
# Creation et entrainement du perceptron
perceptron = Perceptron(2)
perceptron.entrainement(X_train.values, Y_train.values)
#affichage de la droite separatrice
perceptron.affichage_seperateur(X_train.values, Y_train.values)
#evaluer la performance du perceptron
data_test = pd.read_csv("data_test.csv")
X_test = data_test[['X1','X2']]
Y_test = data_test['Y']
print(perceptron.evaluate(X_test.values,Y_test.values))
```