TP

Apprentissage automatique

Algorithme des k-plus proches voisins

Algorithme des k-moyennes

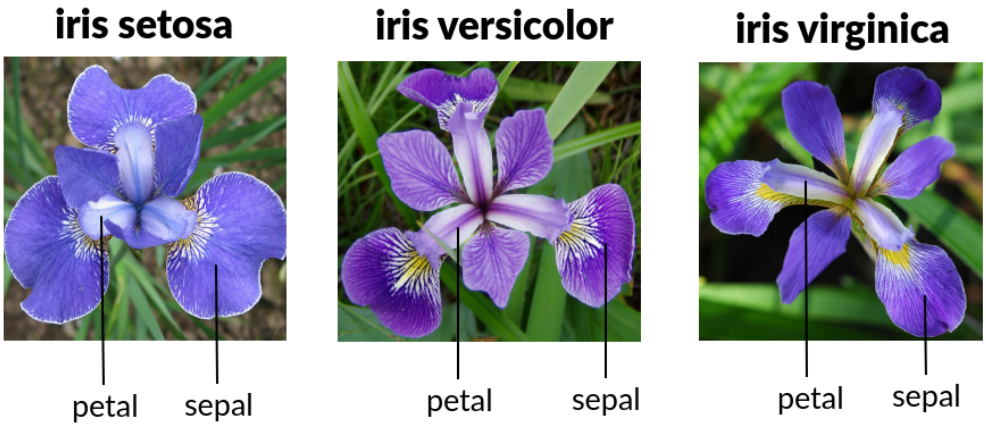
PSI : Lycée Rabelais

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	*setosa*
4.4	1.2	*versicolor*
4.6	1.4	*versicolor*
5.1	1.8	*virginica*
...	...	...



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 ?

Il s'agit d'un algorithme de **classification supervisé**.

On notera X, une donnée d'entrée ayant ici deux caractéristiques (longueur et largeur mesurées sur un pétale). On cherche à prédire la sortie $Y^{\text{préd}}$ qui correspond à l'espèce d'iris. On notera 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^{data} de la base de données est la plus proche de X ;

2 - Affecter à $Y^{\text{préd}}$ 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 $Y^{\text{préd}} = 1$.

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 :

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

Ytot = iris.target
donnees = iris.data
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 :

```
## Préparation des données
from sklearn.model_selection import train_test_split
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 et vérifier qu'il y a deux fois plus de données d'apprentissage que de données de test.

On a bien `len(X_train)` = 100 et `len(X_test)` = 50.

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

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

Voir question bonus.

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`.

```
def affiche(X,Y):
    for i in range(0,len(Y_train)):
        if Y_train[i]==0:
            plt.scatter(X_train[i,0],X_train[i,1],marker='*',color='blue')
        elif Y_train[i]==1:
            plt.scatter(X_train[i,0],X_train[i,1],marker='o',color='red')
        else:
            plt.scatter(X_train[i,0],X_train[i,1],marker='^',color='green')

affiche(X_train,Y_train)
```

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. Écire la fonction `d(X,Xdata)` qui permet de calculer la distance euclidienne entre les deux entrées `X` et `Xdata` (tableaux `numpy` de dimension 2).

```
def d(X,Xdata):
    return ((X[0]-Xdata[0])**2 + (X[1]-Xdata[1])**2)**0.5
```

On va maintenant créer une fonction qui renvoie une liste de listes de taille `(len(X_train),2)` contenant :

- la distance entre une entrée `X` et les données d'apprentissage.
- la sortie associée à la donnée d'apprentissage 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 :

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

Question 6. Compléter les lignes ci-dessous pour définir la fonction qui permet de créer le tableau trié préalablement mentionné.

```
from operator import itemgetter
def creationTableau(X,X_train,Y_train):
    T = []
    for i in range(0,len(X_train)):
        ligne = [d(X,X_train[i]),Y_train[i]]
        T.append(ligne)
    return sorted(T,key=itemgetter(0))
```

1.3 Identification

On cherche maintenant à écrire une fonction `identification(X,k,X_train,Y_train)` qui prédit, pour une entrée X, 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`, `tabOccurence` est égal à `[0,3,1]`, on en conclura que que l'iris est de type 1 (*versicolor*).

Question 7. Compléter la fonction `identification` ci-dessous.

```
def identification(X,k,X_train,Y_train):
    T = creationTableau(X,X_train,Y_train)
    ## calcul des occurences
    n = 3 ## ici il y a 3 types d'iris
    tabOccurence = [0]*n
    for i in range(0,k):
        tabOccurence[T[i][1]] += 1
    ## recherche de l'indice ayant l'occurance maxi
    jm = 0 ## jm est l'indice du max
    for j in range(1,n):
        if tabOccurence[j]>tabOccurence[jm]:
            jm = j
    return jm


X = [4.0,1.0] ## pour faire un test
print(identification(X,3,X_train,Y_train))
```

Question 8. Écrire les instructions permettant de créer la variable `Y_pred` contenant l'ensemble des prédictions pour les entrées de la base de test.

```
for X in  X_test:
    Y_pred.append(identification(X,2,X_train,Y_train))
```

Question 9. Écrire les instructions afin d'obtenir la matrice de confusion sous la forme d'une liste de liste.

```
cm = [[0]*3,[0]*3,[0]*3]

for i in range(0,len(Y_pred)):
    cm[Y_test[i]][Y_pred[i]] += 1
```

Question 10. Écrire les instructions afin d'obtenir la justesse.

```
justesse = (cm[0][0]+cm[1][1]+cm[2][2])/len(Y_pred)
```

Question 11. Commenter la qualité de votre algorithme en analysant notamment l'influence de la valeur de k.

La valeur de k importe peu et la justesse reste bonne.

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 connaitre 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

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn import datasets
iris = datasets.load_iris()

donnees = iris.data
Xtot = donnees[:,2:4]

##### Début - Ne pas tenir compte
######### Affichage
plt.figure()
coul = ['blue','red','orange','green']
mark = ['*','o','^','x']
##### Fin - Ne pas tenir compte

######### Préparation des données
from sklearn.model_selection import train_test_split
X_train, X_test = train_test_split(Xtot,shuffle=True, test_size=0.33)


######### Définition de la fonction qui calcule la distance entre deux données
def d(X,Xdata):
    return ((X[0]-Xdata[0])**2 + (X[1]-Xdata[1])**2)**0.5


######### Définition des k-barycentres (centroides) de manière aléatoires
import random as rd

maxi_0 = max(X_train[:,0])
mini_0 = min(X_train[:,0])
maxi_1 = max(X_train[:,1])
mini_1 = min(X_train[:,1])

### Choix de k
k = 3

### Ci contient les centroides initiaux
Ci = []
for i in range(0,k):
    ci = [rd.random()*(mini_0 + i*(maxi_0-mini_0)/(k-1)),rd.random()*(mini_1 + i*(maxi_1-mini_1
    )/(k-1))]
    Ci.append(ci)
    ### pour l'affichage
    plt.scatter(ci[0],ci[1],s=150,color=coul[i%4])
    ### fin affichage

### Nombre d'actualisation des centroides souhaitée
```

```
Ninc = 4
for inc in range(0,Ninc):

    ###### Création du tableau des distances et association avec la plus proche centroide
    tableau_dist = []
    ndata = len(X_train)

    for i in range(0,ndata):

        jmin = 0
        ligne = []

        for j in range(0,k):
            c = d(X_train[i],Ci[j])
            ligne.append(c)

            if c<ligne[jmin]:
                jmin = j ### jmin est l'indice associé à la centroide la plus proche

        ligne.append(jmin)
        tableau_dist.append(ligne) ### tableau_dist est créé !

    ###### Calcul des nouvelles coordonnées des centroides
    N = [0]*k
    Ci = [[0,0]]*k
    for i in range(0,ndata):
        kplus_proche = tableau_dist[i][k]
        Ci[kplus_proche] = [ Ci[kplus_proche][0] + X_train[i,0], Ci[kplus_proche][1] + X_train[
    i,1] ]
        N[kplus_proche] = N[kplus_proche] + 1

    for j in range(0,k):
        Ci[j][0] = Ci[j][0] / N[j]  ## peut planter si N[j] = 0 !
        Ci[j][1] = Ci[j][1] / N[j]  ## dans ce cas, relancer le code

    ##### Début - Ne pas tenir compte
    ######### Affichage
    plt.title('incrément num'+str(inc))
    ## Tracé des Datas
    for i in range(0,ndata):
        plt.scatter(X_train[i,0],X_train[i,1],s=3,color=coul[tableau_dist[i][k]%4])
    plt.figure()
    ## Tracé des Centroides
    for j in range(0,k):
        plt.scatter(Ci[j][0],Ci[j][1],s=150,color=coul[j%4])
plt.title('incrément num'+str(Ninc))
## Tracé final des Datas
for i in range(0,ndata):
    plt.scatter(X_train[i,0],X_train[i,1],s=3,color=coul[tableau_dist[i][k]%4])
    ##### Fin - Ne pas tenir compte
```

Question 12. Exécuter ce code (**il est sur votre cahier de prépa !**).

Question 13. Analyser et annoter le code pour comprendre le fonctionnement de ce dernier.

Question 14. Écrire les instructions permettant de créer la variable `Y_pred` contenant l'ensemble des prédictions pour les entrées de la base de test.

Voir ci-dessous.

Question 15. Calculer l'erreur quadratique moyenne entre les prédictions et les données de la base de test.

```
Y_pred = []
cout = 0
for xt in X_test:
    kpred = 0 ## kpred est la prédiction envisagée pour X_test
    dk = d(xt,Ci[0]) ## dk est la plus petite distance test-centroide
    for j in range(1,k): ## cherche la distance mini et actualise dk et kpred
        if d(xt,Ci[j]) < dk:
            kpred = j
            dk = d(xt,Ci[j])
    Y_pred.append(kpred) ## sauvegarde de la prédiction
    cout += dk ## calcul du cout

print('cout = ',cout)
```