Lycée Jean Moulin **Draguignan**
NSI 1re Année .....

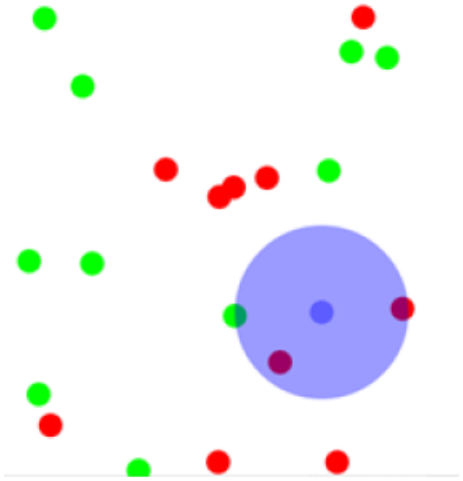
Algorithme des k plus proches voisins

La méthode

On commencera par regarder cette vidéo, qui présente la méthode du KNN (k-nearest neighbours).
Lien vers la vidéo

Le principe : On dispose de données, par exemple des points repérés par leurs coordonnées qui sont colorés soit en rouge soit en vert.
On rajoute un point dans cette colonie, on lui attribue une couleur en fonction de ses k plus proches voisins.

Dans l'exemple ci-contre le nouveau point sera rouge, car parmi ses $k = 3$ plus proches voisins, deux sont rouges.



*Réalisation de l'exemple dans **Processing***

Étape 1 : Affichons dans une fenêtre notre colonie de rouge et de vert

Dans Processing toute variable déclarée en dehors de fonctions est considérée comme globale.
La fonction `setup()` permet d'initialiser le programme.
- Elle initialise une fenêtre de 400 × 400 pixels.
- Sur un fond blanc.
- Et remplit les deux listes R et V avec des coordonnées entières de points (aléatoirement choisis dans des zones différentes) sous forme de tuples.
- R1 et V1 sont les listes qui contiendront les nouveaux points

```
# Algorithme K-NN
K=3
R=[]
V=[]
R1=[]
V1=[]
def setup():
    size(400,400)
    background(255)
    for i in range(0,50,1):
        x = int(random(400))
        y = int(random(400))
        if 100 < x < 300 and 140 < y < 350:
            R.append((x,y))
        else:
            V.append((x, y))
```

La fonction `draw()` est spécifique à Processing, elle s'exécute 60 fois par seconde, ce qui permet d'obtenir des animations dans la fenêtre.

- Elle repeint la fenêtre à chaque tour .
- Elle affiche notre colonie avec les bonnes couleurs.
- Les nouveaux points seront affichés avec de la transparence, pour ne pas les confondre avec les points de bases

```
def draw():
    background(128)
    noStroke()
    # affichage des colonies
    fill(255,0,0)
    for point in R:
        ellipse(point[0],point[1],20,20)
    fill(255,0,0,60)
    for point in R1:
        ellipse(point[0],point[1],20,20)
    fill(0,255,0)
    for point in V:
        ellipse(point[0],point[1],20,20)
    fill(0,255,0,60)
    for point in V1:
        ellipse(point[0],point[1],20,20)
```

Pour info :
`ellipse(absi du centre, ordo du centre, diamètre horizontal, diamètre vertical)`

? Question 1:

Expliquer le fonctionnement de cette boucle

```
for point in R:
    ellipse(point[0],point[1],20,20)
```

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

Étape 2 :
Affichons un cercle centré sur la position de la souris et qui touche les K = 3 plus proches voisins.
Rajouter le code suivant dans la fonction `draw()`

```
    rayon=[]
    # calcul des distances et stockage dans une liste
    for point in R:
        rayon.append(dist(mouseX,mouseY,point[0],point[1]))
    for point in V:
        rayon.append(dist(mouseX,mouseY,point[0],point[1]))
    #tri de la liste
    rayon.sort()
    # affichage d'un cercle "contenant les k plus proches voisins"
    fill(0,0,255,100)
    ellipse(mouseX,mouseY,2*rayon[K-1],2*rayon[K-1])
```

? Question 2:

Expliquer pourquoi : `rayon[K-1]`

.....

.....

Étape 3 :
Affichons un nouvel élément sur la position de la souris et décidons de sa couleur.

Affichage du nouvel élément : Ajouter ce code dans la fonction `draw()`.

```
    # affichage du nouvel élément
    noFill()
    stroke(0)
    ellipse(mouseX,mouseY,20,20)
```

Décision :
Processing possède des fonctions spécifiques sur l'utilisation de la souris, Voici le code qui permet au clic de souris de placer un nouvel individu dans la population et de décider de sa couleur.
Votre travail : Le comprendre, l'implémenter.

```
def mouseClicked():
    # Quand on clique sur la souris on stocke la position de la souris
    x=mouseX
    y=mouseY
    countR=0
    countV=0
    # Si on clique dans la fenêtre
    if x>0 and x<400 and y>0 and y<400:
        D=[]
    # On calcule la distance(Euclidienne) du nouvel élément avec tous les autres
    # On stocke les résultats dans une liste de tuples (distance, couleur)
        for point in R:
            D.append((dist(x,y,point[0],point[1]),"R"))
        for point in V:
            D.append((dist(x,y,point[0],point[1]),"V"))
    # On trie cette liste suivant la première valeur du tuple
        D.sort(key=lambda x:x[0])
    # On incrémente des compteurs suivant le cas
        for i in range(0,K,1):
            if D[i][1]=="R":
                countR=countR+1
            else:
                countV=countV+1
    # On décide de la couleur du nouvel élément
    # On le rajoute à la bonne liste
        if countR>countV:
            R1.append((x,y))
        if countV>countR:
            V1.append((x,y))
```

À faire 1:

- Écrire le programme .
- Tester le programme.
- Modifier la valeur de K.
- Modifier le nombre d'individus.
- Rajouter une troisième population.

Un second exemple

Considérons le jeu de données suivant :

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

David	37	50	2	?

Un nouveau client se présente et on souhaite estimer sa fidélité...
Pour cela, on utilise un algorithme KNN.

On calcule la distance de David avec chacun des autres clients.

Par exemple : d(David, Jean) = $\sqrt{(37-35)^2+(50-35)^2+(2-3)^2} = 15{,}165$
On fait de même pour les autres...
On obtient :

```
[('Louis', 15.0), ('Jean', 15.165750503540039),
('Nicolas', 15.748015403747559),
('Suzanne', 122.00409698486328), ('Anne', 152.2399444580078)]
```

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

À faire 2:

- Récupérer l'archive KNN_CSV.zip , décompresser là
- Rajouter des individus de la classe 0 dans le fichier csv
- Exécuter plusieurs fois le programme en modifiant le nouveau client à chaque fois (de classe 1).
- Modifier le programme pour que ce soit l'utilisateur qui entre les données d'un nouveau client.

À FAIRE 3:

Expliquer ce bout de code :

```
k=3
Co=0
Cn=0
for i in range(0,k,1):
    if liste[i][2]=='O':
        Co=Co+1
    if liste[i][2]=='N':
        Cn=Cn+1
if Co>Cn:
    F='O'
if Cn>Co:
    F='N'
```

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

À FAIRE 4:

- Reprendre le "À faire 3" en modifiant dans le programme la distance utilisée par celle - ci : $|age1 - age2| + |revenus1 - revenus2| + |nb\_achat1 - nb\_achat2|$.
- Que peut-on en conclure ?

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

Un prolongement possible...

Imaginez un jeu de données similaire et créer un programme qui estime un caractère en fonction de ses k plus proches voisins.
Ce travail peut être développé pour un projet