




TD04 : Apprentissage supervisé

1 k -plus proches voisins

1.1 Présentation du problème

On considère le cas d'école décrit ici : https://pixees.fr/informatiquelycee/n_site/nsi_prem_knn.html où l'on cherche à construire un algorithme capable de classer une fleur d'iris en fonction de deux caractéristiques : la largeur et la longueur des pétales.

On cherche à déterminer si une fleur d'iris que l'on a cueillie appartient à l'une des espèces suivantes :

		
iris setosa	iris virginica	iris versicolor

On dispose d'une base de données recensant les caractéristiques (largeur et longueur des pétales) pour un certain nombre de fleurs cueillies et identifiées comme appartenant à l'une de ces trois espèces, et lorsque l'on cueille une nouvelle fleur, on cherche de quelle espèce ses caractéristiques mesurées la rapproche le plus, à l'aide de l'algorithme des k -plus proches voisins.

1.2 Traitement de la base de données de référence

On dispose d'un fichier **iris.csv** dont la première ligne est une ligne d'entêtes et dans laquelle chacune des lignes suivantes contient la largeur et la longueur du pétale d'une fleur d'iris que l'on a classée comme appartenant à l'une des trois espèces citées ci-dessus, et dont l'espèce apparaît en dernière position avec le codage suivant : 0 pour *iris setosa*, 1 pour *iris virginica*, 2 pour *iris versicolor*.

Voici une capture des premières lignes de ce fichier :

```
iris - Bloc-notes
Fichier  Modifier  Affichage  ⚙️

petal_length,petal_width,species
1.4,0.2,0
1.4,0.2,0
1.3,0.2,0
1.5,0.2,0
1.4,0.2,0
```

Question 1. Écrire une séquence d'instructions permettant d'ouvrir le fichier **iris.csv** et construisant une liste de tuples, `data`, dont chaque élément donne, dans cet ordre, la longueur de pétale, la largeur de pétale, l'espèce, de chacune des fleurs décrites dans la base de données.

On rappelle les fonctions de conversion de type, `float` et `int`, permettant respectivement de convertir une chaîne de caractères en flottant ou en entier, si son formatage s'y prête.

Question 2. À partir de la liste `data`, déterminer le nombre de fleurs décrites dans la base, ainsi que le nombre de représentantes de chacune des trois espèces.

Question 3. On souhaite représenter graphiquement les données de la base, en représentant dans un repère, chaque fleur de l'échantillon, par un point dont l'abscisse et l'ordonnée sont, respectivement, la longueur et la largeur de son pétale.

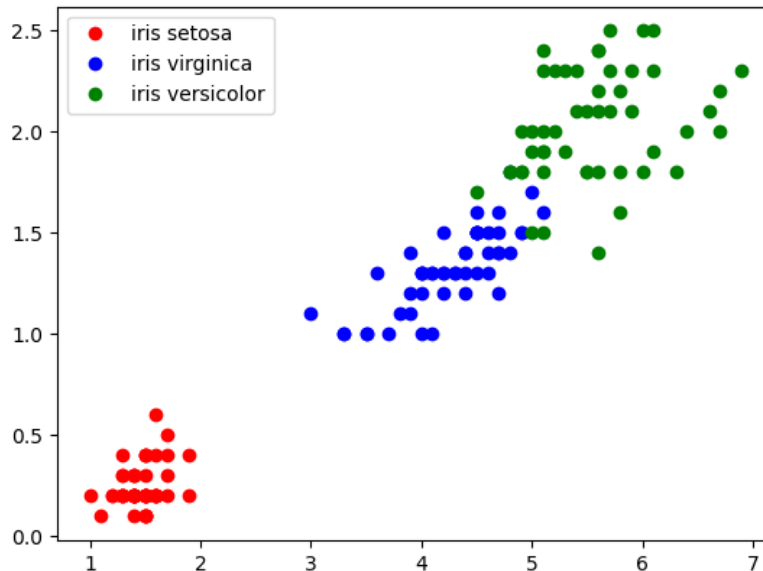
On choisira les couleurs suivantes : rouge pour *iris setosa*, bleu pour *iris virginica*, vert pour *iris versicolor*.

On rappelle que pour représenter un nuage de points dont les abscisses sont données par une liste `Lx` et les ordonnées par une liste `Ly`, avec une couleur `c`, légendé par la chaîne de caractères `etiquette`, on peut utiliser l'instruction suivante :

```
plt.plot(Lx, Ly, marker = 'o', linestyle = '', color = c,
         label = etiquette)
```

et que l'instruction `plt.legend()` permet d'afficher les légendes.

On devra obtenir la figure suivante.



1.3 Mise en place de la méthode

Afin de rapprocher une fleur de l'une des trois espèces, on cherche à calculer sa distance à chacune des fleurs de l'échantillon de référence.

On définit la distance entre deux fleurs dont les longueurs et largeurs de pétales respectives sont (x, y) et (x', y') par la distance euclidienne usuelle entre ces deux couples éléments de \mathbb{R}^2 .

Question 4. Définir une fonction `distance(t1, t2)`, calculant la distance entre deux fleurs en fonction de leurs coordonnées, données par les tuples t_1 et t_2 .

Question 5. Définir une fonction `liste_distances(data, t)`, prenant en argument une liste des données au format défini en question Q1, et les caractéristiques d'une fleur que l'on souhaite classifier, sous la forme d'un couple $t = (longueur, largeur)$, donnant longueur et largeur de ses pétales, et renvoyant la liste des couples (distance, espèce) où la distance est celle de la fleur à classifier à l'une des fleurs de l'échantillon et l'espèce celle de la fleur à laquelle la fleur à classifier aura été comparée. On obtiendra donc une liste comportant autant de couples qu'il y a de fleurs dans l'échantillon.

La liste `data` sera utilisée en tant que variable globale.

Question 6. Définir une fonction `trier`, prenant en argument une liste de couples (distance, espèce) telle que celles renvoyées par la fonction `liste_distances` et ordonnant la liste en argument par distance croissante.

On utilisera un des algorithmes de tri vu en première année, en rappelant sa complexité.

1.4 Algorithme knn (*k-nearest neighbors*)

L'algorithme des k -plus proches voisins consiste, pour déterminer à quelle espèce appartient une fleur, à se donner un entier k , et à examiner quelle est l'espèce majoritaire parmi les k fleurs de la base les plus proches de la fleur à classifier.

Question 7. Définir une fonction `classification_knn(data, t, k)`, renvoyant l'espèce majoritaire parmi les k plus proches voisines trouvées dans la base pour une fleur de caractéristiques t . On réutilisera les fonctions définies précédemment.

Question 8. Évaluer la complexité de la fonction `classification_knn(data, t, k)`, en fonction du nombre p d'espèces distinctes, du nombre k de voisins pris en compte, et du nombre n de fleurs dans la base.

Question 9. Proposer des pistes pour améliorer cette complexité.

1.5 Cartographie de la classification

On souhaite visualiser les résultats de prédiction pour une valeur donnée de k .

Pour cela, on se donne un maillage de pas h de la partie du domaine du plan $[0; 7] \times [0; 2,5]$ et on souhaite colorier chaque carré de côté h et dont le coin inférieur gauche est $(i \times h; j \times h)$ de ce domaine

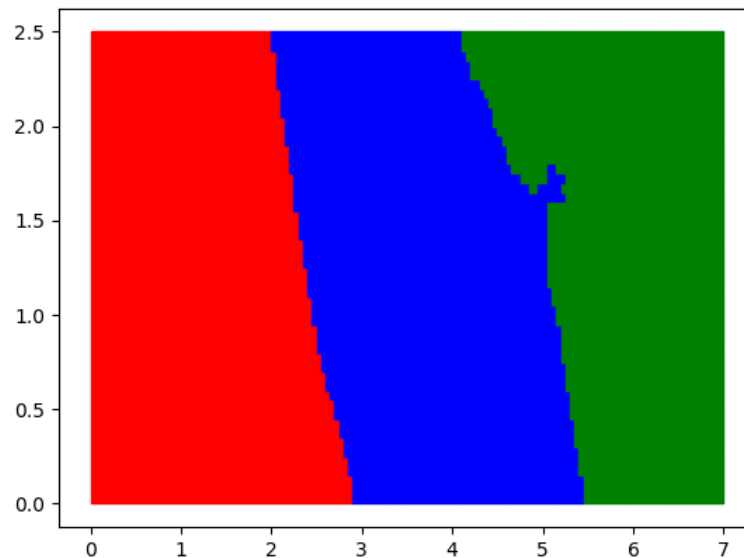
avec la couleur de l'espèce majoritaire parmi les k plus proches voisines de la fleur décrite par le couple $(i \times h; j \times h)$.

Question 10. Écrire une séquence d'instructions produisant un tel graphique pour des valeurs données de k et de h .

On rappelle que pour tracer un rectangle de couleur c , dont les sommets, listés dans le sens trigonométrique, ont pour coordonnées $[(x_0, y_0), (x_1, y_0), (x_1, y_1), (x_0, y_1)]$ on peut utiliser l'instruction

```
plt.fill([x0, x1, x1, x0, x0], [y0, y0, y1, y1, y0], color = c)
```

On obtiendra, pour $k = 3$ et $h = 0,05$:



1.6 Qualité des prédictions

Afin de mesurer la qualité des prédictions, il est d'usage de retirer de l'échantillon une fraction des données, en scindant les données en deux ensembles : un ensemble R servant de référence et avec lequel on construit la liste `data`, et un ensemble T sur les éléments duquel on teste les prédictions.

Question 11. Vérifier que dans la liste `data`, les cinquante premières fleurs sont de l'espèce 0, les cinquante suivantes de l'espèce 1 et les cinquante dernières de l'espèce 2.

Question 12. On donne la documentation de la fonction `sample` du module `random` :

```
>>> from random import sample
>>> help(sample)
Help on method sample in module random:

sample(population, k, *, counts=None) method of random.Random instance
    Chooses k unique random elements from a population sequence or set.
    Returns a new list containing elements from the population while
    leaving the original population unchanged. The resulting list is
    in selection order so that all sub-slices will also be valid random
    samples. This allows raffle winners (the sample) to be partitioned
    into grand prize and second place winners (the subslices).
    Members of the population need not be hashable or unique. If the
    population contains repeats, then each occurrence is a possible
    selection in the sample.
    Repeated elements can be specified one at a time or with the optional
    counts parameter. For example:
        sample(['red', 'blue'], counts=[4, 2], k=5)
    is equivalent to:
        sample(['red', 'red', 'red', 'red', 'blue', 'blue'], k=5)
    To choose a sample from a range of integers, use range() for the
    population argument. This is especially fast and space efficient
    for sampling from a large population:
        sample(range(10000000), 60)
```

Utiliser la fonction `sample` pour construire aléatoirement une liste T de 15 fleurs, dont cinq sont de l'espèce 0, cinq de l'espèce 1 et cinq de l'espèce 2. On construira une liste R avec les fleurs non choisies.

Question 13. Reprendre le code des fonctions `liste_distances(t)` et `classification_knn2(t, k)` afin de passer également la liste des données en paramètre : on définira donc deux fonctions `liste_distances(data, t)` et `classification_knn3(data, t, k)`

Afin de tester la qualité des prédictions, on construit une matrice, appelée [matrice de confusion](#), et donnant le nombre de prédictions correctes et incorrectes pour les fleurs-test (liste T) lorsque l'on prend comme données de bases les fleurs de référence (liste R).

La matrice de confusion correspond, sur l'exemple, au tableau suivant :

		Espèce prédite par le classificateur <code>classification_knn3(R, t, k)</code>		
		espèce 0	espèce 1	espèce 2
Espèce réelle (éléments de T)	espèce 0			
	espèce 1			
	espèce 2			

Chaque ligne, étiquetée « espèce i », est remplie avec le nombre fleurs de l'ensemble de test R qui sont de l'espèce, qui ont été classifiées, par appel à `classification_knn2`, comme appartenant à l'une ou l'autre des trois espèces.

On espère idéalement obtenir une matrice diagonale, ce qui correspondrait à des prédictions exactes pour toutes les fleurs de l'ensemble de test T .

Question 14. Écrire une séquence d'instruction permettant de construire une matrice C , de taille 3×3 , initialement remplie de zéros, et de la remplir ensuite de sorte à obtenir la matrice de confusion pour $k = 3$ et l'ensemble de données R et l'ensemble de test T .

On pourra refaire des tests en changeant les tailles des ensembles T et R , ainsi que le nombre de voisins k .