



Math93.com

# TD - NSI

## Les $k$ plus proches voisins

### - Un exemple

#### Objetctifs

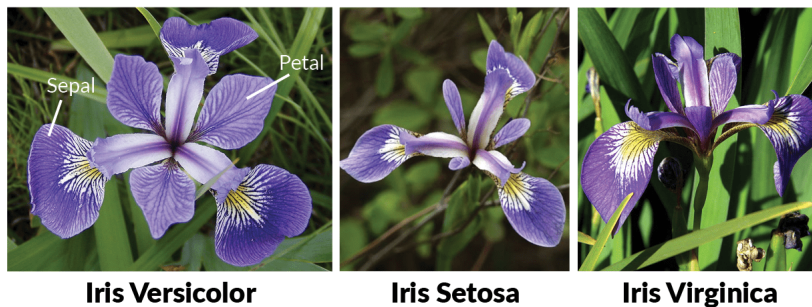
- **Thème**  
Un exemple historique d'utilisation de l'algorithme des  $k$  plus proches voisins.
- **Prérequis Python**  
Bonne connaissance de Python, dictionnaires, listes, fichiers CSV.  
Il est conseillé d'avoir traité le TD sur les fichiers CSV et celui sur les dictionnaires (lien).
- **Prérequis**  
Avoir lu et visionné la vidéo de cours sur l'algorithme des  $k$  plus proches voisins, disponible sur la page [www.math93.com](http://www.math93.com)

## 1 Le jeu de données Iris de Fisher

Nous allons ici appliquer l'algorithme des  $k$  plus proches voisins sur un exemple concret.

Ce **jeu de données Iris** connu aussi sous le nom de **Iris de Fisher** est un jeu de données présenté en 1936 par Ronald Fisher dans son papier "*The use of multiple measurements in taxonomic problems*".

Le jeu de données comprend 50 échantillons de chacune des trois espèces d'iris (Iris setosa, Iris virginica et Iris versicolor). Quatre caractéristiques ont été mesurées à partir de chaque échantillon : la longueur et la largeur des sépales et des pétales, en centimètres.



Sur la base de la combinaison de ces quatre variables, Fisher a élaboré un modèle d'analyse permettant de distinguer les espèces les unes des autres.

## 2 Un travail sur les données brutes

Il est possible de télécharger ces données au format csv : `iris_data_set.csv`

```
iris_data_set.csv  saved
1  sepal_length,sepal_width,petal_length,petal_width,species
2  5.1,3.5,1.4,0.2,setosa
3  4.9,3.0,1.4,0.2,setosa
4  4.7,3.2,1.3,0.2,setosa
5  4.6,3.1,1.5,0.2,setosa
```

Ce jeu de données est composé de 150 entrées, pour chaque entrée nous avons :

- la longueur des sépales (en cm), première valeur : *sepal\_length*;
- la largeur des sépales (en cm), deuxième valeur : *sepal\_width*;;
- la longueur des pétales (en cm), troisième valeur : *petal\_length*;
- la largeur des pétales (en cm), quatrième valeur : *petal\_width*;
- l'espèce d'iris : **Iris setosa**, **Iris virginica** ou **Iris versicolor** (label) , cinquième donnée : *species*.



### Exercice 1

En vous aidant du cours sur les fichiers CSV :

1. Importer ce fichier CSV nommé *iris\_data\_set.csv* sur votre éditeur Python (Repl.it, Jupyter ...) : **lien du fichier**.
2. Dans un souci de simplification, nous allons travailler uniquement sur la taille des pétales, et donc supprimer les colonnes "*sepal\_length*" et "*sepal\_width*".  
A partir du fichier initial, créer un fichier .CSV nommé *NouveauFichier.csv* ne conservant que les colonnes longueur pétale, largeur pétale et l'espèce.
3. Profitez-en pour renommer les titres en : *longueur\_petale*, *largeur\_petale*, *espece*.

Vous devrez obtenir un fichier *NouveauFichier.csv* de cette forme :

	NouveauFichier.csv	saved
1	longueur_petale, largeur_petale, espece	
2	1.4,0.2,setosa	
3	1.4,0.2,setosa	
4	1.3,0.2,setosa	
5	1.5,0.2,setosa	
6	1.4,0.2,setosa	

Aide : voici le fichier **NouveauFichier.csv** si vous êtes bloqués et le début du code.

```
# Dans l'éditeur Python
import csv

def convert(dico): # à compléter
    return {'longueur_petale':float(dico['...']),
    .... }

fichier=open("iris_data_set.csv")
table=list(csv.DictReader(fichier,delimiter=","))
iris=[convert(ligne) for ligne in table]
fichier.close()

# Ecriture (export) d'un CSV avec Python

with open("NouveauFichier.csv","w") as sortie:
    objet=csv.DictWriter(sortie,['longueur_petale','largeur_petale','espece'])

    # Pour écrire la ligne d'entêtes
    objet.writeheader()

    # on prend la table iris en argument
    objet.writerows(iris)
```

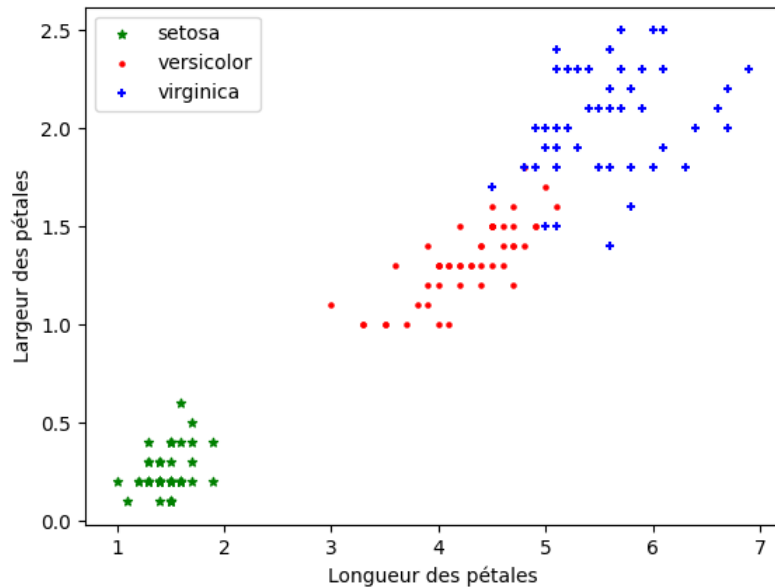
### 3 Représentation graphique

On cherche à obtenir une représentation graphique des données.



#### Exercice 2

Voici le début du code permettant d'afficher les données du premier style d'Iris, complétez-le afin d'obtenir un graphique de ce type. Vous devez obtenir ce graphique :



```
# Dans l'éditeur Python
fichier=open("NouveauFichier.csv")
table=list(csv.DictReader(fichier,delimiter=","))
X_iris_0=[float(ligne['longueur_petale']) for ligne in table
if ligne['espece']=='setosa']
Y_iris_0=[float(ligne['largeur_petale']) for ligne in table
if ligne['espece']=='setosa']
X_iris_1=..
Y_iris_1=..
X_iris_2=...
Y_iris_2=...
fichier.close()
plt.figure()
# Options de scatter : s = taille du point (20 par défaut),
# marker = symbole, on a 'o' : rond. 's' : carré (square).
# '+' : croix en forme de +. 'x' : croix en forme de x. '*' : étoile.
plt.scatter(X_iris_0, Y_iris_0, color='g', label='setosa',s = 20, marker = '*')
plt.scatter(X_iris_1, Y_iris_1, color='r', label='versicolor',s = 20,
marker = '.')
plt.scatter(X_iris_2, Y_iris_2, color='b', label='virginica',s = 20,
marker = '+')
plt.legend()
# lable des axes
plt.xlabel('Longueur des pétales')
plt.ylabel('Largeur des pétales')
plt.show()
plt.savefig('plot.png')
```

## 4 Deux nouvelle Iris : des conjectures

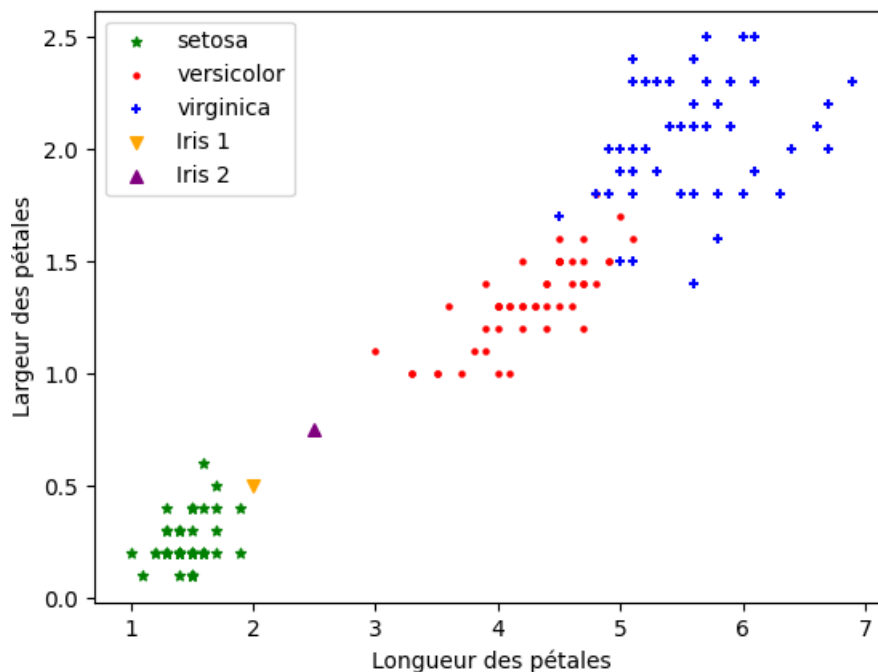
On va introduire deux nouvelles données et on cherche à déterminer à quelle espèce elles appartiennent.

- Une Iris 1 telle que :
  - longueur du pétale = 2 cm ;
  - largeur du pétale = 0,5 cm .
- Une Iris 2 telle que :
  - longueur du pétale = 2,5 cm ;
  - largeur du pétale = 0,75 cm .



### Exercice 3

Introduisez sur votre graphique ces deux nouvelles Iris et conjecturer l'espèce de chacune d'elle. On remarque que pour l'une des deux, c'est assez indécis. Vous devez obtenir ce graphique :



```
# Options de scatter : s = taille du point (20 par défaut),
# marker = symbole, on a
# 'o' : rond.
# 's' : carré (square).
# '+' : croix en forme de +.
# 'x' : croix en forme de x.
# '*' : étoile.
# 'D' : losange (diamond).
# 'd' : losange allongé.
# 'H' : hexagone ('h' est aussi un hexagone, mais tourné).
# 'p' : pentagone.
# '.' : point.
# '>' : triangle vers la droite ('<' pour vers la gauche).
# 'v' : triangle vers le bas ('^' pour vers la haut).
# '|' : trait vertical ('_' pour trait horizontal).
# '1' : croix à 3 branches vers le bas, '2' vers le haut, '3' gauche,
# '4' droite).
```

## 5 L'algorithme des k plus proches voisins

### 5.1 Le cours



#### Algorithme des k plus proches voisins

Soit un ensemble E contenant n données labellisées.

1. On calcule les distances entre la nouvelle donnée C et chaque donnée appartenant à E à l'aide de la fonction distance d. On choisira la distance dite euclidienne définie par :

$$d(A, B) = \sqrt{(x_B - x_A)^2 + (y_B - y_A)^2}$$

2. On retient les k éléments de E les plus proches de C, pour la distance choisie d.
3. On attribue à C la classe qui est la plus fréquente parmi les k données les plus proches.

### 5.2 Fonction knn



#### Exercice 4

On considère que la nouvelle donnée à étudier est B.

Par exemple Iris1(2, 0.5) ou Iris1(2.5, 0.75).

1. Écrire la fonction distance euclidienne  $d(x_A, y_A, x_B, y_B)$  et qui renvoie la distance entre les points  $A(x_A; y_A)$  et  $B(x_B; y_B)$ .
2. On cherche à écrire une fonction **knn(liste\_dico, k, longueur, largeur)** qui calcule la liste triée des tuples (distance, espece) et retient les k premiers.
  - (a) On part de la liste **iris** obtenue dans l'exercice 1. Elle est constituée d'une liste de dictionnaires (voir ci-dessous).
  - (b) On boucle sur les éléments X de cette liste.
  - (c) On va créer une liste L composée des couples (distances, espece).  
La distance est celle entre l'élément X et B (*longueur ; largeur*) soit .
  - (d) On trie cette liste de tuples par rapport au premier élément, c'est à dire la distance.  
**aide** : `sorted(L, key = lambda x : x[0])`
  - (e) On renvoie les k premiers éléments.  
**aide** : `L[0 : k]`

Consultez sur la page suivante le début du code et les résultats attendus.



#### Aide

🔗 Quelques éléments pour trier des listes et des dictionnaires sur la page 8.

```
# Pour trier une liste L de tuples par rapport à un élément,  
# on utilise la fonction sorted et une clé  
# trier/1er élément : sorted(L, key=lambda x: x[0])  
# trier/2e élément : sorted(L, key=lambda x: x[1])  
def knn(liste_dico,k,longueur,largeur):  
    '''In : liste de dictionnaires (ici iris), le paramètre k  
        et la longueur et largeur de l'iris à tester  
        Out : une liste des k plus proches voisins'''  
    L=[]  
    xb=longueur  
    yb=largeur  
    for dico in liste_dico:  
        ...
```

```
>>> iris  
[{'longueur_petale': 1.4, 'largeur_petale': 0.2, 'espece': 'setosa'},  
{'longueur_petale': 1.4, 'largeur_petale': 0.2, 'espece': 'setosa'},  
{'longueur_petale': 1.3, 'largeur_petale': 0.2, 'espece': 'setosa'},  
{'longueur_petale': 1.5, 'largeur_petale': 0.2, 'espece': 'setosa'},  
{'longueur_petale': 1.4, 'largeur_petale': 0.2, 'espece': 'setosa'},  
...  
  
k=10  
longueur=2.5  
largeur=0.75  
  
>>> knn(iris,k,longueur,largeur)  
[(0.6103277807866851, 'versicolor'), (0.6946221994724903, 'setosa'),  
(0.8139410298049854, 'setosa'), (0.8381527307120104, 'versicolor'),  
(0.8381527307120104, 'versicolor'), (0.8381527307120106, 'setosa'),  
(0.873212459828649, 'setosa'), (0.9124143795447328, 'setosa'),  
(0.9178779875342911, 'setosa'), (0.9656603957913982, 'setosa')]
```

### 5.3 Fonction décision



#### Exercice 5

Il reste maintenant à attribuer à  $B$  (*longueur ; largeur*) la classe qui est la plus fréquente parmi les  $k$  éléments les plus proches.

1. On cherche à écrire une fonction **decision(liste\_dico,k,longueur,largeur)** qui renvoie la classe attribuée à l'élément  $B$  (*longueur ; largeur*).
2. Tester votre fonction avec différentes valeurs de  $k$  et proposer une classe pour Iris1(2 ; 0,5) et pour Iris2(2,5 ; 0,75).  
Justifier votre choix.

```
def decision(liste_dico,k,longueur,largeur):  
    '''In : liste de dictionnaires (ici iris), le paramètre k  
        et la longueur et largeur de l'iris à tester  
        Out : l'espèce attribuée'''  
    L=knn(liste_dico,k,longueur,largeur)  
    ...
```

```
k=10  
longueur=2.5  
largeur=0.75  
  
>>> knn(iris,k,longueur,largeur)  
[(0.6103277807866851, 'versicolor'), (0.6946221994724903, 'setosa'),  
(0.8139410298049854, 'setosa'), (0.8381527307120104, 'versicolor'),  
(0.8381527307120104, 'versicolor'), (0.8381527307120106, 'setosa'),  
(0.873212459828649, 'setosa'), (0.9124143795447328, 'setosa'), (0.91787798  
75342911, 'setosa'), (0.9656603957913982, 'setosa')]
```

```
k=10  
longueur=2.5  
largeur=0.75  
  
>>> decision(iris,k,longueur,largeur)  
'setosa'
```



#### Aide

Quelques éléments pour trier des listes et des dictionnaires sur la page 8.

## 6 Quelques éléments utiles

### 6.1 Pour trier une liste L de tuples par rapport à un élément

On utilise la fonction **sorted** et une clé. Cette fonction renvoie une liste triée.

— trier par rapport au 1er élément : **sorted(L, key=lambda x : x[0])**

```
>>> liste_tuples=[(2.5, 'Marc'), (12, 'Bernard'), (100, 'Carole')]
>>> L0=sorted(liste_tuples, key=lambda t: t[0])

[(2.5, 'Marc'), (12, 'Bernard'), (100, 'Carole')]
```

— trier par rapport au 2e élément : **sorted(L, key=lambda x : x[1])**

```
>>> liste_tuples=[(2.5, 'Marc'), (12, 'Bernard'), (100, 'Carole')]
>>> L1=sorted(liste_tuples, key=lambda t: t[1])

[(12, 'Bernard'), (100, 'Carole'), (2.5, 'Marc')]
```

### 6.2 Pour trier un dictionnaire dico par rapport aux clés ou aux valeurs

Pour trier un objet dict, il suffit d'utiliser la fonction **sorted** et une clé de tri.

Cette fonction retourne une liste contenant les valeurs triées.

Dans le cas d'un objet dictionnaire, les données (clés + valeurs) sont converties en tuple.

— trier par rapport aux clés : **sorted(dico.items(), key=lambda x : x[0])**

```
>>> dico={'Pierre':50, 'Anatole':150, 'Zaina':75}
>>> L0=sorted(dico.items(), key=lambda t: t[0])

[( 'Anatole', 150), ( 'Pierre', 50), ( 'Zaina', 75)]
```

— trier par rapport aux valeurs : **sorted(sorted(dico.items()), key=lambda x : x[1])**

```
>>> dico={'Pierre':50, 'Anatole':150, 'Zaina':75}
>>> L1=sorted(dico.items(), key=lambda t: t[1])

[( 'Pierre', 50), ( 'Zaina', 75), ( 'Anatole', 150)]
```

## 7 Représentation Graphique



### Exercice 6

| Représenter sur votre graphique, un cercle centré sur l'iris à tester, contenant les *k* plus proche voisins.

↩️ Fin du TD 🏹