TP7 - Algorithmes d'apprentissage supervisé
-
Les k plus proches voisins

I. Introduction sur les algorithmes pour l'intelligence artificielle

Sous le terme d'intelligence artificielle se cachent un ensemble de techniques permettant à des machines d'accomplir des tâches et de résoudre des problèmes normalement réservés aux humains, comme par exemple reconnaître et localiser des objets dans une image. Depuis quelques années, on associe presque toujours l'intelligence aux capacités d'apprentissage : c'est grâce à l'apprentissage qu'un système intellignet capable d'exécuter une tâche peut améliorer ses performances avec l'expérience.

On distingue deux types d'apprentissage :

- l'*apprentissage supervisé*. Il consiste, pour un opérateur, à montrer à la machine des milliers voire des millions d'exemples étiquetés avec leur catégorie, qui permettront à la machine de déterminer elle-même les paramètres pertinents pour classer chaque objet dans la catégorie qui lui correspond. Une fois cette phase d'apprentissage terminée, la machine doit être capable de généraliser à des objets pas encore vus.
- l'*apprentissage non supervisé*. Il consiste à faire en sorte qu'à partir d'un ensemble de données, la machine soit capable de créer ses propres catégories, et si possible, que ces catégories soient pertinentes pour nous. Ce modèle est plus proche de notre propre modèle d'apprentissage, basé sur l'observation.

Dans ce TP, on s'intéresse à un exemple d'algorithme d'apprentissage supervisé : l'algorithme des k plus proches voisins.

II. Un premier exemple d'apprentissage supervisé

Nous allons d'abord considérer le jeu de données iris de Fisher. En 1936, Edgar Anderson a collecté des données sur trois espèces d'iris : *iris setosa*, *iris virginica* et *iris versicolor*.

Ce jeu de données est composé de 150 entrées. Pour chaque entrée, est fourni :

- la longueur des sépales (en cm),
- la largeur des sépales (en cm),
- la longueur des pétales (en cm),
- la largeur des pétales (en cm),
- l'espèce d'iris.

Ces 150 données seront les données d'entraînement que nous fournirons à la machine pour l'apprentissage supervisé.

II.1. Visualisation du jeu de données

On souhaite tout d'abord visualiser le jeu de données.

- Importer la librairie `matplotlib.pyplot` sous l'alias `plt` et la librairie `sklearn`.
- Cette seconde librairie contient les jeux de données utilisés dans ce TP, notamment celui des iris de Fisher. Pour le récupérer, on utilise la syntaxe suivante.

```
from sklearn import datasets
iris = sklearn.datasets.load_iris()
```

- À l'aide de la commande `iris.keys()`, déterminer ce que renvoie les commandes suivantes : `iris['target']` et `iris['data']`.

  - La commande `iris['target']` renvoie un tableau contenant en $i^{\text{ème}}$ position :
    - × 0 si l'espèce de la $i^{\text{ème}}$ fleur est *setosa*,
    - × 1 si l'espèce de la $i^{\text{ème}}$ fleur est *versicolor*,
    - × 2, si l'espèce de la $i^{\text{ème}}$ fleur est *virginica*.

    On peut lire à quelle espèce correspond quel chiffre grâce à la commande `iris['target_names']`.
  - La commande `iris['data']` renvoie un tableau contenant en $i^{\text{ème}}$ position une liste à 4 éléments.
    - × Le 1^{er} élément correspond à la longueur des sépales de la $i^{\text{ème}}$ fleur en cm.
    - × Le $2^{\text{ème}}$ élément correspond à la largeur des sépales de la $i^{\text{ème}}$ fleur en cm.
    - × Le $3^{\text{ème}}$ élément correspond à la longueur des pétales de la $i^{\text{ème}}$ fleur en cm.
    - × Le $4^{\text{ème}}$ élément correspond à la largeur des pétales de la $i^{\text{ème}}$ fleur en cm.

    On peut lire à quoi correspond chaque élément de chaque liste grâce à la commande `iris['feature_names']`.

- On peut donc stocker dans une liste `x` la longueur des pétales de chaque fleur et dans une liste `y` la largeur des pétales de chaque fleur.

```
n = len(iris['data'])
x = [iris['data'][i][2] for i in range(n)]
x = [iris['data'][i][3] for i in range(n)]
```

Quelle commande permet alors de stocker dans une variable `lab` la liste contenant en $i^{\text{ème}}$ élément l'espèce de la $i^{\text{ème}}$ fleur.
On pourra utiliser un dictionnaire.

```
dico = {0 : 'setosa', 1 : 'versicolor', 2 : 'virginica'}
lab = [dico[iris['target'][i]] for i in range(n)]
```

- On peut alors visualiser le jeu de données à l'aide d'un nuage de points.

```
x1 = [x[i] for i in range(n) if lab[i] == 'setosa']
y1 = [y[i] for i in range(n) if lab[i] == 'setosa']
x2 = [x[i] for i in range(n) if lab[i] == 'versicolor']
y2 = [y[i] for i in range(n) if lab[i] == 'versicolor']
x3 = [x[i] for i in range(n) if lab[i] == 'virginica']
y3 = [y[i] for i in range(n) if lab[i] == 'virginica']

plt.clf()
plt.scatter(x1, y1, color = 'g', label = 'setosa')
plt.scatter(x2, y2, color = 'b', label = 'versicolor')
plt.scatter(x3, y3, color = 'r', label = 'virginica')
plt.legend()
```

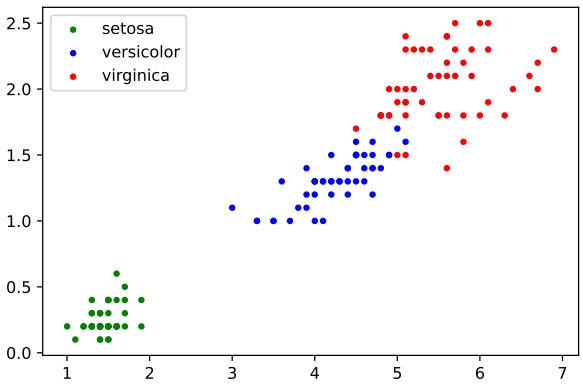


Fig. 1 Données d'entraînement

On peut remarquer que les points sont regroupés en *cluster* correspondant chacun à une espèce différente. Il semble que le nuage *setosa* est isolé, tandis que les deux nuages *versicolor* et *virginica* ont un peu tendance à se mélanger.

II.2. Classification par algorithme des k plus proches voisins

Une fois ces données acquises, supposons que nous ayons trouvé une iris (une nouvelle donnée) et que, n'étant pas spécialiste, nous souhaitions en déterminer l'espèce.

Pour cela, on mesure la longueur et la largeur des pétales de cet iris. Mettons, pour l'exemple, que l'on trouve :

× longueur des pétales : $x^* = 4,96$ cm

× largeur des pétales : $y^* = 1,59$ cm

On place ensuite le point sur la figure précédente.

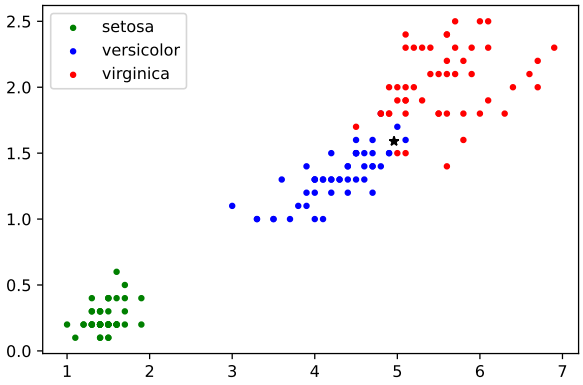


Fig. 2 Donnée à classer

Pour classer ce point, on peut utiliser l'algorithme dit des k **plus proches voisins** (en anglais : k *nearest neighbours* - **kNN**). Il consiste à déterminer les k données d'entrainement les plus proches du point (x^*, y^*), et à attribuer à ce point la catégorie (ici l'espèce) majoritaire parmi ces voisins.

Plus précisément :

1) on code une fonction `distance` qui permettra de calculer les distances entre la donnée (x^*, y^*) et chaque donnée d'entrainement.

2) on détermine les k données du jeu d'entrainement qui sont les plus proches de (x^*, y^*) pour la distance définie précédemment.

3) on attribue à (x^*, y^*) la catégorie majoritaire parmi les k voisins sélectionnés à l'étape précédente.

► Quelle commande permet d'obtenir, par compréhension, la liste des tuples contenant la longueur, la largeur des pétales et l'espèce de chaque fleur ?

```
donnees = [(x[i], y[i], lab[i]) for i in range(n)]
```

- Proposer une fonction `distance` qui prend en argument deux tuples `T1` et `T2` et renvoie la distance euclidienne entre ces tuples.

```python
import numpy as np

def distance(T1, T2) :
    '' distance(T1 : tuple, T2 : tuple) -> float''
    return np.sqrt( (T2[0] - T1[0])**2 + (T2[1] - T1[1])**2 )
```

- Stocker dans la variable `u` le tuple `(4.96, 1.59)`. Quelle commande permet de stocker dans une variable `dist` la liste pour laquelle le $i^{\text{ème}}$ élément est la liste contenant le tuple des longueur largeur des pétales et l'espèce de la $i^{\text{ème}}$ fleur, et la distance de ce tuple à `u`.

```python
u = (4.96, 1.59)
dist = [[donnees[i], distance(donnees[i], u)] for i in range(n)]
```

- Écrire une fonction `tri_fusion` prenant en argument une liste `L` de même type que la liste `distance` définie ci-dessus et renvoyant cette liste triée de l'élément contenant la plus petite distance à celui contenant la plus grande.

```python
def fusion(L1, L2) :
    ''fusion(L1 : list, L2 : list) -> list''
    if L1 == [] :
        return L2
    elif L2 == [] :
        return L1
    elif L1[0][1] < L2[0][1] :
        return [L1[0]] + fusion(L1[1:], L2)
    else :
        return [L2[0]] + fusion(L1, L2[1:])

def tri_fusion(L) :
    '' tri_fusion(L : list) -> list''
    p = len(L) :
    if p <= 1 :
        return L
    else :
        m = p // 2
        return fusion(tri_fusion(L[:m]), tri_fusion(L[m:]))
```

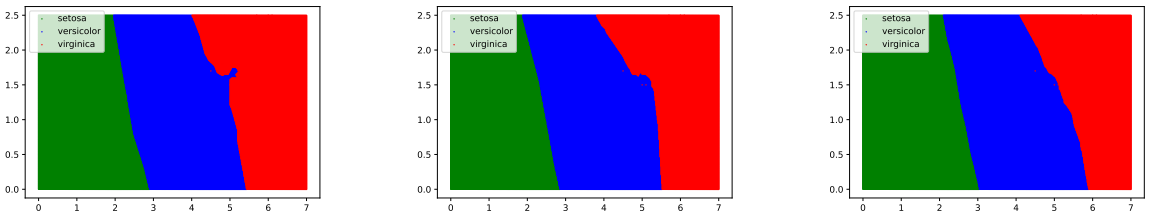
- Proposer une fonction `kPlusProches` qui prend en argument une liste de tuples `T`, un tuple `u` et un entier `k`, et qui renvoie la liste des `k` tuples de `T` les plus proches de `u` pour la distance euclidienne avec la distance associée.

```
def kPlusProches(T, u, k) :
    ''kPlusProches(T : list(tuple), u : tuple, k : int) -> list[int]''
    n = len(T) :
    dist = [ [T[i], distance(T[i], u)] for i in range(n) ]
    return tri_fusion(dist)[:k]
```

- Écrire enfin une fonction `kNN` prenant en paramètre un jeu de données `T` sous forme de liste de triplets, un couple `u` et un entier `k`, et renvoyant la catégorie affectée à la donnée `u` par algorithme des k plus proches voisins.

```
def kNN(T, u, k) :
    ''kNN(T : list(tuple), u : tuple, k : int) -> str''
    voisins = kPlusProches(T, u, k)
    dico = {'setosa' : 0, 'versicolor' : 0, 'virginica' : 0}
    for v in voisins :
        dico[v[0][2]] += 1
    m = 'setosa'
    for c in dico.keys() :
        if dico[c] > dico[m] :
            m = c
    return m
```

Avec la fonction précédente, nous pouvons attribuer, à tout point de $[0, 7] \times [0, 2.5]$, une catégorie grâce à l'algorithme kNN. On obtient les figures suivantes pour différentes valeurs de k.



$k = 3$ $k = 7$ $k = 20$

Fig. 3 Classifications