18

Algorithmique VI - Algorithme des k-plus-proches-voisins (KPP)

Extrait du programme

Thème : Algorithmique

- **Contenus :**
  Algorithme des k plus proches voisins

Capacités attendus :
Écrire un algorithme qui prédit la classe d'un élément en fonction de la classe majoritaire de ses k plus proches voisins.

Commentaires :
Il s'agit d'un exemple d'algorithme d'apprentissage.

I. L'algorithme des k-plus-proches-voisins (KPP)

a) Le principe

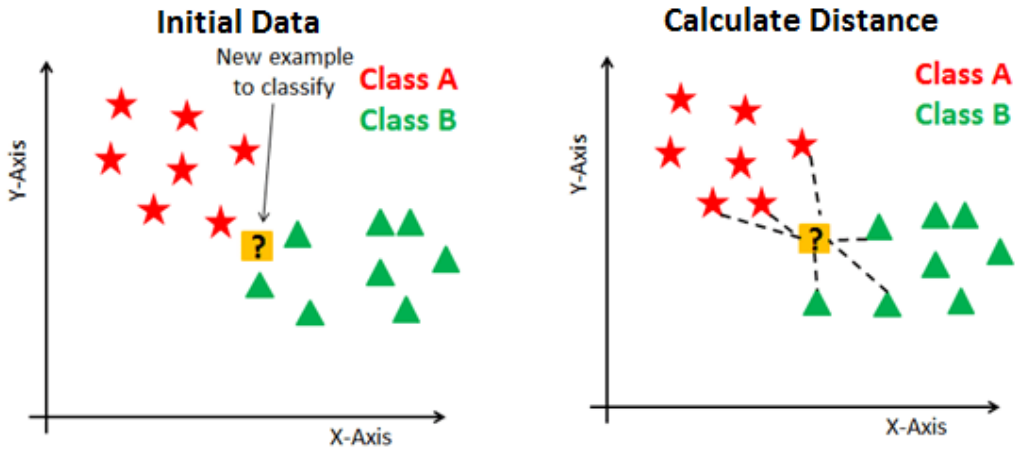
L'algorithme des **k-plus-proches-voisins (KPP)** (ou k-Nearest-Neighbours (KNN)) est une méthode simple et efficace de classification. La classification est un enjeu majeur de l'**Intelligence Artificielle** :

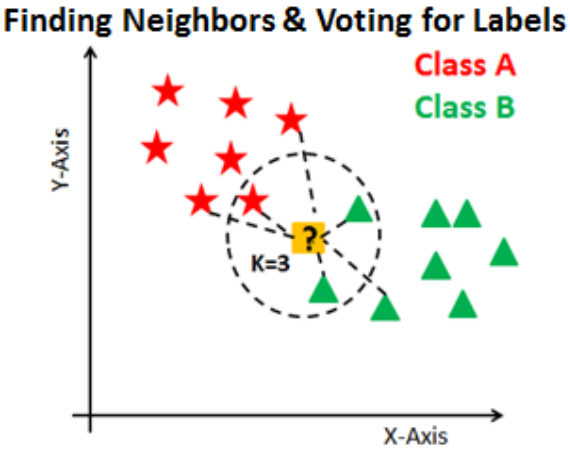
- la caméra d'une voiture autonome perçoit un panneau, mais quel est ce panneau ?
- un grain de beauté est pris en photo par un dermatologue, ce grain de beauté est-il cancéreux ?
- un texte a été écrit à la main, malheureusement certains caractères sont peu lisibles. Est-ce un 'B', un 'R' ou un 'K' ?
- ...

L'algorithme du KPP va trouver quels sont, dans une base de données déjà bien remplie et labellisée, les k-objets (les 3 objets si $k = 3$ par exemple) qui se rapprochent le plus de l'objet à classifier. En prenant ensuite la caractéristique la plus fréquente parmi ces 3 objets, on *devine* alors dans quelle catégorie notre objet doit se classer.

b) Le principe sur un exemple

Considérons une base de donnés d'objets. Certains sont de classe A (étoiles rouge) et d'autres de classe B (triangles verts). Nous avons un nouvel individu que nous cherchons à classer. Est-il de la classe A ou de la classe B ? Nous allons utiliser l'algorithme des KPP [1] pour $k = 3$.
On mesure la distance entre notre individu et tous les éléments de la base de donnée, puis on ne garde que les 3 plus proches. On regarde de quelle classe ils sont.
Nous avons 1 élément de classe A et 2 éléments de classe B. Par proximité avec ces 3 voisins, nous décidons que notre nouvel élément est de la classe B.





1. Le choix de la valeur de k est assez difficile à étudier. Il dépend de nombreux paramètres. Il faut qu'il ne soit pas trop petit, mais pas trop grand non plus. Il dépend aussi du nombre de classes. Ici, comme il y a 2 classes A et B, il faut éviter un nombre k pair. En effet, regardez avec $k = 4$, on obtient 2 éléments proches de la classe A et 2 autres de la classe B. On ne peut pas décider quelle est la classe de notre nouvel élément.

c) Distance

Pour utiliser l'algorithme KPP, nous avons donc besoin d'une distance. La distance la plus commune est la distance que nous utilisons en mathématiques dans un repère orthonormé. On appelle cette distance, la **distance euclidienne** :

$$AB = \sqrt{(x_B - x_A)^2 + (y_B - y_A)^2}$$

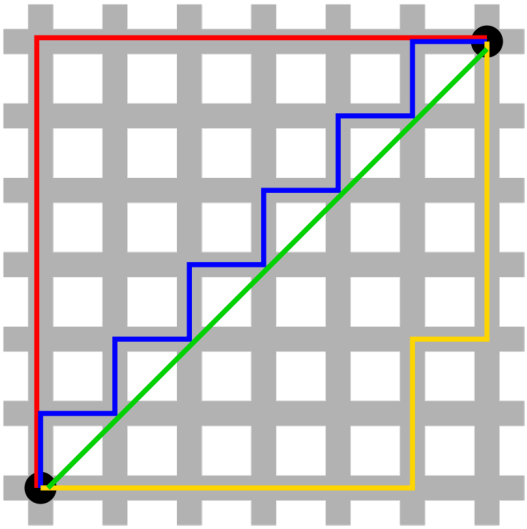
Comme nous avons beaucoup de distances à calculer, il est très intéressant de programmer une fonction pour calculer les distances :

```
def dist(A,B):
    '''
    A[0] est l'abscisse du point
    A[1] est l'ordonnée du point
    '''
    calcul = (B[0]-A[0])**2 + (B[1]-A[1])**2
    calcul = calcul**0.5
    return calcul
```

En réalité, il existe de nombreuses autres distances. Une distance est une application d qui pour deux points A et B d'un ensemble

- est symétrique, autrement dit, $d(A, B) = d(B, A)$;
- vérifie l'axiome de séparation, autrement dit, $d(A, B) = 0$ si et seulement si $A = B$;
- vérifie l'inégalité triangulaire, autrement dit, pour tout troisième point C, $d(A, B) \leq d(A, C) + d(C, B)$.

Un autre exemple de distance est la **distance de Manhattan**. Vous imaginez que vous circulez dans les rues de Manhattan, ville dans laquelle toutes les rues sont parallèles ou perpendiculaires.
Pour aller d'un point A à un point B, vous ne pouvez pas aller en diagonale (distance euclidienne en vert), mais uniquement horizontalement ou verticalement.



Source image : Wikipédia

II. Une utilisation de l'algorithme des KPP

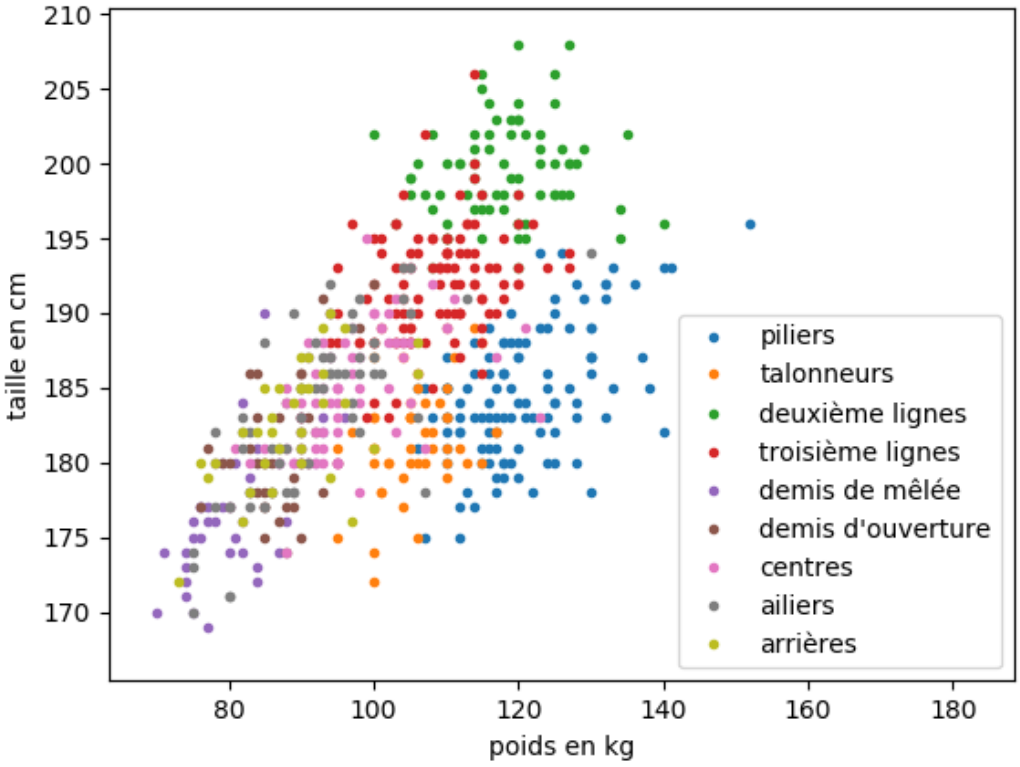
a) Présentation

Nous allons travailler sur une base de données[2] des joueurs du TOP 14 (rugby). Cette base contient pour chaque joueur :

- le nom de l'équipe
- le nom et prénom du joueur
- le poste du joueur
- la date de naissance du joueur
- la taille du joueur en centimètre
- le poids du joueur en kilogramme

	A	B	C	D	E	F
1	Equipe	Nom	Poste	Date de naissance	Taille	Poids
2	Agen	Anton PEIKRISHVILI	Pilier	18/09/87	183	122
3	Agen	Dave RYAN	Pilier	21/04/86	183	116
4	Agen	Giorgi TETRASHVILI	Pilier	31/08/93	177	112

Pour commencer, nous allons étudier cette base de données en affichant un nuage de points[3]. Chaque point représente un joueur avec en abscisse son poids en kg et en ordonnée sa taille en cm. Chaque point a une couleur en fonction du poste du joueur.



En regardant ce graphique on constate que le nuage se découpe en plusieurs catégories assez bien délimitées. Par exemple, les piliers en bleu ou les deuxièmes lignes en verts.

2. Cette base de données a été proposé par Gilles Lassus, professeur de NSI à Bordeaux.
3. Ce graphique peut être obtenu à l'aide du module `matplotlib`. Le code est proposé en annexe.

b) La problématique

Nous sommes entraîneur d'une équipe du TOP 14 et nous venons de recruter un nouveau joueur faisant 93 kg pour 188 cm. Nous voudrions savoir à quel poste nous pourrions le placer. Au regard du graphique, il ne pourra pas être pilier (trop léger) et ne pourra pas être deuxième ligne non plus (trop petit). Nous allons donc utiliser l'algorithme des k-plus-proches-voisins.

c) L'algorithme en action

On peut commencer par afficher la liste des k-plus-proches-voisins :

Rôle :	Écrire la liste des postes des k-plus-proches-voisins.
Entrées :	Un entier *poids* (le poids en kg du joueur à classer), Un entier *taille* (la taille en cm du joueur à classer), Un entier k (le nombre de voisin), Une base de données de joueurs.
Précondition :	Tous les entiers en entrée sont strictement positifs.
Sortie :	Une liste de postes.
Postconditions :	La liste est constituée des k postes des joueurs les plus proches du joueur à classer.

Et voici le code de la fonction `kpp` :

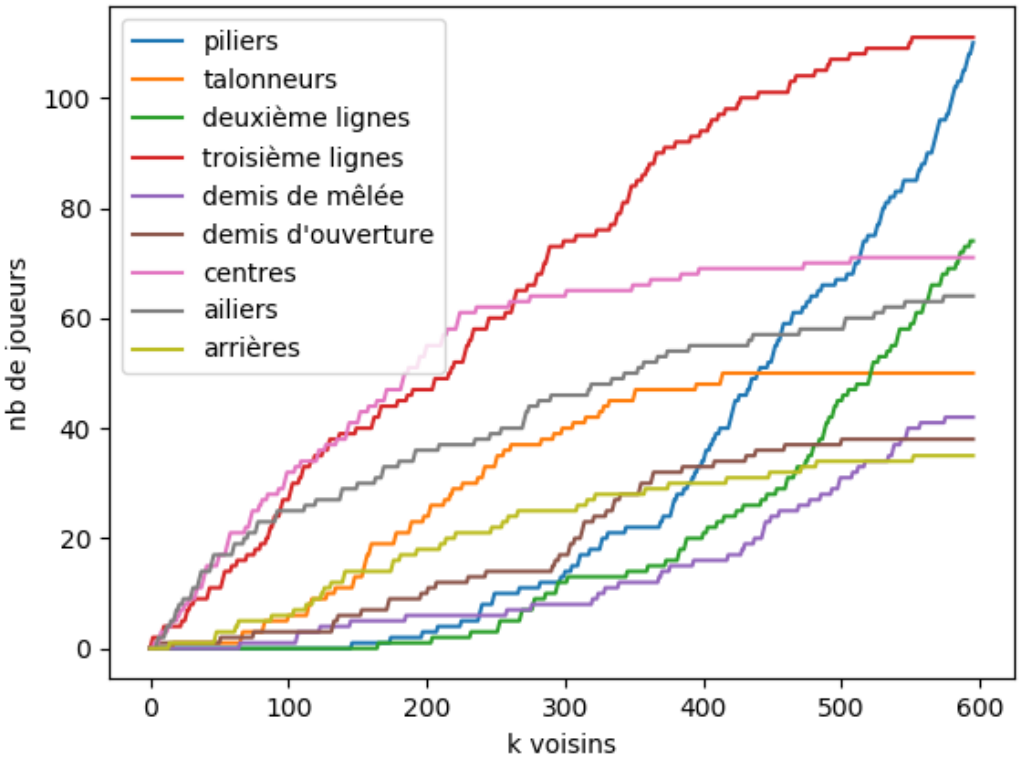
```
def kpp(poids, taille, k):
    #liste des kpp
    voisins = [[100000,"coupeur de citron"] for i in range(k)]
    #l'idée est que cette liste sera toujours ordonnée dans l'ordre croissant
    #donc le dernier élément est à la plus grande distance

    A = [poids, taille]
    for j in joueurs :
        B = [int(j['Poids']),int(j['Taille'])]
        calcul = dist(A,B)
        if calcul < voisins[k-1][0] :
            #j est plus proche que le voisin le plus éloigné
            #il devient un nouveau voisin
            voisins[k-1][0] = calcul
            voisins[k-1][1] = j['Poste']
        #on reordonne la liste des voisins
        voisins.sort()
    return [v[1] for v in voisins]
```

La réponse pour $k = 6$ est : `['Centre', '3ème ligne', 'Ailier', 'Ailier', 'Arrière', 'Ailier']`.

Sans hésiter notre joueur sera Ailier (3 votes sur 6).

d) Influence du paramètre k

Si nous comptabilisons [4] le nombre de vote pour chacun des postes Pilier, Talonneur, 2ème ligne, 3ème ligne, Mêlée, Ouverture, Centre, Ailier, Arrière en fonction du paramètre k, nous obtenons le graphique suivant :



Comme nous voulons être certain du poste que nous allons attribuer à notre joueur faisant 93 kg pour 188 cm, nous décidons, par sécurité de prendre le plus grand nombre de voisins. En regardant le graphique, nous décidons que notre joueur sera soit un 3ème ligne, soit un pilier.
Expliquez pourquoi nous commettons une grossière erreur ?

.....
.....
.....
.....
.....

Finalement, il nous manque un talonneur. Nous aurions bien voulu le placer à ce poste.
Expliquez pourquoi la courbe correspondant aux talonneurs reste constante après 400 voisins ?

.....
.....
.....
.....
.....

Notre nouveau joueur nous affirme qu'il est de formation trois-quart. Autrement dit, il ne peut jouer que centre, ailier ou arrière. Modifier le programme pour rechercher les KPP uniquement parmi les postes qui conviennent.

4. Le code complet en annexe.

A

Code pour tracer le nuage de points des joueurs du TOP 14

```
import codecs #pour lire en UTF8, n'est pas toujours nécessaire
import csv
import matplotlib.pyplot as plt     #pour faire le graphique

joueurs = []

with codecs.open('top14.csv', 'r', 'utf-8') as fichier:
    csvtop14 = csv.DictReader(fichier, delimiter=',')
    for ligne in csvtop14:
        joueurs.append(dict(ligne))

#pour vérification on affiche le premier pays
print(joueurs[2])

#fonction pour récupérer les couples poids taille par poste
def filtre_poste(leposte):
    T = [(j['Poids'],j['Taille']) for j in joueurs if j['Poste']== leposte]
    return T

piliers = filtre_poste('Pilier')
talonneurs = filtre_poste('Talonneur')
deuxiemel = filtre_poste('2ème ligne')
troisiemel = filtre_poste('3ème ligne')
demele = filtre_poste('Mêlée')
deouv = filtre_poste('Ouverture')
centres = filtre_poste('Centre')
ailiers = filtre_poste('Ailier')
arrieres = filtre_poste('Arrière')

#fonction pour récupérer la liste des poids
def poids(liste):
    return [int(j[0]) for j in liste]

#fonction pour récupérer la liste des taille
def taille(liste):
    return [int(j[1]) for j in liste]

#on trace le graphique avec sa légende
plt.axis([65, 190, 165, 210])
plt.plot(poids(piliers),taille(piliers),'.',label='piliers')
plt.plot(poids(talonneurs), taille(talonneurs),'.',label='talonneurs')
plt.plot(poids(deuxiemel), taille(deuxiemel),'.',label='deuxième lignes')
```

```
43 plt.plot(poids(troisiemel), taille(troisiemel),'.',label='troisième lignes')
44 plt.plot(poids(demele), taille(demele),'.',label='demis de mêlée')
45 plt.plot(poids(deouv), taille(deouv),'.',label="demis d'ouverture")
46 plt.plot(poids(centres), taille(centres),'.',label="centres")
47 plt.plot(poids(ailiers), taille(ailiers),'.',label="ailiers")
48 plt.plot(poids(arrieres), taille(arrieres),'.',label="arrières")
49 plt.xlabel("poids en kg")
50 plt.ylabel("taille en cm")
51 plt.legend(loc='lower right')
52 plt.show()
```

B

Code pour tracer les nombres de vote en fonction du nombre de voisins

```
import codecs #pour lire en UTF8, n'est pas toujours nécessaire
import csv
import matplotlib.pyplot as plt     #pour faire le graphique

joueurs = []

with codecs.open('top14.csv', 'r', 'utf-8') as fichier:
    csvtop14 = csv.DictReader(fichier, delimiter=',')
    for ligne in csvtop14:
        joueurs.append(dict(ligne))

#pour vérification on affiche le premier joueur
print(joueurs[2])

#fonction distance
def dist(A,B):
    '''
    A[0] est l'abscisse du point
    A[1] est l'ordonnée du point
    '''
    calcul = (B[0]-A[0])**2 + (B[1]-A[1])**2
    calcul = calcul**0.5
    return calcul

def kpp(poids, taille, k):
    #liste des kpp
    voisins = [[100000,"coupeur de citron"] for i in range(k)]
    #l'idée est que cette liste sera toujours ordonnée dans l'ordre croissant
    #donc le dernier élément est à la plus grande distance

    A = [poids, taille]
    for j in joueurs :
        B = [int(j['Poids']),int(j['Taille'])]
        calcul = dist(A,B)
        if calcul < voisins[k-1][0] :
            #j est plus proche que le voisin le plus éloigné
            #il devient un nouveau voisin
            voisins[k-1][0] = calcul
            voisins[k-1][1] = j['Poste']
```

```
        #on reordonne la liste des voisins
        voisins.sort()
    return [v[1] for v in voisins]

#les postes dans l'ordre sont Pilier, Talonneur, 2ème ligne, 3ème ligne, Mêlée, Ouverture, Centre, Ailier, Arrière
def vote(voisins):
    postes = ['Pilier', 'Talonneur', '2ème ligne', '3ème ligne', 'Mêlée', 'Ouverture', 'Centre', 'Ailier', 'Arrière']
    nb_poste = [0]*9
    k = len(voisins)
    for v in voisins:
        for p in range(9):
            if v == postes[p]:
                nb_poste[p] = nb_poste[p] + 1
    return nb_poste


#on récupère les pourcentages de vote par poste en fonction de k
#nombre total de joueurs
nb_tot = len(joueurs)
abscisse_k = [k for k in range(nb_tot+1)]
piliers = [0]*(nb_tot+1)
talonneurs = [0]*(nb_tot+1)
deuxieme = [0]*(nb_tot+1)
troisieme = [0]*(nb_tot+1)
demele = [0]*(nb_tot+1)
deouv = [0]*(nb_tot+1)
centres = [0]*(nb_tot+1)
ailiers = [0]*(nb_tot+1)
arrieres = [0]*(nb_tot+1)
for k in range(1,nb_tot+1):
    print(k)
    votek = vote(kpp(98,188,k))
    piliers[k] = votek[0]
    talonneurs[k] = votek[1]
    deuxieme[k] = votek[2]
    troisieme[k] = votek[3]
    demele[k] = votek[4]
    deouv[k] = votek[5]
    centres[k] = votek[6]
    ailiers[k] = votek[7]
    arrieres[k] = votek[8]


#on trace le graphique avec sa légende
#plt.axis([65, 190, 165, 210])
plt.plot(abscisse_k,piliers,label='piliers')
plt.plot(abscisse_k,talonneurs,label='talonneurs')
plt.plot(abscisse_k,deuxieme,label='deuxième lignes')
plt.plot(abscisse_k,troisieme,label='troisième lignes')
plt.plot(abscisse_k,demele,label='demis de mêlée')
plt.plot(abscisse_k,deouv,label="demis d'ouverture")
plt.plot(abscisse_k,centres,label="centres")
plt.plot(abscisse_k,ailiers,label="ailiers")
plt.plot(abscisse_k,arrieres,label="arrières")
plt.xlabel("k voisins")
plt.ylabel("nb de joueurs")
plt.legend(loc='upper left')
plt.show()
```

C

Une dernière blague, pour qui peut comprendre