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']`.

- 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.

- 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()
```

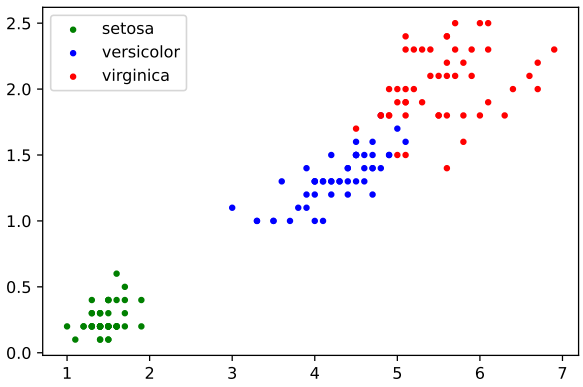


Fig. 1 Données d'entraînement

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.

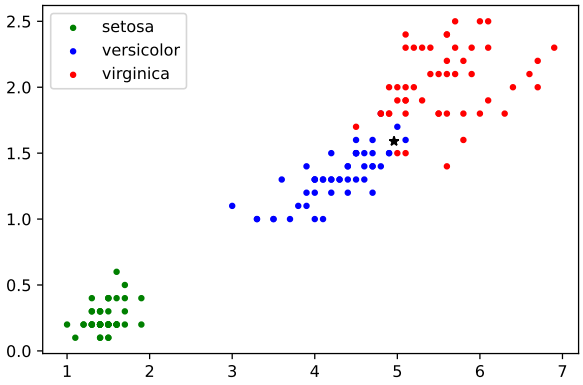


Fig. 2 Donnée à classer

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 ?

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

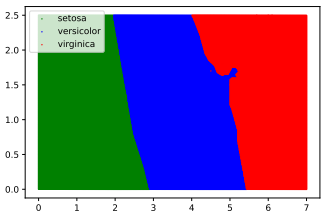
- Stocker dans la variable `u` le tuple `(4.96, 1.59)`. Quelle commande permet de stocker dans une variable `dist` la liste pour laquelle le $i^{\text{ème}}$ élément est la liste contenant le tuple des longueur largeur des pétales et l'espèce de la $i^{\text{ème}}$ fleur, et la distance de ce tuple à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

- 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.

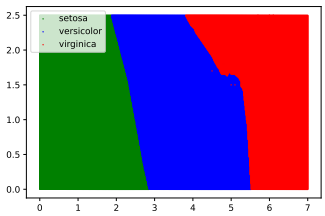
- Proposer une fonction `kPlusProches` qui prend en argument une liste de tuples `T`, un tuple `u` et un entier `k`, et qui renvoie la liste des `k` tuples de `T` les plus proches de `u` pour la distance euclidienne avec la distance associée.

- 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.

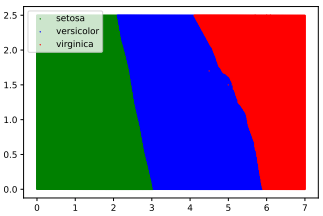
Avec la fonction précédente, nous pouvons attribuer, à tout point de $[0, 7] \times [0, 2.5]$, une catégorie grâce à l'algorithme kNN. On obtient les figures suivantes pour différentes valeurs de k.



$k = 3$



$k = 7$



$k = 20$

Fig. 3 Classifications

II.3. Matrice de confusion

On cherche maintenant à évaluer la qualité de notre prédictions.
Pour cela, on sépare notre jeu de données en deux sous-ensembles :

× un sous-ensemble du jeu de données qui servira toujours à l'apprentissage,

× un sous-ensemble du jeu de données qui servira à tester la qualité de la prédiction fournie par l'algorithme kNN.

▶ Exécuter le code suivant.

```
import random as rd

x_test = []
y_test = []
m = 50 # taille du jeu de test
for i in range(n) :
    j = rd.randint(0, len(x))
    x_test.append(x.pop(j))
    y_test.append(y.pop(j))
    lab_test.append(lab.pop(j))

d_app = [ (x[i], y[i], lab[i]) for i in range(n-m) ] # jeu d'apprentissage
d_test = [ (x_test[i], y_test[i]) for i in range(m) ] # jeu de test
k = 3 # nombre de voisins choisi pour l'algorithme kNN
```

▶ Désormais, chaque donnée du jeu de test possède deux catégories :

× sa catégoriée réelle,

× sa catégorie prédite.

Pour déterminer la qualité de prédiction de l'algorithme kNN, on construit alors une matrice A dite **matrice de confusion** de coordonnées $(a_{i,j})$ définies par :

> La coordonnée $a_{i,j}$ de la matrice de confusion contient le nombre de données de catégorie réelle numéro i et de catégorie prédite numéro j

Notons que le nombre de données bien classées est donc $\mathrm{tr}(A)$ (et le nombre de points mal classés : $n - \mathrm{tr}(A)$).
Le **risque empirique** $R(k)$ de la méthode des k plus proches voisins est alors défini comme le fréquence de points mal classés par l'algorithme. Autrement dit :

$$R(k) = \frac{n - \mathrm{tr}(A)}{n}$$

- Écrire un script permettant d'obtenir la matrice de confusion de l'algorithme kNN pour les jeux d'apprentissage et de test précédents.

- Exécuter votre script pour différentes valeurs de k. Que remarque-t-on ?

III. Un deuxième exemple d'utilisation des k plus proches voisins

Dans ce deuxième exemple, on souhaite utiliser l'algorithme kNN pour reconnaître des caractères. Pour cela, nous allons utiliser la base de données MNIST (*Mixed National Institute of Standards and Technology*) founie par la librairie `sklearn`. Il s'agit d'une base de données de chiffres manuscrits. Il a été initialement utilisé pour la reconnaissance de code postaux.

- On commence par importer la base de données.

```
chiffres = sklearn.datasets.load_digits()
```

- La variable `chiffres` est un dictionnaire. On peut donc accéder à ses clés avec la commande suivante : `chiffres.keys()`. On remarque que :
  - la commande `chiffres['target']` renvoie un tableau contenant en $i^{\text{ème}}$ position le chiffre associé à l'image i,
  - la commande `chiffres['images'][j]` renvoie un tableau représentant une matrice 8×8 de pixels. Chaque coordonnée (chaque pixel) est un entier entre 0 (blanc) et 16 (noir).

  On peut visualiser la `j`$^{\text{ème}}$ matrice de pixels grâce à la commande suivante :

```
plt.imshow(chiffres['images'][j], cmap = plt.cm.gray_r)
```

- Quelle commande permet de stocker dans une variable `n` le nombre de chiffres manuscrits dans la base de données ?

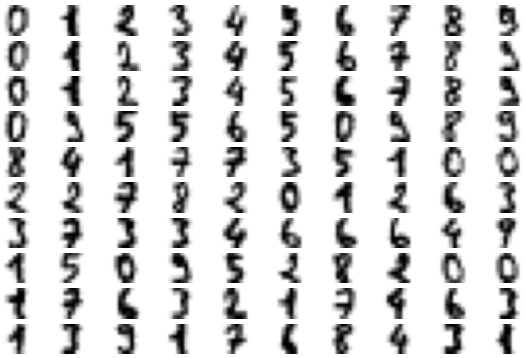


Fig. 4 Extrait de la banque de données

- Nous allons maintenant partager ces données en deux parties sensiblement égales :
  - × la première partie sera consacrée à l'apprentissage,
  - × la seconde partie sera consacrée au test.

  Pour former la partie « apprentissage » du jeu de données, on va choisir 90 représentants de chacune de nos dix classes (les 10 chiffres de 0 à 9). Il restera donc 897 données dans la partie test du jeu de données.

```
X = chiffres['images']
Y = chiffres['target']

d_app = []
d_test = []

nb = [0] * 10
for k in range(n) :
    if nb[Y[k]] < 90 :
        d_app.append(k)
        nb[Y[k]] += 1
    else :
        d_test.append(k)
```

  On souhaite maintenant coder l'algorithme des k plus proches voisins en suivant les 3 étapes décrites en partie précédente.

- On commence par choisir une distance. Pour ce jeu de données, nous devons utiliser une distance entre matrices. Pour cela, nous choisirons la distance euclidienne entre matrice. Coder alors une fonction `distance_m` prenant en argument deux matrices `M1` et `M2`, et renvoyant la distance euclidienne entre ces deux matrices.

- Écrire ensuite une fonction `kVoisins` qui prend en argument une liste `D` de couples (matrice, chiffre), une matrice `u` et un entier `k`, et qui renvoie la liste des `k` matrices de `D` les plus proches de `u` pour la distance définie ci-dessus, avec la distance associée.
  On pourra utiliser la fonction `tri_fusion` *définie précédemment.*

- Écrire une fonction `kPP` prenant en paramètre un jeu de données `D` sous forme de liste de couples (matrice, chiffre), une matrice `u` et un entier `k`, et renvoyant le chiffre affecté à la donnée `u` par l'algorithme des `k` plus proches voisins.

- Proposer enfin un script permettant de compter le nombre d'erreurs effectuées par l'algorithme avec les jeux d'apprentissage et de test définis plus haut.

Pour $k = 3$, les erreurs que la machine a commise sont présentées ci-dessous.

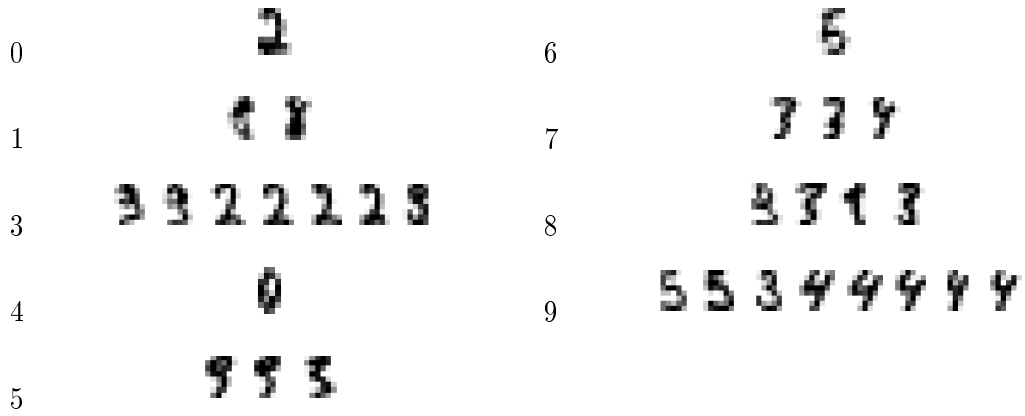


Fig. 5 Erreurs commises par la machine

Quelles améliorations envisager ?

La méthode des k plus proches voisins peut être améliorée de plusieurs façons. Par exemple :

- au lieu de considérer les k plus proches voisins, on pourrait considérer tous les voisins contenus dans une boule de rayon ε centrée autour de la donnée à classer ;
- on pourrait pondérer la contribution de chaque voisin par un coefficient inversement proportionnel à sa distance de la donnée à classer ;
- on pourrait utiliser d'autres distances que la distance euclidienne.