TD/TD : Implémentation des méthodes des Kplus proche Voisins et du Perceptron en Python

TP 1 : L'algorithme KppV

Vous devez créer un programmes permettant d'étudier l'algorithme de discrimination étudié durant le cours, à savoir la méthode KppV :

```
[clas]=decision_kppv(test,apprentissage,oracle,k);
```

Ce programme applique la méthode de discrimination de kppv sur un ensemble d'individu élément de R^2.

Les paramètres :

- La variable `test` est un tableau qui doit contenir les différents individus à classer rangés par colonne. Le nombre de ligne est 2 et le nombre de colonne est n.
- La variable `apprentissage` est un tableau qui doit contenir les différents individus de l'ensemble d'apprentissage rangés par colonne. Le nombre de ligne est 2 et le nombre de colonne est m.
- La variable `oracle` est un vecteur qui indique la classification de l'ensemble d'apprentissage. `oracle[i]` indique le numéro de la classe de l'individu `apprentissage[:,i]`.
- La variable `k` indique le nombre de voisins utilisés dans l'algorithme.

Le résultat :

- La variable `clas` est un vecteur qui indique le résultat de l'algorithme de la discrimination. `clas[i]` indique le numéro de la classe de l'individu `x[:,i]`.

— *Réaliser la fonction.*

Pour vous aidez, nous fournissons un début de script Python avec la fonction

```
affiche_classe(x,clas,K);
```

Cette fonction permet de visualiser graphiquement dans une représentation planaire le résultat de la classification.

Afin de tester votre programme de KppV, vous allez créer un ensemble d'apprentissage correspondant à l'acquisition de 2 classes d'individus chacun associé à une distribution gaussienne bidimensionnelle.

La première classe est composée de 128 individus et est associée à la fonction de densité de la loi normale $\mathcal{N}_2((4,4)^T, I_2)$:

$$f(x) = \frac{1}{2\pi} \exp\left\{-\frac{1}{2}\left(x - \begin{pmatrix} 4 \\ 4 \end{pmatrix}\right)^T \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}^{-1} \left(x - \begin{pmatrix} 4 \\ 4 \end{pmatrix}\right)\right\}$$

La seconde classe est composée de 128 individus et est associée à la fonction de densité de la loi normale $\mathcal{N}_2((-4,-4)^T, 4I_2)$:

$$f(x) = \frac{1}{8\pi} \exp\left\{-\frac{1}{2}\left(x - \begin{pmatrix} -4 \\ -4 \end{pmatrix}\right)^T \begin{bmatrix} 4 & 0 \\ 0 & 4 \end{bmatrix}^{-1} \left(x - \begin{pmatrix} -4 \\ -4 \end{pmatrix}\right)\right\}$$

— *Pour créer ces données d'apprentissage, vous utilisez Python*

```
numpy.random.multivariate_normal(...)
data=numpy.concatenate(...)
%x2=[rand(1,128,'normal')*2-4;rand(1,128,'normal')*2-4];
%x=[x1 x2 ];
```

Ensuite nous allons créer un ensemble de test, 64 éléments issus de la première classe, 64 éléments issus de la seconde classe.

Pour vous aidez, nous fournissons un début de script Python.

— *D*évelopper l'algorithme de KppV.

— *Appliquer l'algorithme de Kppv sur vos données de tests.*

— *Etudier graphiquement le résultat de la partition. Calculer l'erreur et tester pour différentes valeurs de K.*

Trame (Python)

```
import numpy as np
import matplotlib.pyplot as plt
import random
import os



def kppv(x,appren,oracle,K):


      return clas


def affiche_classe(x,clas,K):
   for k in range(0,K):
       ind=(clas==k)
       plt.plot(x[0,ind],x[1,ind],"o")
   plt.show()

# Données de test
mean1 = [4, 4]
cov1 = [[1, 0], [0, 1]]  #
data1 = np.transpose(np.random.multivariate_normal(mean1, cov1, 128))
mean2 = [-4, -4]
cov2 = [[4, 0], [0, 4]]  #
data2 = np.transpose(np.random.multivariate_normal(mean2, cov2, 128))
data=np.concatenate((data1, data2), axis=1)

oracle=np.concatenate((np.zeros(128),np.ones(128)))

test1=np.transpose(np.random.multivariate_normal(mean1, cov1, 64))
test2=np.transpose(np.random.multivariate_normal(mean2, cov2,64))

test=np.concatenate((test1,test2), axis=1)

K=3
clas=kppv(test,data,oracle,K)
affiche_classe(test,clas,2)
```

TP 2 : Le Perceptron

Note : durant tout le TP, le coefficient d'apprentissage α sera égal à 0.1.

1 Mise en place d'un perceptron simple

— *Créer une fonction*

```
[y]=perceptron(x,w,active)
```

Ce programme doit évaluer la sortie d'un perceptron simple (1 neurone) pour une entrée élément de R^2.

Les paramètres :

- La variable `w` contient les poids synaptiques du neurone. C'est un vecteur à 3 lignes. La première ligne correspond au seuil.
- La variable `x` contient l'entrée du réseau de neurones. C'est un vecteur à 2 lignes.
- La variable `active` indique la fonction d'activation utilisée. Si `active==0` $\varphi(x)$ =sign(x) si `active==1` $\varphi(x) = \tanh(x)$

Le résultat :

- La variable `y` est un scalaire correspondant à la sortie du neurone.

2 Etude de l'apprentissage

— *A partir de la fonction* `perceptron` *créer une fonction*

```
[w,erreur]=apprentissage(x,yd)
```

Ce programme retourne le vecteur de poids w obtenu par apprentissage selon la règle d'apprentissage utilisant la descente du gradient.

Les paramètres :

- La variable `x` contient l'ensemble d'apprentissage. C'est une matrice à 2 lignes et n colonnes.
- La variable `yd(i)` indique la réponse désirée pour chaque élément `x(:,i)`. `yd` est un vecteur de 1 ligne et n colonnes de valeurs +1 ou -1 (classification à 2 classes).
- Vous utiliserez la fonction d'activation $\varphi(x) = \tanh(x)$ et on suggère d'utiliser 100 itérations (présentatino de 100 fois de l'ensemble d'apprentissage).

Le résultat :

- La variable `w` contient les poids synaptiques du neurone après apprentissage. C'est un vecteur à 3 lignes. La première ligne correspond au seuil.
- La variable erreur contient l'erreur cumulée calculée pour le passage complet de l'ensemble d'apprentissage à savoir

$$erreur = \sum_{i=0}^{N-1} (yd(i) - y(i))^2$$

La variable erreur sera un vecteur de taille fixée par le nombre d'itération. Cela permettra de représenter l'évolution de l'erreur au cours des itérations de l'apprentissage.

2.2 Test sur données simulées

Nous allons reprendre les données du premier TP. Vous allez créer un ensemble d'apprentissage correspondant à l'acquisition de 2 classes d'individus chacun associé à une distribution gaussienne bidimensionnelle.

La première classe est composée de 128 individus et est associée à la fonction de densité de la loi normale $\mathcal{N}_2((4,4)^T, I_2)$:

$$f(x) = \frac{1}{2\pi} \exp\left\{ -\frac{1}{2} \left(x - \begin{pmatrix} 4 \\ 4 \end{pmatrix} \right)^T \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}^{-1} \left(x - \begin{pmatrix} 4 \\ 4 \end{pmatrix} \right) \right\}$$

La seconde classe est composée de 128 individus et est associée à la fonction de densité de la loi normale $\mathcal{N}_2((-4,-4)^T, 4I_2)$:

$$f(x) = \frac{1}{8\pi} \exp\left\{ -\frac{1}{2} \left(x - \begin{pmatrix} -4 \\ -4 \end{pmatrix} \right)^T \begin{bmatrix} 4 & 0 \\ 0 & 4 \end{bmatrix}^{-1} \left(x - \begin{pmatrix} -4 \\ -4 \end{pmatrix} \right) \right\}$$

— *Pour créer ces données d'apprentissage, vous utilisez Python*

```
numpy.random.multivariate_normal(...)
data=numpy.concatenate(...)
```

— *Appliquer votre algorithme d'apprentissage.*
— *Afficher l'erreur*
— *Pour évaluer la séparation associée au neurone, nous fournissons une fonction* `affiche_classe(x,clas,K,w)`. *Etudier le code de la fonction pour la comprendre, l'utiliser pour évaluer la qualité de votre apprentissage.*

Trame programme (Python)

```
import numpy as np
import matplotlib.pyplot as plt
import random

def perceptron(x,w,active):



    return y


def apprentissage(x,yd,active):


    return w, mdiff


def affiche_classe(x,clas,K,w):

    t=[np.min(x[0,:]),np.max(x[0,:])]
    z=[(-w[0,0]-w[0,1]*np.min(x[0,:]))/w[0,2],(-w[0,0]-w[0,1]*np.max(x[0,:]))/w[0,2]]

    plt.plot(t,z);

    ind=(clas==-1)
    plt.plot(x[0,ind],x[1,ind],"o")

    ind=(clas==1)
    plt.plot(x[0,ind],x[1,ind],"o")

    plt.show()


# Données de test
mean1 = [4, 4]
cov1 = [[1, 0], [0, 1]]  #
data1 = np.transpose(np.random.multivariate_normal(mean1, cov1, 128))
mean2 = [-4, -4]
cov2 = [[4, 0], [0, 4]]  #
data2 = np.transpose(np.random.multivariate_normal(mean2, cov2, 128))
data=np.concatenate((data1, data2), axis=1)

oracle=np.concatenate((np.zeros(128)-1,np.ones(128)))

w,mdiff=apprentissage(data,oracle,1)

plt.plot(mdiff)
plt.show()

affiche_classe(data,oracle,2,w)
```