ZANCHETTA ENZO M1 CHPS

Programmation d'un Perceptron en Python

Ce rapport a pour but d'expliciter certains choix de développement ainsi que des problèmes rencontrés et des adaptations ayant été nécessaires lors du développement de ce projet.

```python
import numpy as np
from perceptron_data import bias
from perceptron_data import iris
from numpy.random import rand # Mon python n'arrivait pas à importer pylab
# import matplotlib.pyplot as plt -> Je ne peux pas l'installer "Package python-matplotlib is not available, but i
# Tout le code concernant maplotlib sera sous commentaire car je ne peux pas le faire fonctionner sur mon terminal
```

Comme expliqué dans les entêtes je n'ai pas pu utiliser certaines librairies telles que Pylab qui n'étaient pas détecté par mon python malgré de multiples tentatives ainsi que Matplotlib.

Pour Pylab, j'ai pu contourner le problème principal en utilisant la fonction rand de base et en passant directement par le module Matplotlib pour tracer les courbes.

Pour Matplotlib, comme indiqué en commentaires, mon linux ne veux pas le télécharger en m'indiquant qu'il est disponible par d'autres sources mais je ne peux pas accéder au module même en passant par ces autres sources. C'est pour cela que toutes les parties de Matplotlib sont sous commentaires sinon le code ne s'exécuterait pas lorsque je le teste. Je n'ai donc pas pu tester mon affichage par Matplotlib mais toutes les lignes de codes sont implémentés et devraient s'exécuter si python arrive à charge le module Matplotlib.

Je ne m'attarderai pas sur les fonctions basiques que sont produitscalaire() qui est très simple et genererDonnees() qui n'est que l'implémentation de l'algorithme donné en pseudo-code.

```python
def perceptron_apprentissage(N,n): # Fonction
    w = []
    predicty = 0
    apprentissage = genererDonnees(n)
    nbx = len(apprentissage[0][0])
    for i in range(nbx):
        w.append(0)
    nbligne = len(apprentissage)
    for k in range(1,N):
        for i in range(nbligne):
            x = apprentissage[i][0]
            y = apprentissage[i][1]
            scal = produitscalaire(w,x)
            if scal < 0:
                predicty = False
            else:
                predicty = True
            if predicty != y:
                if y == True:
                    for j in range(nbx):
                        w[j] = w[j] + x[j]
                else:
                    for j in range(nbx):
                        w[j] = w[j] - x[j]
    return w
```

Cette fonction correspond à l'implémentation du perceptron pour les données d'apprentissage. Elle prend en paramètres N le nombre d'itérations sur les données que l'on veut faire et n la taille du jeu de données généré.

Après création du jeu de données, j'initialise mon perceptron avec une taille adapté puis je rentre dans ma boucle d'itération.

A l'intérieur de celle-ci je vais, pour chaque ligne, extraire les éléments de la ligne sur laquelle le perceptron travaille, faire le produit scalaire entre le perceptron et les valeurs de jeu de données et je vais me servir de ce produit afin de faire une prédiction sur la valeur de l'étiquette. En fonction de la justesse ou de l'erreur de ma prédiction, j'adapte mon perceptron.

Le traitement reste le même pour la fonction de test, qui lui travaille sur le jeu de données test, ainsi que pour le perceptron avec biais qui prend ce biais en paramètres et modifie le produit scalaire de la même façon qu'indiqué dans le sujet mais dont le fonctionnement reste identique aux fonctions précédentes.

Je ne peux malheureusement pas faire de commentaires sur les résultats car, comme indiqué plus haut, je n'ai pas pu tester et commenter mes affichages de courbes et car j'ai eu l'impression de ne pas bien saisir le sujet.

J'ai bien compris qu'elles étaient les implémentations attendues ainsi que le fonctionnement des divers formes de perceptron mais j'avoue ne pas avoir très bien saisit quels étaient les résultats attendus de ces implémentations et les formes qu'ils devaient prendre.