

# TP

Apprentissage automatique

Algorithme des  $k$ -plus proches voisins

Algorithme des  $k$ -moyennes

PSI : Lycée Rabelais

## 1 Apprentissage supervisé et algorithme des $k$ -plus proches voisins

On s'intéresse au problème de classification abordé en cours. Il s'agit de distinguer différents types d'iris à partir de la longueur et de la largeur du pétale de la plante.

On considère donc une base de données contenant des mesures sur des fleurs d'iris qui ont été réalisées par des botanistes. Dans cette base de données, on retrouve les longueurs et largeurs des pétales et l'espèce d'iris associée : *setosa*, *versicolor* et *virginica*.

Longueur d'un pétale (cm)	Largeur d'un pétale (cm)	Espèce
1.4	0.2	<i>setosa</i>
4.4	1.2	<i>versicolor</i>
4.6	1.4	<i>versicolor</i>
5.1	1.8	<i>virginica</i>
...	...	...

**iris setosa**



petal sepal

**iris versicolor**



petal sepal

**iris virginica**



petal sepal

Dans cette première partie, on cherche à identifier de nouvelles iris avec la méthode des  $k$ -plus proches voisins.

**Question 1.** De quel type de problème d'intelligence artificielle s'agit-il ?

On notera  $X_i$ , la  $i$ -ème donnée d'entrée ayant ici deux caractéristiques (longueur et largeur mesurées sur un pétale). On cherche à prédire la sortie  $Y_i^{\text{préd}}$  qui correspond à l'espèce d'iris associée à l'entrée  $X_i$ .  $Y_i^{\text{préd}}$  prendra les valeurs 0 pour une iris *setosa*, 1 pour une *versicolor* et 2 pour une *virginica*.

On rappelle que l'algorithme des  $k$ -plus proches voisins s'appuie sur les deux étapes suivantes :

- 1 - Rechercher les  $k$  données "voisines" dont l'entrée  $X^{\text{data}}$  de la base de données est la plus proche de  $X_i$  ;

- 2 - Affecter à  $Y_i^{\text{préd}}$  la valeur du groupe majoritaire. Si pour une entrée, on compte parmi les  $k = 4$  (par exemple) plus proches voisins : 3 iris *versicolor*, 1 iris *virginica* et aucune *setosa*. Il faudra prédire *versicolor*, ce qui correspond à l'entier  $Y_i^{\text{préd}} = 1$ .

## 1.1 Création des données

La base de données que l'on souhaite utiliser est déjà enregistrée dans la bibliothèque sklearn de python. Pour y accéder, on écrira :

```
1 from sklearn import datasets
2 iris = datasets.load_iris()
3
4 Ytot = iris.target
5 donnees = iris.data
6 Xtot = donnees[:,2:4]
```

À ce stade :

- Ytot contient toutes les sorties sous la forme d'un tableau numpy rempli de 0, 1 ou 2 (type d'iris).
- Ces sorties sont associées aux entrées Xtot sous la forme d'un tableau numpy de 2 colonnes (longueur et largeur).

On écrira ensuite :

```
1 ## Préparation des données
2 from sklearn.model_selection import train_test_split
3 X_train, X_test, Y_train, Y_test = train_test_split(Xtot, Ytot, shuffle=True, test_size=0.33)
```

Ces lignes permettront de découper les données en :

- X\_train et Y\_train qui sont le tableau des entrées et le tableau des sorties pour la phase d'apprentissage.
- X\_test et Y\_test qui sont le tableau des entrées et le tableau des sorties pour la phase de test.

On rappelle que test\_size permet d'indiquer la proportion des données seront utilisées pour le test et pour l'apprentissage. Avec test\_size=0.33, 33% des données seront utilisées pour le test et 67% pour l'apprentissage. L'instruction shuffle = True permet de mélanger les données avant la séparation pour éviter d'utiliser une base de données déjà triée (ce qui est le cas ici).

**Question 2.** Écrire les lignes précédentes. Vérifier qu'il y a deux fois plus de données d'apprentissage que de données de test. Représenter et indiquer la signification des objets suivants sur votre feuille :

- X\_train, Y\_train, X\_test et Y\_test ;
- X\_train[9] et Y\_train[9] ;
- X\_train[9,0] et Y\_train[9,1] ;

**NOTA :** La notation X\_train[i,j] (qui est ici équivalente à X\_train[i][j]) est ici possible parce que X\_train est un tableau numpy. S'il s'agissait d'une liste de listes, il faudrait nécessairement écrire X\_train[i][j].

Pour représenter les données par un nuage de points, on pourra utiliser la fonction scatter de matplotlib. Il faudra donc écrire :

```
1 import matplotlib.pyplot as plt
2 plt.scatter(x,y,marker='*',color='blue')
3 plt.xlabel('nom x')
4 plt.ylabel('nom y')
```

Où :

- x et y sont respectivement l'abscisse et l'ordonnée des données.
- marker='\*' affiche des étoiles.

- `color='blue'` permet de sélectionner la couleur.
- `plt.xlabel('nom x')` et `plt.ylabel('nom y')` permet de définir le noms des axes.

**Question 3.** Afficher le nuage de points correspondant aux données d'apprentissage. On ne se souciera pas du type d'iris (même représentation pour toutes les iris).

**Question 4. BONUS.** Définir une fonction `affiche(X, Y)` qui affiche les données X (tableau numpy de 2 colonnes) sous forme de nuage de points en changeant de symbole ou de couleur en fonction de la valeur associée dans le vecteur Y.

## 1.2 k données les plus proches

Pour savoir quelles sont les données les plus proches d'une entrée X, il est nécessaire de calculer la distance entre cette entrée et chacune des données d'apprentissage.

**Question 5.** Écrire la fonction `d(Xi, Xdata)` qui permet de calculer la distance euclidienne entre les deux entrées Xi et Xdata (Xi et Xdata sont deux tableaux numpy de dimension 2).

**Question 6.** Tester la fonction avec `Xi = np.array([4.,4.])` et `Xdata = np.array([4.,4.])` puis avec `Xi = np.array([2.,2.])` et `Xdata = np.array([2.,4.])`.

On va maintenant créer une fonction qui renvoie un tableau T sous forme de liste de listes de taille `(len(X_train), 2)` contenant pour une donnée d'entrée Xi :

- la distance entre cette entrée Xi et les données d'apprentissage (Xdata représentant une des lignes du tableau X\_train).
- la sortie associée à la donnée d'apprentissage (Xdata) en question.

Avant de renvoyer ce tableau, on le triera dans l'ordre des distances croissantes avec la fonction `sorted` de Python. Pour trier une liste de listes, notée L, selon les données de la colonne d'indice i, on pourra utiliser, à titre d'exemple, les instructions suivantes :

```
1 | from operator import itemgetter ## pour importer le module itemgetter
2 | sorted(L, key=itemgetter(i))    ## trie la liste L selon les données rangées dans
3 |                               ## la colonne d'indice i
```

**Question 7.** Compléter les lignes ci-dessous pour définir la fonction qui permet de créer le tableau trié préalablement mentionné.

```
1 | from operator import itemgetter
2 | def creationTableau(Xi, X_train, Y_train):
3 |     ## Xi est une donnée d'entrée
4 |     ## X_train sera le tableau des entrées des données d'apprentissage
5 |     ## Y_train sera le tableau des sorties des données d'apprentissage
6 |
7 |     T = []
8 |     for i in range(0, len(X_train)):
9 |         ligne = [ ..... , ..... ]
10 |         .....
11 |     return .....
```

## 1.3 Identification

On cherche maintenant à écrire une fonction `identification(Xe, k, X_train, Y_train)` qui prédit, pour une entrée Xe quelconque (tableau numpy de dimension 2 contenant la largeur et la longueur d'un pétale), la sortie avec la méthode des k-plus proches voisins. Pour ce faire, on définit d'abord le tableau T puis on cherche quelles sont les espèces d'iris les plus représentées parmi les k premiers éléments de ce tableau.

Pour faire le comptage de l'iris la plus présente, on utilisera la liste `tabOccurence`. Cette liste sera d'abord initialisée à

[0,0,0] puis on incrémentera de 1 l'élément de la liste dont l'indice est celui du type d'iris rencontré dans le tableau T. Par exemple, si à la fin de la lecture des  $k = 4$  premières lignes du tableau T, tabOccurrence est égal à [0,3,1], on en conclura que que l'iris est de type 1 (*versicolor*).

**Question 8.** Compléter la fonction identification ci-dessous.

```

1 def identification(X,k,X_train,Y_train):
2     T = creationTableau(X,X_train,Y_train)
3     ## calcul des occurrences
4     n = 3 ## ici il y a 3 types d'iris
5     tabOccurrence = ..... ### initialisation de tabOccurrence
6     for i in range(0,k):
7         tabOccurrence[.....] += 1
8     ## recherche de l'indice ayant l'occurrence maxi
9     jm = 0 ## jm est l'indice du max
10    for j in range(1,n):
11        if ..... :
12            .....
13    return jm
14
15
16 Xe = [4.0,1.0] ## pour faire un test
17 print(identification(Xe,3,X_train,Y_train))

```

**Question 9.** Écrire les instructions permettant de créer la variable Y\_pred contenant l'ensemble des prédictions pour les entrées de la base de test.

**Question 10.** Écrire les instructions afin d'obtenir la matrice de confusion sous la forme d'une liste de liste.

**Question 11.** Écrire les instructions afin d'obtenir la justesse.

**Question 12.** Commenter la qualité de votre algorithme en analysant notamment l'influence de la valeur de  $k$ .

## 2 Apprentissage non-supervisé et algorithme des $k$ -moyennes

On considère maintenant le problème non-supervisé qui correspond au cas où l'espèce ne serait pas connue dans la base de données. On veut donc regrouper les iris dans différents groupes sans connaître ces derniers. On suppose donc que la base de données ne contient cette fois-ci que les longueurs et largeurs d'un pétale :

Longueur d'un pétale (cm)	Largeur d'un pétale (cm)
1.4	0.2
4.4	1.2
4.6	1.4
5.1	1.8
...	...

On donne ci-dessous le code complet permettant de localiser les centroïdes (barycentres). Il y est notamment question du calcul du tableau tableau\_dist qui contient les informations suivantes :

	Distance avec la centroïde $C_0$	Distance avec la centroïde $C_1$	...	Distance avec la centroïde $C_j$	...	Indice de la centroïde la plus proche
Entrée $X_0$	flottant	flottant	...	flottant	flottant	0
Entrée $X_1$	flottant	flottant	...	flottant	flottant	0
...	...	...	...	...	...	2
Entrée $X_i$	flottant	flottant	...	flottant	flottant	1

```

1 import matplotlib.pyplot as plt
2 import numpy as np
3 from sklearn import datasets
4 iris = datasets.load_iris()
5
6 donnees = iris.data
7 Xtot = donnees[:,2:4]
8
9 ##### Début - Ne pas tenir compte
10 ##### Affichage
11 plt.figure()
12 coul = ['blue','red','orange','green']
13 mark = ['*','o','^','x']
14 ##### Fin - Ne pas tenir compte
15
16 ##### Préparation des données
17 from sklearn.model_selection import train_test_split
18 X_train, X_test = train_test_split(Xtot,shuffle=True, test_size=0.33)
19
20
21 ##### Définition de la fonction qui calcule la distance entre deux données
22 def d(X,Xdata):
23     return ((X[0]-Xdata[0])**2 + (X[1]-Xdata[1])**2)**0.5
24
25
26 ##### Définition des k-barycentres (centroïdes) de manière aléatoires
27 import random as rd
28
29 maxi_0 = max(X_train[:,0])
30 mini_0 = min(X_train[:,0])
31 maxi_1 = max(X_train[:,1])
32 mini_1 = min(X_train[:,1])
33
34 ### Choix de k
35 k = 2
36
37 ### Ci contient les centroïdes initiaux
38 Ci = []
39 for i in range(0,k):
40     ci = [rd.random()*(mini_0 + i*(maxi_0-mini_0)/(k-1)),rd.random()*(mini_1 + i*(maxi_1-mini_1)
41     )/(k-1)]
42     Ci.append(ci)
43     ### pour l'affichage
44     plt.scatter(ci[0],ci[1],s=150,color=coul[i%4])
45     ### fin affichage
46
47 ### Nombre d'actualisation des centroïdes souhaitée
48 Ninc = 4
49 for inc in range(0,Ninc):
50
51     ##### Création du tableau des distances et association avec la plus proche centroïde
52     tableau_dist = []
53     ndata = len(X_train)
54
55     for i in range(0,ndata):
56
57         jmin = 0

```

```

57     ligne = []
58
59     for j in range(0,k):
60         c = d(X_train[i],Ci[j])
61         ligne.append(c)
62
63         if c<ligne[jmin]:
64             jmin = j ### jmin est l'indice associé à la centroïde la plus proche
65
66     ligne.append(jmin)
67     tableau_dist.append(ligne) ### tableau_dist est créé !
68
69 ##### Calcul des nouvelles coordonnées des centroïdes
70 N = [0]*k
71 Ci = [[0,0]]*k
72 for i in range(0,ndata):
73     kplus_proche = tableau_dist[i][k]
74     Ci[kplus_proche] = [ Ci[kplus_proche][0] + X_train[i,0], Ci[kplus_proche][1] + X_train[
75 i,1] ]
76     N[kplus_proche] = N[kplus_proche] + 1
77
78 for j in range(0,k):
79     Ci[j][0] = Ci[j][0] / N[j] ## peut planter si N[j] = 0 !
80     Ci[j][1] = Ci[j][1] / N[j] ## dans ce cas, relancer le code
81
82 ##### Début - Ne pas tenir compte
83 ##### Affichage
84 plt.title('incrément num '+str(inc))
85 ## Tracé des Datas
86 for i in range(0,ndata):
87     for j in range(0,k):
88         plt.scatter(X_train[i,0],X_train[i,1],s=3,color=coul[tableau_dist[i][k]%4])
89 plt.figure()
90 ## Tracé des Centroïdes
91 for j in range(0,k):
92     plt.scatter(Ci[j][0],Ci[j][1],s=150,color=coul[j%4])
93 plt.title('incrément num '+str(Ninc))
94 ## Tracé final des Datas
95 for i in range(0,ndata):
96     for j in range(0,k):
97         plt.scatter(X_train[i,0],X_train[i,1],s=3,color=coul[tableau_dist[i][k]%4])
98 ##### Fin - Ne pas tenir compte

```

**Question 13.** Exécuter ce code (il est sur votre cahier de prépa !).

**Question 14.** Analyser et annoter le code pour comprendre le fonctionnement de ce dernier.

**Question 15.** Écrire les instructions permettant de créer la variable Y\_pred contenant l'ensemble des prédictions pour les entrées de la base de test.

**Question 16.** Calculer l'erreur quadratique moyenne entre les prédictions et les données de la base de test.