Algorithme des k plus proches voisins

Initiation au Machine Learning

1. Introduction

- Objectif de l’intelligence artificielle : rendre des machines capables de résoudre des problèmes complexes.
- Deux approches de l’IA :
  - top-down : formaliser des problèmes et concevoir des méthodes intelligentes pour les résoudre.
  - bottom-up : apprendre à partir des cas pratiques et généraliser.

- Situé dans cette deuxième catégorie, l’algorithme des k plus proches voisins("k nearest neighbors" en anglais, knn en abrégé) est un des plus anciens mais aussi des plus efficaces de l’intelligence artificielle, plus précisément de l’apprentissage automatique.

2. Présentation de l'algorithme

L'algorithme des k plus proches voisins est très répandu notamment :

- Dans les systèmes de recommandationsaux abonnés de sites Internet(livres, musiques, vidéos, etc...) .
- Dans la reconnaissance de formes, de caractères manuscrits, d'images...

Un exemple emblématique de données : les iris

Afin de travailler sur un exemple, nous allons utiliser un jeu de données très connu dans le monde du machine learning : le jeu de données des fleurs "iris".En 1936, le botaniste américain, Edgar Anderson a collecté des données sur 3 espèces d'iris : "iris setosa", "iris virginica" et "iris versicolor"

iris setosa

iris versicolor

iris virginica

Il les a classées à l'aide de plusieurs caractéristiques :

- La longueur des sépales (en cm)
- La largeur des sépales (en cm)
- La longueur des pétales (en cm)
- La largeur des pétales (en cm)

Ces données sont ici stockées dans un fichier au format `.csv` , dont voici les deux premières lignes :

```
5.1,3.5,1.4,0.2,Iris-setosa
4.9,3.0,1.4,0.2,Iris-setosa
```

Chaque ligne de données est composée des attributs, séparés par une virgule. Dans cet exemple, chaque donnée a donc 5 attributs, les 4 premiers sont des nombres, le dernier est par convention l'étiquette de la donnée(label).

Par souci de simplification :

- Nous nous intéressons uniquement à la largeur("width") et à la longueur("length") des pétales.
- Les étiquettes sont raccourcies :
  - "Iris-setosa" est remplacé par `setosa`
  - "Iris-virginica" par `virginica`
  - "Iris-versicolor" par `versicolor`

Question : en trouvant un iris, comment identifier son espèce ?

Exercice 1:

1. Ouvrir le fichier iris.csv (iris.csv) avec un tableur.
2. Combien de colonnes contient ce fichier ?
3. Combien de lignes contient ce fichier ?

	A	B	C
1	petal_length	petal_width	species
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
7	1.7	0.4	setosa
8	1.4	0.3	setosa
9	1.5	0.2	setosa
10	1.4	0.2	setosa
11	1.5	0.1	setosa
12	1.5	0.2	setosa

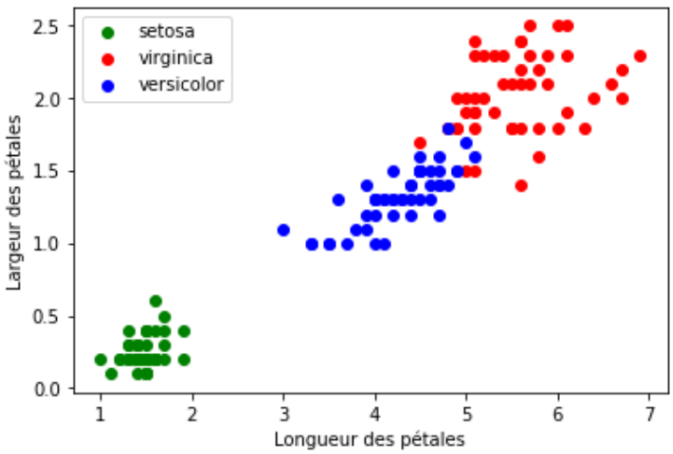
Réponse :

1.
2.
3.

Principe de l'algorithme

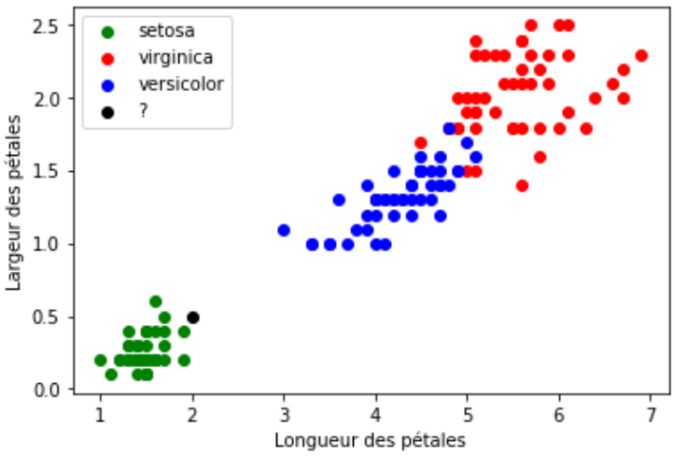
Voici une réprésentation de ce jeu de données :

- En abscisse : la longueur des pétales
- En ordonnée : la largeur des pétales



On trouve un nouvel iris dont dont la longueur des pétales est 2 cm et la largeur 0,5 cm.

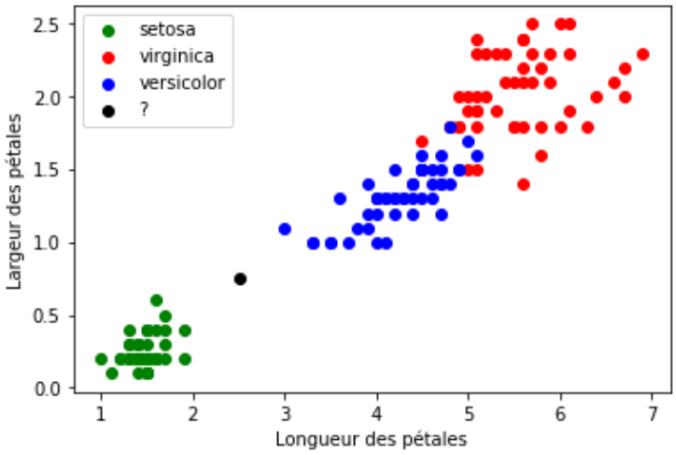
En plaçant le point de coordonnées $(2; 0,5)$, on constate qu'il y a de fortes chances que cet iris soit "iris-setosa"



On trouve un nouvel iris dont dont la longueur des pétales est 2,5 cm et la largeur 0,75 cm.

En plaçant le point de coordonnées $(2,5; 0,75)$, on constate qu'il est plus difficile de prendre une décision.

C'est l'algorithme des k plus proches voisins qui va prendre la décision.

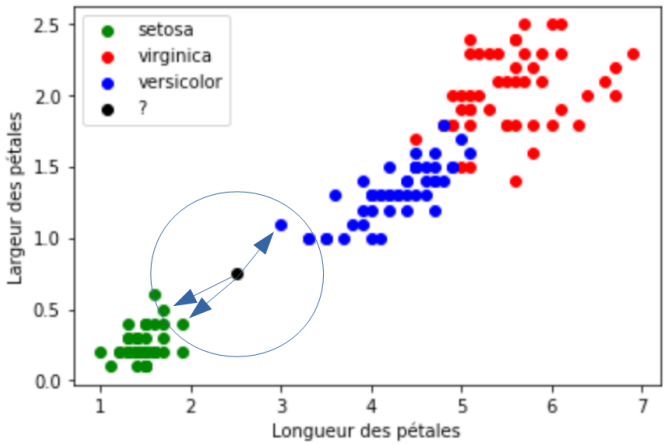


La valeur de k est le nombre de plus proches voisins(en terme de distance) avec lesquels le nouvel iris sera comparé.

Dans l'exemple ci-contre, $k = 3$, les trois plus proches voisins sont indiqués par des flèches.

Parmi ces trois voisins, deux sont étiquettés "setosa" et un seul est étiquetté "versicolor".

L'algorithme des 3 plus proches voisins choisira d'étiquetter ce nouvel iris comme "setosa".



Rappel avant de continuer :

- La distance dont il s'agit ici est la distance euclidienne dans un repère orthonormé entre deux points $A(x_A;y_A)$ et $B(x_B;y_B)$:

$$AB=\sqrt{(x_B-x_A)^2+(y_B-y_A)^2}$$

3. Programmation de l'algorithme

Import des données depuis le fichier

Exercice 2 :

A l'aide de la bibliothèque `csv` Le programme ci-dessous stocke les données du fichier dans une liste de listes. Le jeu de données est mémorisé dans la variable `dataset` .Exécuter cette cellule et observer les affichages.

1. Qu'est ce qui est affiché ?
2. Qu'est-ce que `A` ?

In [ ]:

```
import csv



def lirefichier(nomfichier):
    fichierCSV = open(nomfichier,"r")
    lignes = csv.reader(fichierCSV)
    next(lignes)
    data = list(lignes)

    for i in range(len(data)):
        for j in range(len(data[i])-1):
            data[i][j] = float(data[i][j])

    return data

A=(2.5,0.75)
dataset=lirefichier('iris.csv')
print(dataset[0])
```

Réponses:

1.
2.

Calculer une distance d'un point à un autre

Exercice 2 :

Compléter la fonction `distance(A,B)` qui prend en paramètres deux tuples contenant les coordonnées de deux iris et qui renvoie leur distance.Ls fonction racine carrée est importée du module `math` (fonction `sqrt()`).

Ainsi avec `A=(2.5,0.75)` et `B=(1.4,0.2)` , `distance(A,B)` doit renvoyer `1.2298373876248845` .

In [ ]:

```
#fonction distance
from math import *

def distance(A,B):
    xA,yA=A
    xB,yB=B
    return "A compléter"

B=(dataset[0][0],dataset[0][1])
print(A)
print(B)
print(distance(A,B))
```

Calculer la distance d'un point à chacun des points du dataset.

Exercice 3 :

Compléter la fonction `distances(A,dataset)` :

- Entrées :
  - Un tuple `A` contenant les coordonnées d'un iris.
  - La liste `dataset` des données
- Sortie: Une liste triée par valeur croissante dont chaque élément est un tuple qui contient:
  - Une distance à `A` (de type float)
  - L'indice de l'élément de `dataset` par rapport auquel la distance est calculée

Ainsi, le premier tuple de cette liste doit contenir le voisin le plus proche de l'iris A. L'instruction `print(distances(A, dataset)[0])` doit renvoyer `(0.6103277807866851, 98)` (Ce qui signifie que l'iris le plus proche de `A` est le 99ème iris du jeu de données).

In [ ]:

```
#liste des distances d'un point A aux points du dataset
#renvoie la liste des indices et de la distance à A triée par distance croissante

def distances(A,dataset):
    resultat = []
    for i in range(len(dataset)):
        point=("A compléter","A compléter")
        resultat.append("A compléter")
    return sorted(resultat)

print(distances(A, dataset)[0])
```

Déterminer les étiquettes des trois plus proches voisins.

Exercice 4 :

Compléter la fonction `kppv(A,dataset)` :

- Entrées :
  - Un tuple `A` contenant les coordonnées d'un iris.
  - La liste `dataset` des données
- Sortie: Une liste contenant les étiquettes des trois plus proches voisins de A

Ainsi, l'instruction `print(kppv(A,dataset))` doit renvoyer `['versicolor', 'setosa', 'setosa']`
(Ce qui correspond bien au schéma avec les flèches dans la partie précédente).

In [ ]:

```
#renvoie les étiquettes des 3 plus proches voisins
def kppv(A,dataset):
    resultats = distances(A,dataset)
    voisins=[]
    for i in range(3):
        voisins.append("A compléter")

    return voisins


print(kppv(A,dataset))
```

Identifier l'étiquette majoritaire parmi les voisins.

Exercice 5 :

Compléter la fonction `decompte(A,dataset)` :

- Entrées :
  - Un tuple `A` contenant les coordonnées d'un iris.
  - La liste `dataset` des données
- Sortie: Un dictionnaire dont les clés sont les étiquettes des iris et les valeurs le nombre de voisins correspondant à chaque étiquette

Ainsi, l'instruction `print(decompte(A, dataset))` doit renvoyer `{'versicolor': 1, 'setosa': 2}`

In [ ]:

```
#decompte des kppv dans un dictionnaire

def decompte(A,dataset):
    voisins=kppv(A,dataset)
    resultats={}
    for etiquette in voisins:
        if etiquette in resultats :
            resultats[etiquette]="A compléter"
        else:
            resultats[etiquette]="A compléter"

    return resultats

print(decompte(A, dataset))
```

Exercice 6 :

Compléter la fonction `prediction(A,dataset)` :

- Entrées :
  - Un tuple `A` contenant les coordonnées d'un iris.
  - La liste `dataset` des données
- Sortie: L'étiquette majoritaire des voisins de A.

Ainsi, l'instruction `prediction(A,dataset)` doit renvoyer `'setosa'` .

In [ ]:

```
#prediction de l'algorithme pour A
def prediction(A,dataset):
    resultats= decompte(A, dataset)
    nmax=0
    iris=''
    for etiquette in resultats:
        if resultats[etiquette] > nmax:
            nmax = "A compléter"
            iris = "A compléter"
    return iris

print(prediction(A,dataset))
```

Exercice 8 :

Utiliser l'algorithme des 3 plus proches voisins pour déterminer l'espèce des iris suivants :

1. C=(5.2,1.2)
2. D=(4,2)
3. E=(4.8,1.6)

In [ ]:

```
#réponses
C=(5.2,1.2)
D=(4,2)
E=(4.8,1.7)

#1.

#2.

#3.
```

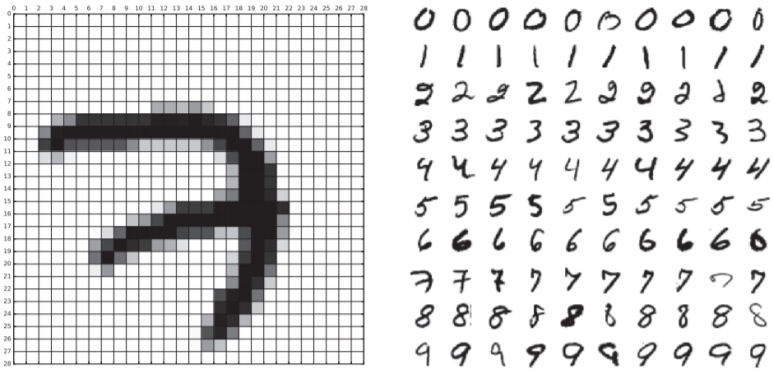
Remarques

- Dans la pratique :
  - Afin d'utiliser cet algorithme au mieux, on sépare les données en deux jeux :
    - Le jeu d'entrainement : c'est à partir de ces données que l'on fait fonctionner l'algorithme
    - Le jeu de test : on utilise l'algorithme pour étiqueter les valeurs de ce jeu, comme si on ne les connaissait pas et on compare ensuite avec la réalité pour déterminer le pourcentage d'erreur.
  - On fait varier le nombre de voisins (la valeur de k) et on identifie celle pour laquelle l'algorithme donne le plus petit pourcentage d'erreur. Dans le cas des iris, cette valeur est 4.
- Bien que très efficace, cet algorithme a un coût mémoire important (il doit mémoriser toutes les données).

4. Machine learning avec python : scikit-learn

La bibliothèque `scikit-learn` permet de travailler avec des algorithmes d'apprentissage automatique. L'algorithme des k plus proches voisins est présent parmi d'autres dans cette bibliothèque.

Dans cette partie , l'algorithme des k plus proches voisins sera utilisé avec un autre type de jeu de données célèbres : La base de données MNIST(Modified ou Mixed National Institute of Standards and Technology) qui est une base de données de chiffres écrits à la main.Elle est très utilisée pour tester et entraîner des algorithmes d'apprentissage automatique.



Ci-contre : Un exemple agrandi du chiffre 7, et quelques exemples du jeu de données MNIST. Chaque chiffre manuscrit est numérisé dans une image composée de $28 \times 28 = 784$ pixels.Chaque pixel est une valeur entre 0 (noir) et 255 (blanc).

Modules nécéssaires

Si l'éxécution de la cellule ci-dessous ne renvoie pas d'erreurs, passer à la suite. Sinon installer la(les) bibliothèque(s) depuis une console à l'aide de la commande `pip install nom_du_module` ou `nom_du_module` est le nom de la bibliothèque à installer.

In [ ]:

```
from sklearn import * #pour l'apprentissage automatique
from matplotlib import pyplot #pour la représentation graphique
from random import * #pour la génération de valeurs aléatoires
```

Import des données

Ce programme télécharge les données (122 Mo) de 70000 images de chiffes manuscrits.L'éxécution de cette cellule peut être assez longue (quelques dizaines de secondes).

Lorsque les données s'affichent, passer à la suite.

In [ ]:

```
# Source des données: https://www.openml.org/d/554
print('Données en cours de chargement...')
dataset = datasets.fetch_openml('mnist_784', version=1)
print(dataset)
```

Exercice 9 :

On observe dans l'affichage précédent que les données sont stockées dans un dictionnaire.

1. A quelle clé sont associées les valeurs des pixels d'une image ?
2. A quelle clé est associée l'étiquette(label) de l'image ?
3. Afficher les valeurs des pixels de la première image.
4. Afficher l'étiquette de la première image.

Réponses :

1.
2.

In [ ]:

```
#3.

#4.
```

Visualisation d'un exemple d'image

Exercice 10 :

La cellule ci-dessous génère l'affichage d'une image au hasard parmi les 70000 du jeu de données. Compléter ce programme en remplaçant "à compléter" par l'instruction qui convient, pour que l'étiquette de l'image s'affiche.

Ainsi, si le chiffre manuscrit est un cinq, la valeur 5 doit s'afficher en-dessous de l'image

In [ ]:

```
#On choisit une image au hasard
alea=randint(0,70000)
#conversion en liste de pixels
sample=list(dataset['data'][alea])
#conversion en 28 listes de 28 valeurs
sample=[sample[28*i:28*i+28] for i in range(28)]
#affichage de l'image
#pyplot.gray()
pyplot.matshow(sample)

pyplot.show()
#affichage de l'étiquette
print("à compléter")
```

Echantillon

Pour des raisons de temps et de mémoire, on prélève un échantillon de 21000 images. C'est cet échantillon qui sera utilisé par la suite. Exécuter cette cellule et lorsque "OK !" s'affiche, passer à la suite.

In [ ]:

```
print('Extraction en cours...')
x_junk,x_keep,y_junk,y_keep=model_selection.train_test_split(dataset['data'],dataset['t
arget'],test_size=0.3)#

dataset['data'],dataset['target'] = x_keep , y_keep
print('OK !')
```

Jeux de tests et d'entrainement

Lors de l'entrainement d'un algorithme de Machine Learning, la bonne pratique veut que l'on découpe notre jeu de données en jeu d'entrainement (Training Set) et jeu de test (Testing Set). Ainsi, nous pourrons tester les performances du modèle obtenu suite à l'entrainement de l'algorithme. Le test de performance se fait sur le jeu de test. L'algorithme des k plus proches voisins va prédire les étiquettes de chacune des images du jeu de test en utilisant les données du jeu d'entrainement.

En général, les proportions entraînement/test sont :

- Entraînement : 80 % des données
- Test : 20 % des données

La cellule ci-dessous génère un jeu de test et un jeu d'entraînement. Lorsque 'OK !' s'affiche, passer à la suite :

In [ ]:

```
#découpage du jeu de données
print('Découpage en cours...')
x_train,x_test,y_train,y_test=model_selection.train_test_split(dataset['data'],dataset[
'target'],test_size=0.2)#
print('OK !')
```

Exercice 11 :

Pour prédire une étiquette, l'algorithme des k plus proches voisins va donc pour chacune des images du jeu de test:

- Calculer la distance entre celle-ci et toutes celles du jeu d'entraînement
- Identifier les k plus proches.
- Trouver l'étiquette majoritaire parmi ces k plus proches voisins.

Mais qu'est-ce que la distance entre deux images ?

Réponse :

-
-
-
-
-
-
-
-

Trouver la meilleure valeur pour k

Le but de l'entraînement de l'algorithme est de trouver la valeur de k telle que le pourcentage d'erreur est le plus faible possible dans les prédictions. Pour ce faire on teste bien sûr différentes valeurs de k, mais dans la pratique, on fait aussi varier les jeux de tests et d'entraînement.

Le programme ci-dessous entraîne l'algorithme des k plus proches voisins pour $k = 4$ sur le jeu de test précédemment créé et affiche le pourcentage de réussite. L'éxécution peut être longue(quelques dizaines de secondes).

In [ ]:

```
print('Prédictions en cours...')
KNN = neighbors.KNeighborsClassifier(4) # on sélectionne l'algorithme des 4 plus proche
s voisins
KNN.fit(x_train, y_train)
reussite=KNN.score(x_test,y_test)

echec=1-reussite
print("Taux de réussite : ", reussite)
print("Taux d'échec : ",echec)
```

Exercice 12:

En réutilisant le programme précédent, compléter le programme ci-dessous:

- n est le nombre maximal de voisins
- Ce programme doit calculer et le taux de prédictions correctes sur le jeu de test précédemment créé pour chaque valeur de $k \leq n$.
- Tous ces résultats seront contenus dans la liste `resulats` .

Attention : il y a beaucoup de calculs à effectuer. Par conséquent l'éxécution de ce programme prend plusieures dizaines de minutes.

In [ ]:

```
#Réponse
resultats=[]
n=15




print(resultats)
```

Exercice 13 :

On suppose que la liste `resultats` contient les valeurs suivantes :

```
[0.9545238095238096, 0.9469047619047619, 0.9571428571428572, 0.9557142857142857,
0.9564285714285714, 0.954047619047619, 0.9528571428571428, 0.9511904761904761,
0.9514285714285714, 0.95, 0.949047619047619, 0.949047619047619, 0.9469047619047619,
0.9457142857142857, 0.9469047619047619]
```

1. Compléter les listes `X` et `Y` pour que le graphique affiche le pourcentage d'erreur en fonction du nombre de voisins.
2. Quelle valeur de k semble la plus adaptée au jeu de données étudié?

In [ ]:

```
#1.
resultats=[0.9545238095238096, 0.9469047619047619, 0.9571428571428572, 0.95571428571428
57, 0.9564285714285714, 0.954047619047619, 0.9528571428571428, 0.9511904761904761, 0.95
14285714285714, 0.95, 0.949047619047619, 0.949047619047619, 0.9469047619047619, 0.94571
42857142857, 0.9469047619047619]
X=[]
Y=[]
pyplot.xlabel('Nombre de voisins')
pyplot.ylabel("Pourcentage d'erreur")
pyplot.grid()
pyplot.plot(X,Y)
pyplot.show()
```

1.

Exemples de réussites et d'erreurs

L'algorithme des k plus proches voisins ne se trompe pas beaucoup.Le programme ci-dessous calcule la liste `erreurs` des indices des étiquettes mal prédites et la liste `exactes` des indices des étiquettes correctement prédites.L'éxécution de cette cellule peut être longue(quelques dizaines de secondes). Lorsque "OK !" s'affiche, passer à la suite

In [ ]:

```
#Liste qui contient les prédictions
print('Calculs en cours...')
prediction=KNN.predict(x_test)
#listes qui contiennent les erreurs d'étiquetage, les choix corrects
erreurs=[]
exactes=[]
i=0
while i < len(y_test):
    if prediction[i]!=y_test[i]:
        erreurs.append(i)
    else:
        exactes.append(i)
    i=i+1
print('OK !')
```

Exercice 14 :

Afficher le nombres d'erreurs d'étiquetage et le nombre de prédictions correctes.

In [ ]:

```
#Réponses
```

Le programme ci-desous affiche deux images au hasard parmi celles du jeu de test ainsi que leurs étiquettes prédites: Une mal étiquetée et une correctement étiquetée.

In [ ]:

```
#on choisit une erreur au hasard
f=choice(erreurs)
#liste des pixels de l'image choisie
image1=list(x_test[f])
#conversion en 28 listes de 28 valeurs
image1=[image1[28*i:28*i+28] for i in range(28)]

#on choisit une bonne prédiction au hasard
v=choice(exactes)
#liste des pixels de l'image choisie
image2=list(x_test[v])
#conversion en 28 listes de 28 valeurs
image2=[image2[28*i:28*i+28] for i in range(28)]

#affichages
#pyplot.gray()

pyplot.matshow(image1)
pyplot.title("Prédiction : "+ prediction[f], pad=20)

pyplot.matshow(image2)
pyplot.title("Prédiction : "+ prediction[v], pad=20)

pyplot.show()
```

FIN