TP14 : Intelligence artificielle : méthode KNN

Nous allons étudier dans cette partie l'algorithme des k plus proches voisins sur l'exemple de données d'iris. En 1936, Edgar Anderson a collecté des données sur 3 espèces d'iris : "iris setosa", "iris virginica" et "iris versicolor"

iris setosa / iris virginica / iris versicolor

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

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

Par souci 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 virginica" ou "iris versicolor").
Les données sur les iris sont regroupés dans une liste `donnees`. Chaque élément de cette liste est un tuple contenant :

— Un flottant correspondant à la longueur des pétales

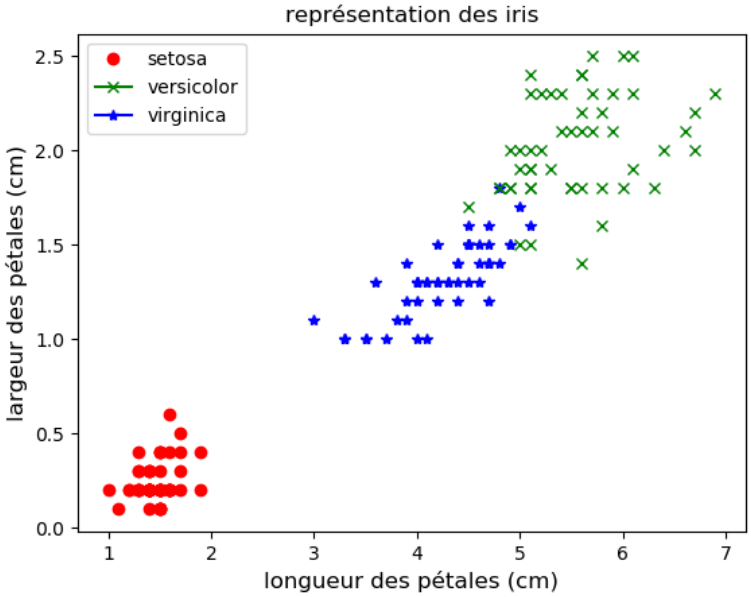
— Un flottant correspondant à la largeur des pétales

— un entier associé à l'espèce de l'iris : 0 pour "iris setosa", 1 pour "iris virginica" et 2 pour "iris versicolor".

Un exemple de tuple est `(1.4, 0.2, 0)` pour un "iris setosa" de longueur de pétales 1,4 cm et de largeur de pétales 0,2 cm.
Les données sont initialement stockées dans un fichier `iris.txt`. Chaque ligne contient la longueur des pétales, la largeur des pétales et la nature de l'espèce, le tout séparés par des virgules. Le début du fichier est le suivant :

```
1.4,0.2,0
1.4,0.2,0
1.3,0.2,0
```

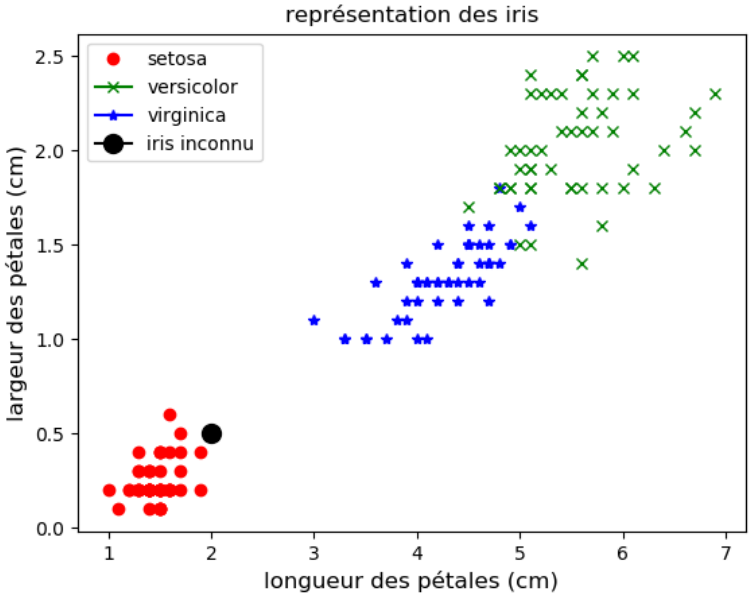
1. Compléter la ligne 4 du fichier élève en indiquant l'adresse du dossier contenant votre fichier `iris.txt`. Exécuter le fichier élève et vérifier que vous obtenez bien la représentation suivante des iris.



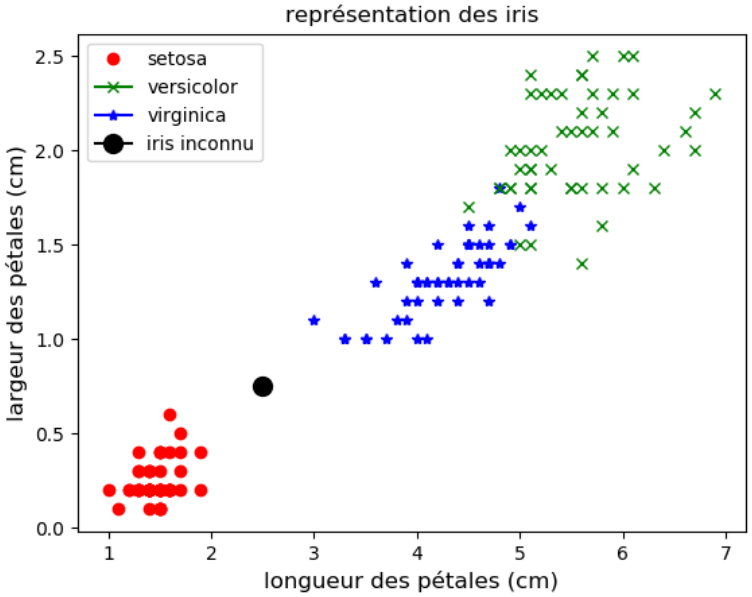
Nous obtenons des "nuages" de points, on remarque ces points sont regroupés par espèces d'iris (sauf pour "iris virginica" et "iris versicolor", les points ont un peu tendance à se mélanger).

Imaginez maintenant qu'au cours d'une promenade vous trouviez un iris, n'étant pas un spécialiste, il ne vous est pas vraiment possible de déterminer l'espèce. En revanche, vous êtes capables de mesurer la longueur et la largeur des pétales de cet iris. Partons du principe qu'un pétale fasse 0.5 cm de large et 2 cm de long.

Le résultat est le suivant et il semble sans appel. Il y a de fortes chances que votre iris soit de l'espèce "iris setosa".



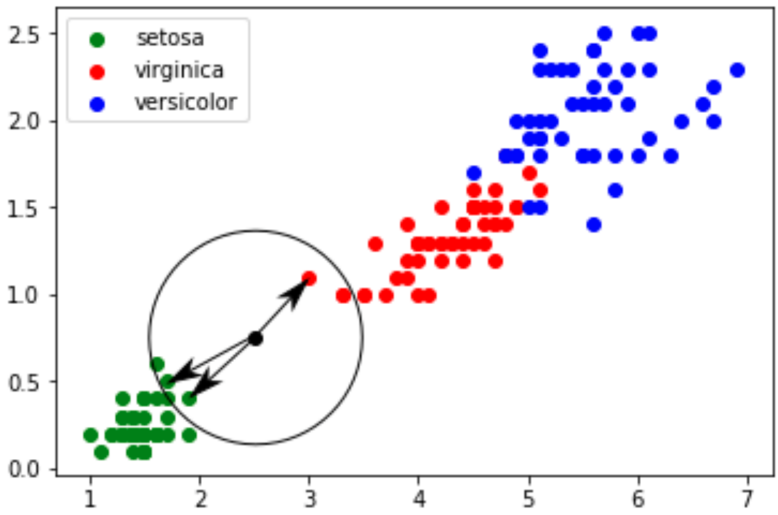
Il est possible de rencontrer des cas plus difficiles, par exemple : largeur du pétale = 0,75 cm ; longueur du pétale = 2,5 cm :



Dans ce genre de cas, il peut être intéressant d'utiliser l'algorithme des "k plus proches voisins", qui repose sur les étapes suivantes :

- on calcule la distance entre notre point (largeur du pétale = 0,75 cm ; longueur du pétale = 2,5 cm) et chaque point issu du jeu de données "iris" (à chaque fois c'est un calcul de distance entre 2 points tout ce qu'il y a de plus classique)
- on sélectionne uniquement les k distances les plus petites (les k plus proches voisins)
- parmi les k plus proches voisins, on détermine quelle est l'espèce majoritaire. On associe à notre "iris mystère" cette "espèce majoritaire parmi les k plus proches voisins"

Par exemple, pour $k = 3$:



Les 3 plus proches voisins sont signalés ci-dessus avec des flèches : nous avons deux "iris setosa" (point vert) et un "iris virginica" (point rouge). D'après l'algorithme des "k plus proches voisins", notre "iris mystère" appartient à l'espèce "setosa".

2. Ecrire une fonction `nombre_iris(donn:list)->list` prenant en argument une liste de tuples d'iris `donn` (telle que `donnees`) et retournant une liste de 3 éléments contenant le nombre d'iris de chaque espèce.

   Test : on vérifiera que `nombre_iris(donnees)` retourne `[50,50,50]`

3. Ecrire une fonction `distance(P1:tuple,P2:tuple)->float` qui prend en arguments deux tuples `P1` et `P2` correspondant aux coordonnées (x,y) de deux points du graphique et retournant la distance euclidienne entre ces deux points.

   Test : on vérifiera que `distance((0,1),(1,0))` retourne `1.4142135623730951`

4. Ecrire une fonction `liste_dist(donnees:list,P:tuple)->list` prenant en argument la liste `donnees` telle que celle issue de la question 1 (contenant toutes les informations sur les iris mémorisés) et un tuple `P` correspondant aux coordonnées (x,y) d'un iris inconnu et retournant une liste contenant autant de tuples que d'élements dans `donnees`. Le ième tuple contiendra la distance de `P` au ième point de `donnees` ainsi que l'entier associé à cet iris.

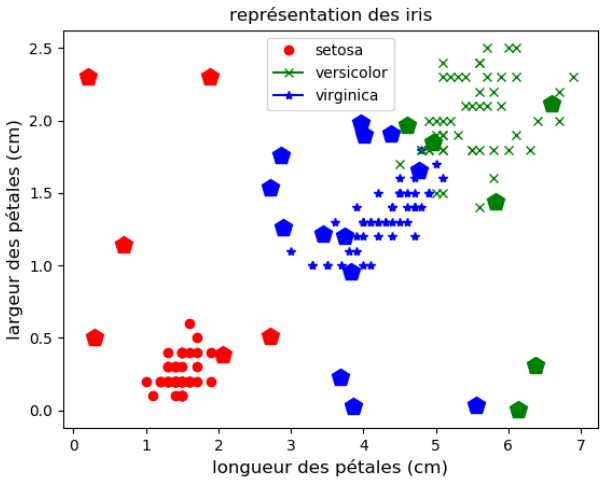
   Test : on vérifiera que `liste_dist([(1.4,0.2,0),(4.5,1.5,1)],(1.0,1.0))` retourne environ `[(0.9,0), (3.5,1)]`

Il faut maintenant procéder à un tri de la liste précédente par ordre croissant de distances. On utilisera la méthode `sort` de Python qui trie en place une liste `L` selon la syntaxe `L.sort()`

5. Tester la méthode `sort` sur l'exemple `[(7.3,2),(2.8,1),(1.3,0)]`.

Maintenant que l'on peut trier la liste des distances, le principe de l'algorithme KNN est de se donner un entier k et de retourner la classe d'iris majoritaire parmi les k iris les plus proches de l'iris inconnu.

6. Ecrire une fonction `indice_max(L:list)->int` retournant l'indice du maximum d'une liste `L`.
7. Quelle est la complexité de la fonction `indice_max` en fonction de `n = len(L)` ?
8. Ecrire une fonction `knn(donnees:list, P:tuple, k:int, n=3:int)->int` prenant en argument la liste `donnees` des iris enregistrés, les coordonnées (x,y) de l'iris inconnu dans le tuple `P`, le nombre `n` de catégories différentes (3 par défaut) et l'entier `k` de la méthode KNN et qui retourne la catégorie supposée du point `P`. On suivra les étapes suivantes :
   - Calculer la liste des distances entre `P` et `donnees`.
   - Trier cette liste par distances croissantes.
   - Déterminer le nombre d'iris de chaque catégories en se limitant aux `k` premiers iris.
   - Retourner la catégorie de l'iris inconnu.
9. **Test :** Vérifier que `knn(donnees,(2.5,0.75),3,3)` retourne 0 et que `knn(donnees,(6.2,2.2),10,3)` retourne 2.
10. Pour une deuxième batterie de test, on va générer 25 iris aléatoires, leur affecter une catégorie via la méthode KNN puis représenter ces iris. Compléter le fichier élève pour que la liste `inc` contienne 25 points aléatoires dont les abscisses seront comprises entre 0 et 7 et les ordonnées entre 0 et 2.5. On rappelle que la fonction `random` retourne un flottant aléatoire entre 0 et 1. Exécuter le fichier élève pour représenter ces iris inconnus. Ils apparaîtront comme des pentagones de la couleur correspondante à la catégorie. Voici par exemple ce que j'ai obtenu :



On souhaite tester autrement notre méthode des KNN sur nos données. Pour cela, parmi les 150 iris présents dans nos données (cf question 2), on va utiliser 100 de ces iris comme "apprentissage" et 50 comme "test". Comme les iris sont initialement classés par catégorie, on va commencer par mélanger toutes nos données à l'aide de la fonction `shuffle(liste)` que l'on supposera importée.

11. Ecrire une fonction `test(donnees:list, k:int , n=3:int)->float` prenant en argument la liste `donnees`, le nombre `n` de catégories et l'entier `k` de la méthode KNN et retournant le pourcentage de bonnes prédictions.

    On commencera par mélanger les données, sélectionner les 100 premières comme "apprentissage" et considérer les 50 autres comme des iris inconnus. Comme on connaît leur catégorie, on pourra comparer la catégorie retournée par la méthode KNN avec les 100 données d'apprentissage à leur vraie catégorie.

12. Procéder au test pour différentes valeurs de `k` et afficher les résultats comme suit :

```
le test pour k = 1 a conduit a 94.0 % de bonnes prédictions
le test pour k = 5 a conduit a 96.0 % de bonnes prédictions
le test pour k = 10 a conduit a 94.0 % de bonnes prédictions
le test pour k = 50 a conduit a 96.0 % de bonnes prédictions
le test pour k = 80 a conduit a 28.0 % de bonnes prédictions
le test pour k = 100 a conduit a 18.0 % de bonnes prédictions
```

On veut terminer en calculant la matrice de confusion. Elle est construite de la manière suivante :

- Chaque ligne correspond à la classe réelle de l'iris.
- Chaque colonne correspond à la classe prédite.
- À l'intersection d'une ligne i et d'une colonne j, on trouve le nombre d'éléments de classe réelle i et de classe prédite j. On trouve notamment sur la diagonale les prédictions correctes.

13. Ecrire une fonction `matrice_confusion(donnees:list, k:int, n=3:int)` prenant en argument les données `donnees`, le nombre de catégories `n` et l'entier `k` de la méthode KNN et retournant la matrice de confusion suite aux tests. On choisira de nouveau 50 échantillons tests aléatoires parmi les données et 100 échantillons d'apprentissage.
14. Ecrire une fonction `kmax(donnees:list,n=3:int)->int` prenant en argument les données et le nombre de catégories et retournant la valeur de k qui conduit au meilleur taux de prédictions. On cherchera pour des valeurs de k dans l'intervalle $[\![1; 100]\!]$.

Pour aller plus loin : algorithme des K-moyennes

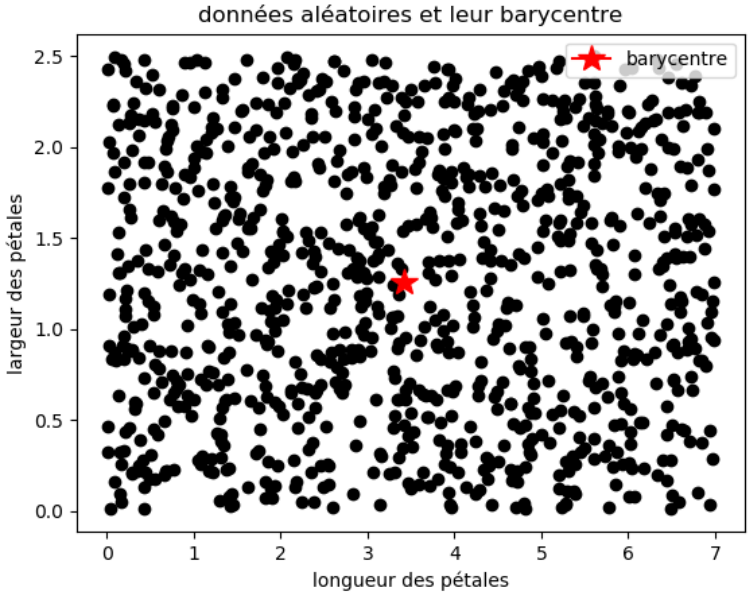
Dans la partie précédente, on a travaillé à partir d'iris déjà catégorisés dont on s'est servis pour déterminer la catégorie d'iris inconnus. Cette méthode s'appelle, en intelligence artificielle, de l'apprentissage **supervisé**. Il existe une autre méthode, appelée apprentissage **non supervisé** dans laquelle on va partir de données initialement non catégorisées. Dans cette partie, on va se baser sur l'algorithme des K-moyennes, qui va consister à construire K catégories à partir de nos données.

15. Générer, dans une liste `diris`, 1000 points aléatoires dont les abscisses sont comprises entre 0 et 10 et les ordonnées sont comprises entre 0 et 10.

    On rappelle que la fonction `random` du module `random` retourne un flottant compris entre 0 et 1.

16. Ecrire une fonction `barycentre(L:list)->tuple` prenant en argument une liste `L` de coordonnées (x,y) et retournant le barycentre de cette liste. L'abscisse x_B du barycentre et l'ordonnée y_B du barycentre sont simplement les moyennes de toutes les abscisses et de toutes les ordonnées des points des données.

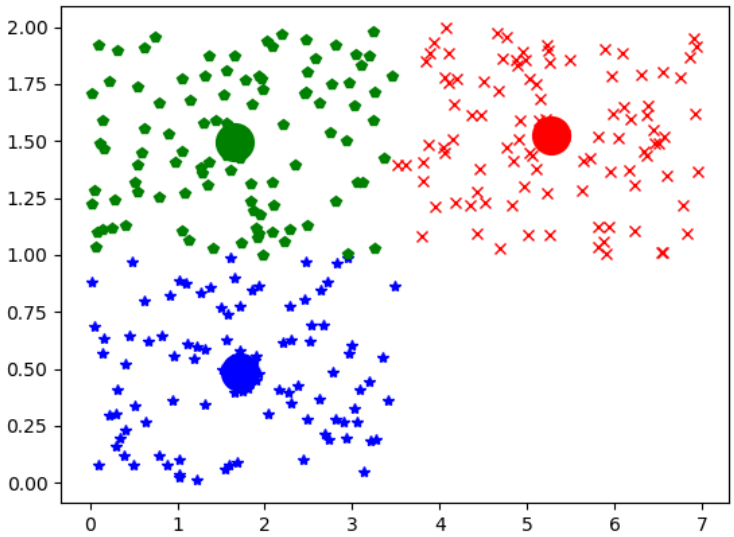
    Test : exécuter le fichier élève pour afficher vos données aléatoires ainsi que le barycentre. Voici un exemple de ce que l'on doit obtenir :



Dans la suite, on va manipuler des clusters de différentes catégories. On va donc manipuler une liste de listes où chacune des sous-listes correspond à un ensemble de points de la même catégorie.

17. Ecrire une fonction `calcul_barycentres(clusters:list)->list` prenant en argument une liste de listes `clusters` et retournant une liste contenant les coordonnées des barycentres de chacun des clusters différents. *Cette fonction devra utiliser la fonction* ***`barycentre`***.

    Test : exécuter le fichier élève et vérifier que l'on obtient bien une figure comme celle ci-dessous avec 3 barycentres pour chacun des 3 clusters :



Le principe de l'algorithme des K-moyennes est le suivant :

- Initialiser aléatoirement K centroïdes comme centres des K clusters souhaités.
- Affecter chaque donnée de départ au cluster dont le barycentre est le plus proche.
- Recalculer les barycentres des clusters ainsi formés.
- Recalculer les clusters avec les nouveaux barycentres.
- Continuer ainsi jusqu'à ce que les centroïdes ne bougent plus. **On admettra la convergence de l'algorithme.**

18. Ecrire une fonction `initialisation(L:list, K:int)->list` prenant en argument une liste `L` de points du plan et retournant une liste contenant `K` points aléatoires de cette liste.

    ATTENTION : il faut s'assurer que tous les centroïdes sont bien différents.

19. Ecrire une fonction `plus_proche(P,L_bary)` prenant en argument un point `P` et la liste des barycentres des clusters `L_bary` et retournant **l'indice** du barycentre le plus proche de `P`.

20. Ecrire une fonction `clust(L:list, L_bary:list)->list` prenant en argument la liste de nos données `L` et une liste de barycentres `L_bary` et retournant la liste des K clusters (où K est le nombre de barycentres) formés en plaçant chaque donnée dans le cluster du barycentre le plus proche.

21. Ecrire une fonction `Kmoyennes(L:list,K:int)->list` effectuant l'algorithme des Kmoyennes en se basant sur la méthode ci-dessus. La fonction devra retourner la liste des K clusters une fois que les centroïdes sont stables.

    Test : exécuter le fichier élève pour représenter vos clusters. Vous pourrez faire varier le nombre de clusters et le nombre de points des données pour observer l'efficacité de la méthode. Voici un exemple de ce qu'on obtient pour 6 clusters :