



























TP1 ML : k-PPV

1. Travaillez sur votre machine ou dans la machine virtuelle : « VMWARE Windows 10 x64 - Développement ».
2. Récupérez les fichiers « iris.data » et « kppv.py » dans « TP1 ML - kPPV.zip » sur Célène.
3. Dans le programme kppv.py, complétez la méthode de calcul de distances qui prend un exemple en entrée (data) et qui retourne les distances euclidiennes avec les exemples de la base d'apprentissage (les 25 premiers exemples de chaque classe).

dataset [nbExParClasse * nbClasse] [nbCaract] :
base d'apprentissage  , base de tests 

	classe 0	classe 1	classe 2
0			
			
			
24			
25			
			
			
49			

4. Ecrire une méthode qui retourne la classe d'un exemple à partir de ses distances. Pour le 1-PPV, cette classe est celle de l'exemple dont la distance est minimale.
5. Faire l'évaluation qui va parcourir les exemples de tests et déterminer leur classe puis va calculer la matrice de confusion (sans classe de rejet) et le taux de reconnaissance.
6. (Facultatif) Quand le programme sera fonctionnel, vous ajouterez un paramètre k pour que le programme calcule la classe en fonction des k plus proches voisins.
7. (Facultatif) Enfin, vous ajouterez la possibilité de faire de la validation croisée pour l'évaluation.