Chapitre 6

ALGORITHME DES k PLUS PROCHES VOISINS

Dans les journaux et romans de science, le terme « Intelligence Artificielle » est assez galvaudé et probablement mal utilisé, il nourrit beaucoup de fantasmes. Dans ce chapitre, le terme d'algorithme d'apprentissage désigne le fait d'essayer de faire "apprendre" quelque chose à l'ordinateur, comme reconnaitre des lettres ou des chiffres, des sons ou des éléments sur une image, mais il ne s'agit pas de croire que l'ordinateur pense par lui-même !

Ce chapitre donne des idées simples mais efficaces pour aborder les plus faciles des problèmes cités ci-dessus. Il s'agit pour l'ordinateur de regrouper des données par similitude et y coller une étiquette qui ait du sens. On aura éventuellement besoin d'un jeu de données de référence. L'algorithme des k-plus proches voisins (algorithme KNN) permet répondre partiellement à ce type de questions.

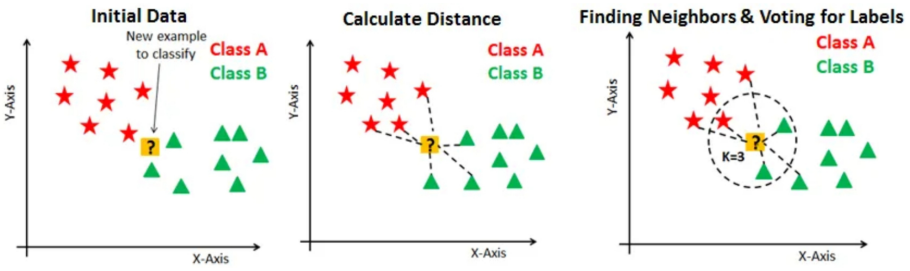


Table des matières

6 ALGORITHME DES k PLUS PROCHES VOISINS **1**

I INTELLIGENCE ARTIFICIELLE ET APPRENTISSAGE 2

II ALGORITHME DES k PLUS PROCHES VOISINS 3

II.1 JEU DE DONNÉES DE RÉFÉRENCE 3

II.2 DISTANCE(S) ENTRE DEUX DONNÉES 4

II.3 RECHERCHE DE LA RÉFÉRENCE MAJORITAIRE 5

II.4 ALGORITHME DES k PLUS PROCHES VOISINS 5

II.5 MATRICE DE CONFUSION 6

II.6 QUELQUES REFLEXIONS 6

III EXERCICES 7

I INTELLIGENCE ARTIFICIELLE ET APPRENTISSAGE

L'intelligence artificielle (IA) est un ensemble de techniques qui permettent à des machines d'accomplir des tâches qui semblent nécessiter de l'intelligence. Par exemple jouer à un jeu comme les échecs, traduire un texte, ou être capable "d'apprendre", c'est à dire de s'améliorer avec de l'entraînement.

L'apprentissage automatique (en anglais : machine learning) est un domaine de l'intelligence artificielle, qui permet de donner aux ordinateurs la capacité "d'apprendre" à partir de données, c'est-à-dire d'améliorer leurs performances à résoudre des tâches (reconnaître des chiffres ou un chat sur une image, distinguer des types de sons) sans être explicitement programmés pour chacune.

Apprentissage supervisé : on dispose d'un **ensemble d'apprentissage**, qui est un ensemble de données sur lequelles on connait la réponse à notre problème. On s'en sert pour prédire la réponse pour de nouvelles données. Ex : On a des photos de chats, de chiens et de chevaux, identifiées comme telles. On reçoit une nouvelle photo d'animal, et on doit déterminer s'il s'agit d'un chat, d'un chien ou d'un cheval.
Note : l'ensemble d'apprentissage peut parfois être appelé ensemble d'entraînement (car en anglais on dit training set), ou jeu de données de référence.

Apprentissage non supervisé : On cherche à structurer les données en groupes de données similaires, sans savoir d'avance quels seront les types de groupes. Ex : une plateforme de films veut regrouper ses utilisateurs selon la similiarité de leur profil (intérêt : leur faire des recommandations de films qu'ils ont de grandes chances d'aimer, car les personnes qui leur sont similaires ont aimé ces films).

Problèmes de classement/étiquetage (en anglais : classification) : on a un ensemble fini de classes/étiquettes (ex : chat, chien, cheval), et on veut déterminer quelle est la classe/étiquette d'un élément pris en entrée. C'est en général de l'apprentissage supervisé : on a un ensemble de données dont on connait la classe/étiquette, et on se sert de cet ensemble d'apprentissage pour déterminer la classe/étiquette d'une nouvelle donnée

Algorithme des K plus proches voisins (ou KNN pour k-Nearest-Neighbor) : algorithme d'apprentissage supervisé qui permet de résoudre des problèmes de classification.

Principe de l'algorithme : On a une notion de distance (ou de similiarité) sur nos données. Etant donnés un entier k, l'ensemble d'apprentissage connu, et une nouvelle donnée à classer :

— on détermine les k éléments de l'ensemble d'apprentissage les plus proches de la nouvelle donnée

— on attribue à la donnée à classer la classe majoritaire parmi ses k plus proches voisins.

La sortie de l'algorithme n'est pas forcément une réponse exacte : il peut y avoir des éléments dont la classse réelle n'est pas la classe majoritaire parmi ses k plus proches voisins. De plus la sortie de l'algorithme dépend de la valeur de k, et aussi de l'ensemble d'apprentissage. On peut chercher à améliorer les performances en ayant un meilleur ensemble d'apprentissage (quantité mais aussi qualité des données), ou en lançant l'algorithme avec différentes valeurs de k. De plus l'algorithme peut être plus ou moins adapté à différents types de problèmes. Par exemple, si on veut déterminer la nationalité d'une personne connaissant sa date de naissance, sa taille et son poids, les résultats seront probablement très mauvais même avec beaucoup de données dans l'ensemble d'apprentissage, car pas vraiment de corrélation entre ce qu'on veut déterminer et les autres données. Déterminer la nationalité connaissant le nom, le prénom et la ville de résidence donnerait probablement de bien meilleurs résultats...

Pour évaluer l'algorithme : On fait un jeu de test T (données pour lesquelles on a la réponse à notre problème mais qui ne sont pas dans l'ensemble d'apprentissage). On lance l'algorithme sur chaque élément de T. Chaque élément t de T a une classe prédite par l'algorithme, qu'on compare à sa classe réelle.

Matrice de confusion : Matrice carrée M de taille $n \times n$ où n est le nombre de classes, et $M_{i,j}$ est le nombre d'éléments de classe i prédits comme étant de classe j. Ainsi la trace de M est le nombre de bons résultats, et la somme des coefficients de M est le nombre de tests. On peut analyser plus finement les lignes et les colonnes pour savoir quelles classes sont bien prédites et quelles classes ne sont pas bien prédites.

II ALGORITHME DES k PLUS PROCHES VOISINS

Dans cette partie, nous allons voir en détail comment implémenter l'algorithme des k plus proches voisins, en prenant l'exemple du problème de la reconnaissance automatique de chiffres écrits à la main et numérisés.

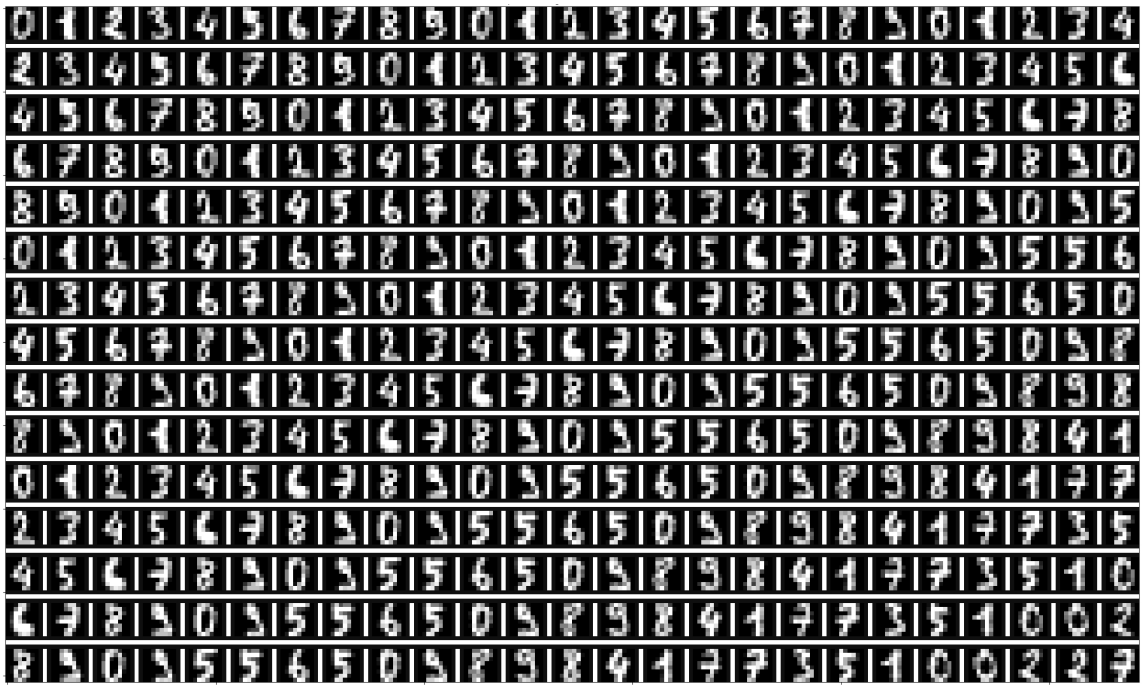
II.1 JEU DE DONNÉES DE RÉFÉRENCE

Nous allons utiliser les données issues d'une bibliothèque Python nommée `sklearn` dédié à l'intelligence artificielle. Avec les commandes ci-dessous :

```
from sklearn import datasets # Chargement des jeux de données

digits=datasets.load_digits() # Jeu de données des chiffres
# Conversion en liste de listes des images
Images=[digits.images[k].tolist() for k in range(len(digits.images))]
Numeros=list(digits.target) # listes des "étiquettes" des images
```

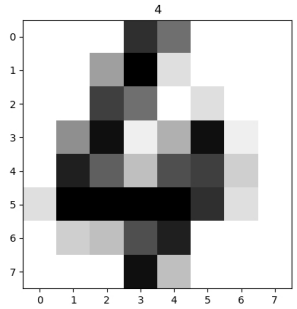
Nous disposons d'une liste d'images nommée `Images` donc voici un extrait :



Extrait de 15×25 **images des** 1797 **images du jeu de données**

Chaque image représente un chiffre manuscrit. De plus, pour chaque image, le chiffre manuscrit a été indiqué manuellement (par un humain) dans la liste `Numeros`. Par exemple, si on fait :

```
plt.figure()
#Affichage de l'image d'indice 13
plt.imshow(Images[13], cmap='binary')
#titre de l'image: son numéro
plt.title(Numeros[13])
plt.show()
```



Une image d'étiquette 4

Pour l'algorithme KNN, on aura toujours un **jeu de données** avec N éléments classifiés par une **étiquette**.

- Ici chaque donnée est une image 8×8 pixels en niveau de gris (entier entre 0 et 255).
- Ici l'étiquette est le chiffre représenté par l'image.
- On souhaite deviner l'étiquette d'une image n'appartenant pas à l'ensemble des données.
- Il va falloir dans un premier temps définir une distance entre deux images afin que l'ordinateur puisse les comparer et « quantifier leurs différences ».

II.2 DISTANCE(S) ENTRE DEUX DONNÉES

Les mathématiques permettent de définir facilement plusieurs notions de distance entre deux éléments :

- Si $x, y \in \mathbb{R}$ ou $\mathbb{C}$, alors $|x-y|$ représente la distance entre x et y.
- Si on a $x = (x_1, \dots, x_n) \in \mathbb{R}^n$ et $y = (y_1, \dots, y_n) \in \mathbb{R}^n$, alors $\boxed{d(x,y) = \sqrt{\sum_{k=1}^{n} (x_k - y_k)^2}}$.

  (En fait $d(x,y) = \|x-y\| = \sqrt{(x-y|x-y)}$ où $(x|y)$ est le produit scalaire canonique de $\mathbb{R}^n$...)
- pour 2 fonctions : $f, g \in \mathcal{C}^0([a,b], \mathbb{R})$, alors $\boxed{d_1(f,g) = \sqrt{(f-g|f-g)}}$, où $(f|g) = \displaystyle\int_a^b f(x)g(x)\,\mathrm{d}x$.

  Mais $\boxed{d_2(f,g) = \|f-g\|_\infty}$ est une autre possibilité ! (voir chapitre sur les e.v.n.)
- pour 2 matrices $A, B \in \mathcal{M}_{n,p}(\mathbb{R})$ alors $\boxed{d(A,B) = \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{p} (a_{i,j} - b_{i,j})^2}}$.

  (En fait $d(A,B) = \sqrt{(A-B|A-B)}$ avec $(A|B) = \mathrm{tr}(A^T B)$, produit scalaire canonique de $\mathcal{M}_{n,p}(\mathbb{R})$.)

C'est cette dernière possibilité qui va être utilisée ici car nous avons des images assimilées à des matrices carrées de taille 8 et donc en Python :

```
def Dist(Img1,Img2):
    S=0
    for i in range(8):
        for j in range(8):
            S=S+(Img1[i][j]-Img2[i][j])**2
    return S**(1/2)
```

Remarque. En maths, $d(x,y) = N(x-y)$ définit une distance dès que N est une norme (cf. chapitre e.v.n.).

II.3 RECHERCHE DE LA RÉFÉRENCE MAJORITAIRE

A partir de là, on peut construire une méthode qui va permettre de classifier notre élément.

On fixe $k \in \mathbb{N}^\star$ et **on regarde les k éléments de nos données les plus proches de notre élément** (suivant la distance que l'on s'est fixée) et on prendra l'étiquette majoritaire.

Écrivons alors une fonction qui pour une liste `L` va renvoyer l'élément (ou un élément si celui-ci n'est pas unique) dont le nombre d'occurrences dans `L` est maximal.

```
def Majoritaire(L):
    """Renvoie un élément de L dont le nombre d'occurrences est maximal"""
    Dico={}# Dictionnaire du nombre d'occurrences des éléments de L
    Max=0
    for elt in L: # Pour chaque élément de L
        if elt in Dico: # Si on a déjà vu cet élément
            Dico[elt] = Dico[elt]+1 # Alors on a ajoute 1 à son nombre d'occurrences
        else:
            Dico[elt] = 1 # Sinon son nombre d'occurrences vaut 1
        if Dico[elt] >= Max: # Si le nombre d'occurrences  dépasse Max
            Max,eltmax = Dico[elt],elt # On actualise Max et eltmax
    return eltmax
```

Remarque. Bien entendu il faudra au préalable avoir construit la fameuse liste `L`, ce qui va nécessiter des calculs de distance mais également un algorithme de tri des données selon la distance calculée (pour déterminer les k plus proches).

II.4 ALGORITHME DES k PLUS PROCHES VOISINS

Finalement nous pouvons coder **l'algorithme des k plus proches voisins (KNN)**, qui étant donné une image non étiquetée retournera **l'étiquette majoritaire parmi les k images les plus proches** de notre image test.

Il reste donc à trier la liste `L` et donc faire appel à un algorithme de tri. Seulement ici, on a des images que l'on veut trier par proximité à une image fixée. On a donc besoin de la notion de **clé de tri** (fonction qui à tout élément d'un ensemble associe un nombre réel) qui permet de trier les éléments par valeurs croissante de clef.

```
def PlusProchesVoisins(img,k):
    """Renvoie l'étiquette majoritaire parmi les k images les plus proches img"""
    def cle(j):#Clé de tri
        return Dist(img,Images[j])
    trié = sorted(donnees,key=cle) # la liste données est trié selon les valeurs de la clé
    L=[Numeros[j] for j in trié[:k]] # on sélectionne les k plus proches éléments
    return Majoritaire(L) # on renvoie l'élément majoritaire
```

Remarque. Au lieu d'utiliser la commande `trié = sorted(donnees,key=cle)` il est possible de construire la liste `L = [(clef(j),j) for j in donnees]` puis de faire `L.sort()` qui trie `L` selon le premier élément de chaque tuple et ensuite récupérer la liste des éléments triés : `trié=[x[1] for x in L]`.

Pour tester notre algorithme, il va falloir séparer nos données en deux ensembles disjoints :

- L'ensemble d'apprentissage, qui contient les données de référence, qui vont permettre "d'apprendre" la correspondance éléments → étiquette.
- L'ensemble de test, qui contient les données que l'on va utiliser pour évaluer les performances de l'algorithme, en comparant la prévision de l'algorithme sur ces éléments à leur étiquette réelle connue.

```
N=1000  # On choisit le nombre N de données de référence
donnees=list(range(N))  # ensemble d'apprentissage (contient N données de référence)
test=list(range(N,len(Images)))  # le reste = données que l'on utilisera pour le test
```

II.5 MATRICE DE CONFUSION

On rappelle que l'on a fixé une valeur de k de façon arbitraire et qu'il est légitime de se demander quelle est la meilleure valeur possible. Voici une réponse possible à cette question. Ici, comme toutes les images qui sont dans `test` sont toutes étiquetées, on peut comparer à l'étiquette le résultat renvoyé par l'algorithme, c'est à dire la prédiction, pour savoir si l'algorithme a bien deviné.

Soit $C_{i,j}$ le nombre de fois où une image portant l'étiquette i, l'algorithme lui a attribué une étiquette j.
On obtient une matrice C, de taille 10×10, appelée **matrice de confusion**.
Plus cette matrice est proche d'une matrice diagonale, plus *a priori* l'algorithme est pertinent
(et donc plus le choix de k est pertinent). On peut programmer son affichage sous Python :

```
def MatriceDeConfusion(k):
    C=[[0 for j in range(10)] for i in range(10)]
    S=0 # Nombre de succès (prédictions justes de l'ordinateur)
    for l in test:
        i=digits.target[l] # l'étiquette sur Images[i] = vrai chiffre
        j=PlusProchesVoisins(l,k) # la prédiction donné par l'algorithme KNN
        C[i][j]=C[i][j]+1
        if i==j: # Bonne prédiction!
            S=S+1   # On compte un succès de plus!
    for i in range(10):
        print(C[i]) # on affiche la liste
    print("Pourcentage de réussite:",S*100/len(test))
```

On dit que l'algorithme des k plus proches voisins est un **algorithme supervisé**, en effet, il fonctionne en ayant donné à l'ordinateur les étiquettes de données afin qu'il puisse s'en servir sur des images tests.

II.6 QUELQUES REFLEXIONS

Cette idée d'apprentissage automatique ne date pas d'hier, puisque le terme de machine learning a été utilisé pour la première fois par l'informaticien américain Arthur Samuel en 1959.

Pourquoi le machine learning est tant « à la mode » depuis quelques années ? Simplement parce que le nerf de la guerre dans les algorithmes de machine learning est la qualité et la quantité des données (les données qui permettront à la machine d'apprendre à résoudre un problème), or, avec le développement d'Internet, il est relativement simple de trouver des données sur n'importe quel sujet (on parle de « big data »).

À noter aussi l'importance des stratégies mises en place par les 5 plus grandes plateformes numériques (Google, Apple, Facebook, Amazon et Microsoft) afin de récupérer un grand nombre de données concernant leurs clients. Ces données sont très souvent utilisées pour « nourrir » des algorithmes de machine learning. (Comment, d'après vous, Amazon arrive à proposer à ses clients des « suggestions d'achats » souvent très pertinentes ?)

III EXERCICES

Afin de travailler sur un exemple, nous allons utiliser un jeu de données relativement connu dans le monde du machine learning : le jeu de données « iris ».

Exercice 1 (Mignonne, allons voir si l'iris... (Ronsard, 1524)).
En 1936, Edgar Anderson, botaniste américain, a collecté des données sur 3 espèces d'iris : "iris setosa", "iris virginica" et "iris versicolor".

Pour chaque iris étudié, Anderson a mesuré (en cm) :

- la largeur des sépales
- la longueur des sépales
- la largeur des pétales
- la longueur des pétales

Iris Setosa

Iris Virginica

Iris Versicolor

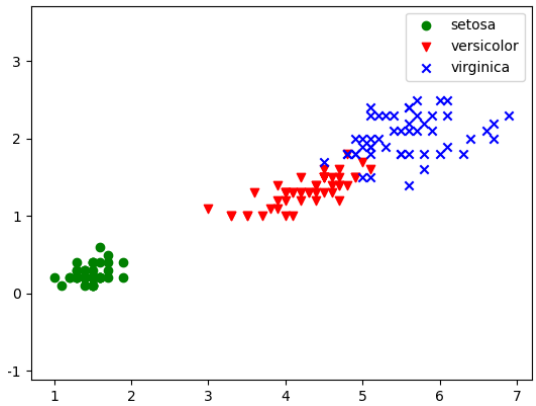
Par souci de simplification, nous nous intéresserons uniquement à la largeur et à la longueur des pétales. Pour chaque iris mesuré, Anderson a aussi noté l'espèce ("iris setosa", "iris versicolor" ou "iris virginica").

Vous trouverez 50 de ces mesures dans le fichier `iris.csv`.

En résumé, vous trouverez dans ce fichier :

- la largeur des pétales
- la longueur des pétales
- l'espèce de l'iris (on utilisera : 0 pour "iris setosa", 1 pour "iris versicolor" et 2 pour "iris virginica")

On peut représenter cet échantillon sur un graphe (en abscisse la longueur des pétale et en ordonnée la largeur des pétales).



Espèce de l'iris selon les caractéristiques des pétales

```
import pandas # Bibliothèque pour lire les fichiers csv
import matplotlib.pyplot as plt # Bibliothèque de tracé graphique
iris=pandas.read_csv("iris.csv") # Lecture du fichier iris.csv
x=iris.loc[:,"petal_length"].tolist() # Liste des longueurs des pétales des iris
y=iris.loc[:,"petal_width"].tolist() # Liste des largeurs des pétales des iris
species=iris.loc[:,"species"].tolist() # Liste des espèces des iris
N=len(species)
```

Voici les commandes nécessaires au tracé du graphique ci-dessus.

```
plt.axis('equal')
plt.scatter([x[k] for k in range(N) if species[k]==0],\
 [y[k] for k in range(N) if species[k]==0], color='g', label='setosa')
plt.scatter([x[k] for k in range(N) if species[k]==1],\
 [y[k] for k in range(N) if species[k]==1],  color='r', label='versicolor')
plt.scatter([x[k] for k in range(N) if species[k]==2],\
 [y[k] for k in range(N) if species[k]==2],  color='b', label='virginica')
plt.legend()
plt.show()
```

Le but est d'identifier l'espèce de tout nouvel iris connaissant la longueur et la largeur de ses pétales.

1. Exécuter le code proposé dans le fichier `TP_06.py`, observer que le graphique obtenu est bien celui sur votre énoncé de TP.

2. On a trouvé un nouvel iris, dont les pétales font $2,5$ cm de long et $0,75$ cm de large, et on cherche à déterminer son espèce. On le note sous Python `new_iris=(2.5,0.75)`. Ajouter une ligne de code permettant de le faire apparaitre sur le graphique en noir (`color='k'`). A votre avis, à quelle espèce appartient-il ?

On cherche à confirmer ou infirmer votre hypothèse par l'algorithme des k plus proches voisins.

3. A partir des listes `x`, `y` et `species`, créer une liste `data_flowers` dont chaque élément est un tuple de 3 éléments qui caractérise chaque iris : la longueur des pétales (flottant), la largeur des pétales (flottant), et enfin l'espèce (entier valant 0, 1 ou 2).
4. Créer une fonction `Dist(iris1,iris2)` qui calcule une distance entre deux iris de la forme de ceux de la liste `data_flowers`, ou bien de la forme de `new_iris`. (*On réfléchira à une formule adaptée pour caractériser la distance entre deux fleurs, en faisant comme si l'espèce était inconnue dans tous les cas.*)
5. Coder une fonction `Majoritaire(L)` qui renvoie l'élément qui apparait le plus souvent dans une liste `L`. Quelle est la complexité de votre fonction ?
6. Coder enfin une fonction `KNN(iris,k)` qui code l'algorithme des k plus proches voisins pour un entier naturel non nul `k` et un tuple `iris` ayant 2 ou 3 éléments (espèce connue ou inconnue).
7. Exécuter l'algorithme KNN pour l'iris `new_iris=(2.5,0.75)` et $k = 3$. A quelle espèce appartient-il ? Ré-exécuter KNN pour $k = 4$. Que se passe-t-il ? Expliquer.
8. Exécuter l'algorithme KNN pour des valeurs de k entre 1 et 20. Quelle espèce pensez-vous retenir au final ? Quelle valeur de k vous semble la plus adaptée ? (il n'y a pas une unique réponse à cette question...)
9. On cherche à obtenir une répartition du graphique tracé par zone de couleur : chaque couleur représenterait l'espèce logique à associé à tout nouvel iris découvert.
   Exécuter l'algorithme `KNN(iris,k)` pour la valeur de `k` choisie dans la question précédente et des iris ayant des longueurs de pétales entre entre 1 et 7 centimètres, par pas de 1 mm, et des largeurs de pétales entre 0 et 3 cm, par pas de 1 mm.
   Associer sur le graphique à chaque iris un point de la couleur associée à l'espèce renvoyée par l'algorithme `KNN`. Observer les 3 zones obtenues (verte, rouge, bleue).
   Vous pouvez faire varier `k` si vous le souhaitez pour voir les différences.
10. On souhaite tester l'efficacité de notre algorithme. On sépare les iris de `data_flowers` en deux lots :
    - ceux qui ont un numéro pair dans `data_flowers` deviennent des iris test, noté `test_flowers`.
    - ceux qui ont un numéro impair restent les iris du jeu de données `data_flowers_2`.

    Écrire une fonction `Mat_Confusion(k)` qui renvoie la matrice de confusion C_k obtenue avec cette répartition du jeu de données.
11. Faire afficher la matrice de confusion pour plusieurs valeurs de k. Laquelle vous semble la plus pertinente à choisir pour classifier vos iris ?