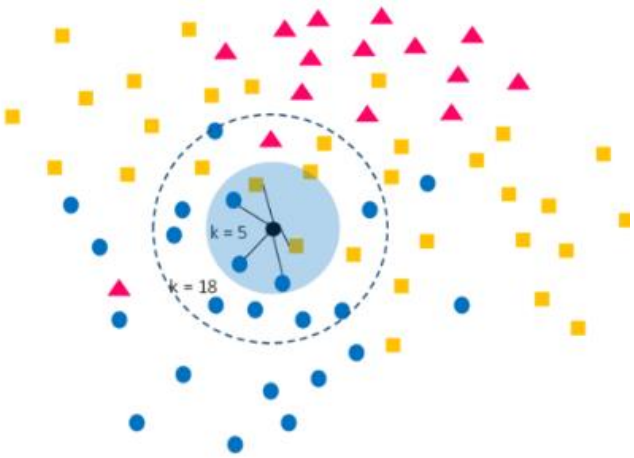
Algorithme des k plus proches voisins (k-Nearest Neighbors : K-NN)

D'après : https://isn-icn-ljm.pagesperso-orange.fr/NSI/co/grain_algo_KNN.html

Dans le domaine de l'intelligence artificielle, on utilise des algorithmes d'apprentissage. Par exemple certains algorithmes permettent de faire une prévision sur une caractéristique d'un élément en fonction de l'étude de ses k plus proches voisins.



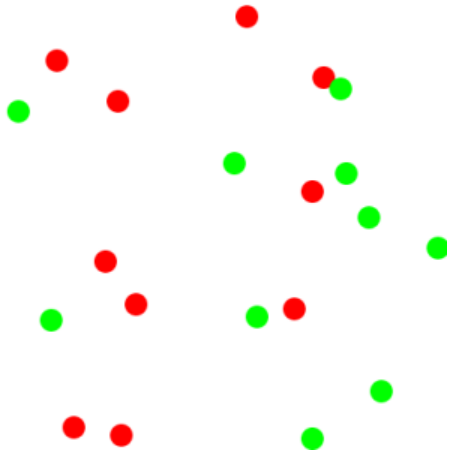
On dispose d'un ensemble de données (entrées) et l'on s'intéresse plus particulièrement à l'une d'entre elles (sortie).

On ajoute une donnée et on lui donnera en sortie la valeur majoritaire des sorties de ses k plus proches voisins.

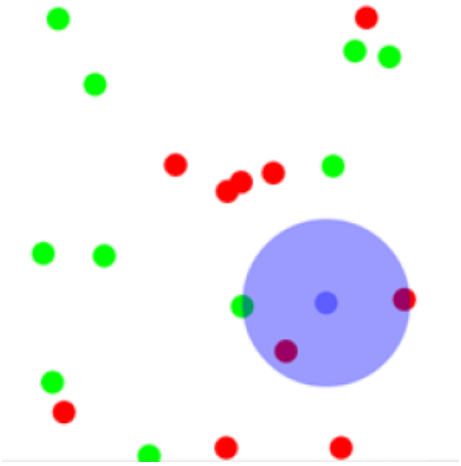
Exemple-1 : Visualiser la vidéo 01-k_plus_proches_voisins

Exemple-2 :

On dispose de deux populations dont on connaît la localisation (coordonnées) et une caractéristique (Rouge ou Verte).



On rajoute un élément en précisant ses coordonnées et on lui impose la caractéristique (Rouge ou Verte) majoritaire parmi ses k plus proches voisins.



Dans cet exemple (k=3) : Le nouvel élément sera Rouge, car parmi ses 3 plus proches voisins, deux sont rouges.

Le calcul de distance utilisé ici est la distance euclidienne, mais il existe bien entendu d'autres types de « distances » selon le problème étudié...

Exemple-3 : Simulation.

Considérons le jeu de données suivant :

Nom	Age	Revenus (k€)	Nombre d'achats	Fidélité
Jean	35	36	3	N
Louis	22	50	2	O
Anne	63	200	1	N
Suzanne	59	170	1	N
Nicolas	25	40	4	O

Utiliser un éditeur de texte pour créer un fichier nommé « **clients.csv** » qui regroupe les données du tableau ci-dessus, avec des virgules comme séparateur.

Un nouveau client se présente et on souhaite estimer sa fidélité…

Nom	Age	Revenus (k€)	Nombre d'achats	Fidélité
David	37	50	2	?

Pour cela, on utilise un algorithme K-NN (algorithme des k plus proches voisins)

On calcule la « distance » de David avec chacun des autres clients

Remarquez ici que le terme « distance » indique juste une comparaison entre les caractéristiques du nouveau client (David) et les caractéristiques des autres clients.

Même si on calcule une distance euclidienne entre deux points.
$d = \sqrt{(xB - xA)^2 + (yB - yA)^2 + (zB - zA)^2 + \cdots}$

Par exemple : distance (David, Jean)=$\sqrt{(37 - 35)^2 + (50 - 36)^2 + (2 - 3)^2} = 14.177$
Et ainsi de suite. On obtient les distances suivantes :

[('Jean', 14.17744688), ('Louis', 15.0), ('Anne', 152.2399422), ('Suzanne', 122.0040983), Nicolas', 15.74801575)]

Les 3 plus proches sont : Jean, Louis et Nicolas.
On en conclut que la fidélité de David est : Oui

Voici le programme écrit en Python (exemple-simulation.py)

Ecrivez-le dans la console « Python » et faites-le fonctionner. (Mais il faut le comprendre… donc y ajouter le maximum de commentaires !)

```
# K-NN algorithme : estimation de la fidelite d'un client,
#a partir de ses k plus proches voisins

import csv # bibliotheque csv
from math import* # pour utiliser la racine carree

############################################################
# Ouverture du fichier csv et stockage dans une liste

with open('clients.csv',newline="") as fichier:
    lecteurCSV=csv.reader(fichier,delimiter=",")
    tableau=[]
    for ligne in lecteurCSV:
        #print(ligne)#  pour verification
        tableau.append(ligne)
fichier.close()
print(tableau)#pour verification

############################################################
# Donnees du nouveau client

nom_nc='David'
age_nc=37
revenu_nc=50
achat_nc=2
```

Modifiez ce programme pour que ce soit l'utilisateur qui entre les données d'un nouveau client.

Exemple-4 : A vous de jouer !
Votre travail : Imaginez un jeu de données similaire et créer un programme qui estime un caractère en fonction de ses k plus proches voisins.

Exemple-5 : Couleur d'un point

On cherche à mettre en application l'exemple 2 vu au-dessus.

1. Créer un programme python nommé « **exemple-4_NOM_Prenom** » dans lequel :
   - Vous importerez la bibliothèque « randint » :
     from random import randint
   - Puis une liste de couleurs possibles, nous nous limiterons à rouge et bleu : couleur = ["r", "b"]

   Créer ensuite une fonction « points(n,c) » qui retourne une liste de n points caractérisés par leurs coordonnées x et y (telles que les coordonnées x et y soient comprises entre -c et c), et une couleur cL (rouge ou bleue)

   Voilà une exemple de résultat :

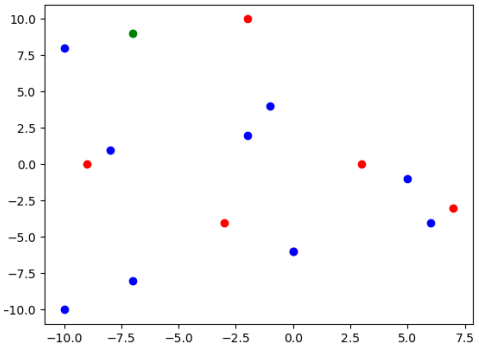
```
>>> points(15, 10)
[[1, -1, 'r'], [-7, 4, 'r'], [10, -6, 'b'], [3, -2, 'r'], [8, -2, 'b'], [-7, 1, 'r']
, [6, 2, 'b'], [7, 7, 'b'], [2, 1, 'r'], [-2, 10, 'b'], [-2, -6, 'r'], [-6, -1, 'r']
, [-9, 3, 'b'], [-9, 1, 'r'], [6, 5, 'r']]
```

2. Création d'un graphe comportant une série de points rouges, bleus et le point vert ("g" pour green) que l'on cherchera à placer dans la catégorie bleue ou rouge en fonction de ses k plus proches voisins.

```
import matplotlib.pyplot as plt

dim = 10
nbpoints = 15

point_vert = (randint(-dim, dim), randint(-dim, dim), "g")
print(point_vert)
pts = points(nbpoints, dim)
print(pts)

for u in pts:
    plt.plot(u[0], u[1], u[2]+"o")
plt.plot(point_vert[0], point_vert[1], point_vert[2]+"o")

plt.show()
```

Reportez-vous à l' « activité-1_Decouverte_Matplolib » pour utiliser quelques commandes de tracé de points sur un graphique.

Voici un exemple de résultat :



3. Calcul des distances euclidiennes entre le point vert et les points bleus et rouges.

Importer le module « sqrt » de la bibliothèque math en haut du programme:
from math import sqrt

Créer une liste ptsA qui regroupe les coordonnées des points rouges et bleus. Il s'agit tout simplement de la liste « pts » précédente.

Créer une liste ptsB = [xB, yB, cL], qui regroupe les coordonnées du point vert que l'on cherche à classer dans la catégorie bleue ou rouge.

Créer une fonction « distance(ptsA, ptsB) » qui calcule et retourne dans une liste « distance_AB » les distances euclidiennes entre le point B et les points A rouges et bleus.

Vous utiliserez la commande « round (valeur, n) avec valeur = distance euclidienne calculée, et n = 2 le nombre de décimales maximales pour la distance euclidienne.

```
def distance(ptsA, ptsB):
    distances_AB = []
    for i in range(0, len(ptsA)):
        distance = sqrt((ptsA[i][0] - ptsB[0])**2 + (ptsA[i][1] - ptsB[1])**2)
        distance = round(distance, 2)
        distances_AB.append(distance)
    #print(distances_AB)
```

Compléter la fonction afin que cette distance calculée soit ajoutée dans la liste « ptsA ».

```
for i in range(0, len(ptsA)):
    for j in range(0, len(ptsA[i])):
        ptsA[i].append(distances_AB[i])
```

Afin de classer les éléments de la liste par ordre croissant des distances euclidiennes, nous allons amener cette valeur en position 0.
Puis utiliser la commande « sorted » pour classer la liste par ordre croissant des distances.

```
for i in range(len(ptsA)):
    ptsA[i][0], ptsA[i][3] = ptsA[i][3], ptsA[i][0]
#print(ptsA)

ptsA_classees = sorted(ptsA)
```

4. Compléter le programme pour qu'il crée une liste des k plus proches voisins.

```
ptsA_classees = distance(ptsA, ptsB)

k = int(input("saisir une valeur pour k: "))

proches_voisins = []
for i in range(0, (k)):
    proches_voisins.append(ptsA_classees[i])
print(proches_voisins)
```

5. Terminer le programme en ajoutant un compteur qui va compter les rouges et les verts dans la liste « proches_voisins », puis qui va attribuer la bonne couleur au point vert, en fonction de ses k plus proches voisins.

```
cpt = [0, 0]
for v in proches_voisins:
    if v[2] =="r":
        cpt[0] = cpt[0] + 1
    else:
        cpt[1] = cpt[1] +1

i = cpt.index(max(cpt)) #l'indice de la couleur la plus représentée
#print(i)
choix = couleur[i] # la couleur la plus représentée#print(choix)
#print(choix)

if choix == 'r':
    print('le point appartient à la catégorie rouge')
else:
    print('le point appartient à la catégorie bleue')
```