Algorithme des plus proches voisins

September 20, 2026

On dispose d'objets classés en n catégories. A chaque objet est associé un vecteur de $\mathbb{R}^d$.

Le problème est de classifier un nouvel objet connaissant uniquement le vecteur qui lui est associé.

Ces données peuvent être stockées sous forme de dictionnaire. Les clés peuvent être des numéros et les valeurs une liste dont le premier élément est la catégorie de l'objet (sous forme de chaîne de caractères le plus souvent) et le deuxième le vecteur qui lui est associé.

On peut également stocker ces données uniquement à l'aide de listes.

L'idée est de considérer les k plus proches voisins du nouvel objet (dont on ne connaît pas la catégorie) où k est un entier qu'il faudra choisir judicieusement (s'il n'y a que deux catégories, on choisit k impair!).

On peut alors penser que cet objet est de la catégorie majoritaire parmi ces plus proches voisins.

Il faut donc calculer la distance entre l'objet étudié et n'importe qu'elle autre objet. En général, on utilise pour cela la distance euclidienne de $\mathbb{R}^d$~:

$$d((x_1,\dots,x_d),(y_1,\dots,y_d)) = \sqrt{\sum_{i=1}^{d}(x_i-y_i)^2}.$$

On classe les distances obtenue par ordre décroissant et on s'intéresse aux k premières.

Les listes sont stockés dans une liste. Chaque élément de la liste sont eux-même des listes contenant les coordonnées de l'objet et sa catégorie.

Pour les tests on utilisera la liste suivante ($d = n = 2$) :

[3]:
```
Points=[[0,0,'red'],[3,7,'blue'],[4,1,'blue'],[2,7,'red'],[4,4,'red'],
        [5,0,'red'],[7,5,'blue'],[5,6,'blue'],[6,2,'blue'],[7,6,'blue'],
        [2,3,'red'],[1,2,'red'],[1,5,'blue'],[2,5,'blue'],[1,4,'blue'],
        [0,7,'red'],[1,6,'red'],[5,3,'red'],[6,4,'red']]
```

Exercice 1

Ecrire une fonction `affichage` qui représente sur un graphique une liste de points.

[1]:
```
import matplotlib.pyplot as plt

def affichage(Points):
```

[ ]:
```
affichage(points)
```

Exercice 2

Ecrire une fonction `distance` prenant en paramètre deux vecteurs de R^d et qui renvoie la distance euclidienne entre ces deux vecteurs.

[12]:
```
def distance(A,B): # A et B sont deux listes de réels de même taille

```

Exercice 3

Ecrire une fonction `liste_dist` prenant en paramètre un point M de $\mathbb{R}^d$ et une liste `Points`, et qui renvoie une liste `D` contenant la distance de M à chaque objet de la liste `Points` et la couleur de l'objet.

Plus précisément `D[i][0]` est la distance entre l'objet numéro i et le point M, `D[i][1]` est la couleur de l'objet numéro i.

[70]:
```
def liste_dist(M,Points):

```

[ ]:
```
liste_dist((2,3),Points)
```

Exercice 4

1. Ecrire une fonction `max_valeur` qui prend en argument un dictionnaire dont les valeurs sont des entiers positifs et qui renvoie une clé dont la valeur est maximale. Tester cette fonction.
2. Ecrire une fonction `ppv` qui prend en argument un point M de $\mathbb{R}^d$ et un entier k, et qui affiche ce point de la couleur majoritaire de ses k plus proches voisins. Cette fonction doit utiliser la fonction `max_valeur`.

[36]:
```
def max_valeur(dico):

```

[5]:
```
dico={'A':3,'B':5,'E':1,'Z':7,'T':2}
max_valeur(dico)
```

[100]:
```
def ppv(Points,M,k):

```

[ ]:
```
ppv(Points,[3,4],3)
affichage(Points)
```

Exercice 5

Tester la fonction sur tous les points dont les coordonnées sont comprises entre 0 et 10.

[ ]:
```
def ex5(Points,k,N)
```

[ ]:
```
ex5(points,3,5)
```