Informatique Tronc Commun
Concours Blanc
Intelligence artificielle - Application en médecine
(d'après CCP PSI 2019)

L'objectif de ce sujet est d'appliquer l'algorithme KNN au domaine médical : à partir d'une base de données contenant des propriétés sur le bassin et le rachis lombaire, on cherche à déterminer si un patient présente un développement anormal de type hernie discale (saillie d'un disque intervertébral) ou spondylolosthésis (glissement du corps vertébral par rapport à la vertèbre sous-jacente).

1 Analyse des données

La base de données contient des informations administratives sur les patients et des informations médicales. Pour simplifier le problème, on considère deux tables : PATIENT et MEDICAL.

La table PATIENT contient les attributs suivants :
- `id` (entier, clé primaire) ;
- `nom` (chaîne de caractères) ;
- `prenom` (chaîne de caractères) ;
- `adresse` (chaîne de caractères) ;
- `email` (chaîne de caractères) ;

La table MEDICAL contient les attributs suivants :
- `id` (entier, clé primaire) ;
- `incidence_bassin` (flottant) ;
- `orientation_bassin` (flottant) ;
- `angle_lordose` (flottant) ;
- `pente_sacrum` (flottant) ;
- `rayon bassin` (flottant) ;
- `glissement_spon` (flottant) ;
- `id_patient` (entier, clé étrangère)
- `etat` (chaîne de caractères)

Les attributs `incidence_bassin`, ..., `glissement_spon` sont des données relatives à l'analyse médicale considérée. L'attribut `etat` peut prendre pour valeur `"normal"`, `"hernie discale"` ou `"spondylolisthésis"`.

1. Écrire une requête SQL permettant d'extraire les identifiants des patients ayant une hernie discale.
2. Écrire une requête SQL permettant d'extraire les noms et prénoms des patients atteints de spondylolisthésis.
3. Écrire une requête SQL affichant pour chaque état le nombre de patients concernés.
4. Écrire une requête SQL permettant d'extraire le nom et le prénom des patients dont l'attribut `glissement_spon` est supérieur à la moyenne.

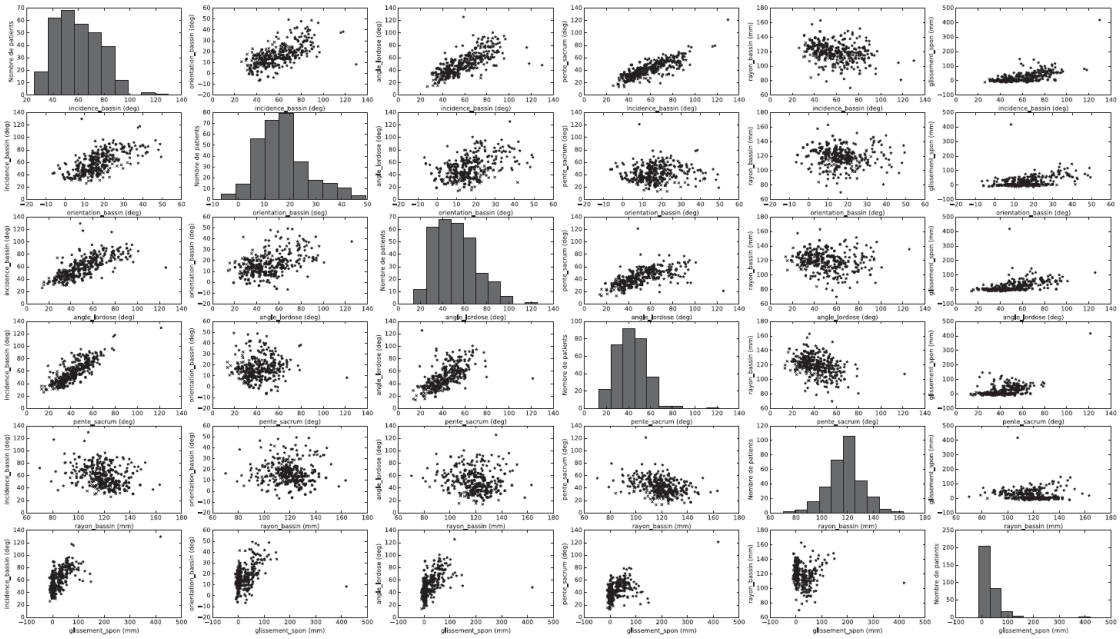
On suppose à présent que l'on a extrait de cette base les attributs médicaux de chaque patient, sous la forme de deux tableaux `numpy` : un tableau de flottants `data` à N lignes (nombre de patients) et 6 colonnes correspondant aux données médicales, et un vecteur de N entiers `etat`, contenant les états de chaque patient, selon le code 0 pour normal, 1 pour hernie discale et 2 pour spondylolisthésis.

5. En considérant que les flottants de ces tableaux `numpy` sont codés sur 32 bits, et les entiers sur 8 bits, déterminer la quantité de mémoire totale en Go nécessaire pour stocker ces données quand $N = 100000$.

On souhaite maintenant visualiser ces données de la façon suivante : en numérotant les attributs de 0 à 5, pour chaque paire (i, j) d'attributs,

- si $i = j$, on trace un histogramme représentant la répartition des valeurs de cet attribut ;
- si $i \neq j$, on trace pour chaque patient un point dont les coordonnées sont les attributs i et j, et dont la forme est déterminée par l'état du patient : o pour normal, x pour hernie discale et * pour spondylolisthésis.

On souhaite ainsi obtenir un tableau de figures tel que celui-ci :



Pour réaliser cette figure, il faut séparer les données en fonction de l'état du patient afin d'affecter un symbole par état. Cela revient à classer les patients en différents groupes.

6. Écrire une fonction `separation_par_groupe` qui prend en argument les tableaux `data` et `etat` et renvoie une liste de trois sous-tableaux obtenus à partir de data en le séparant selon les différentes valeurs de `etat`.
7. Écrire une ligne permettant d'importer la bibliothèque `matplotlib.pyplot` sous l'alias `plt`.

Pour afficher la figure, les instructions suivantes sont ensuite exécutées :

```
fig = plt.figure()
mark = ['o', 'x', '*']
label_attributs = ['incidence_bassin (deg)', 'orientation_bassin (deg)', \
'angle_lordose (deg)', 'pente_sacrum (deg)', 'rayon_bassin (mm)', 'glissement_spon (mm)']
groupes = separation_par_groupe(data, etat)
for i in range(len(groupes)):
    groupes[i] = array(groupes[i])
n = len(data[0])
for i in range(n):
    for j in range(n):
        ax1 = plt.subplot(ARGS1)
        plt.xlabel(label_attributs[i])
        if TEST:
            for k in range(len(groupes)):
                plt.ylabel(label_attributs[j])
                ax1.scatter(ARGS2)
        else:
            plt.ylabel("Nombre de patients")
            ax1.hist(ARGS3)
plt.show()
```

8. Remplacer `ARGS1, TEST, ARGS2, ARGS3` par des expressions permettant d'obtenir la figure désirée. On pourra trouver en annexe des indications sur les fonctions utilisées.

2 Apprentissage et prédiction

On rappelle le principe de la méthode KNN : pour prédire l'état d'un nouveau patient, on calcule la distance euclidienne entre ses données et celles de toutes les données d'entrainement, on extrait les K données les plus proches, et on renvoie l'état majoritaire au sein de ce groupe.

9. KNN relève-t-il de l'apprentissage supervisé ou non supervisé ? Justifier.

Avant de calculer la distance euclidienne, il est préférable de normaliser les attributs pour éviter que l'un ait plus de poids que les autres. On choisit de ramener toutes les valeurs des attributs entre 0 et 1. Une technique de normalisation consiste à appliquer à chaque valeur une fonction affine envoyant le minimum sur 0 et le maximum sur 1.

10. Écrire une fonction `min_et_max` prenant en argument un vecteur `X` et renvoyant son minimum et son maximum, en temps linéaire.
11. Écrire une procédure `normaliser` prenant en argument un vecteur et normalisant ses valeurs.
12. Écrire une fonction `distances` prenant en argument un vecteur Z et le tableau `data` (qu'on suppose déjà normalisé), et qui renvoie la liste de longueur N des distances entre Z et chaque ligne de `data`.

Pour déterminer les K plus proches voisins, il suffit d'utiliser un algorithme de tri efficace. La liste `T` à trier est une liste de couples (distance, état) que l'on veut trier selon les distances. On utilise pour cela la fonction suivante :

```
def tri(T):
    if len(T) <= 1:
        return T
    else:
        m =len(T)//2
        tmp1 = []
        for i in range(m):
            tmp1.append(T[i])
        tmp2 = []
        for i in range(m, len(T)):
            tmp2.append(T[i])
        return f(tri(tmp1), tri(tmp2))
```

13. Écrire la fonction `f` nécessaire au fonctionnement de ce tri. Cette fonction devra être de complexité linéaire.
14. Donner le nom de ce tri ainsi que sa complexité.
15. Ce tri est-il en place ? Cela peut-il être un problème dans le contexte considéré ?

On peut à présent terminer l'implémentation de la méthode KNN

16. Écrire une fonction `KNN` prenant en argument les tableaux `data` (normalisé), `etat`, un tableau `Z` correspondant aux attributs du nouveau patient, l'entier `K`, et le nombre d'états différents `nb`, et renvoyant l'état prédit par la méthode pour le nouveau patient.

Pour tester l'algorithme, on utilise un jeu de données normalisées de 100 patients supplémentaires dont l'état est connu. On note `data_test` ces données, et `etat_test` le vecteur d'états correspondant. Pour une valeur de `K` fixée, on peut alors appliquer l'algorithme sur chaque élément de ce jeu de données pour obtenir la matrice de confusion résultante.

17. Rappeler à quoi correspond la valeur de la case de coordonnées i, j de la matrice de confusion.
18. Écrire une fonction `test_KNN` prenant en argument `data_test`, `etat_test`, `data`, `etat`, `K` et `nb` le nombre d'états différents, et renvoyant la matrice de confusion pour ce test.

19. À partir de la matrice de confusion, il est facile de déterminer la proportion de prédictions correctes sur l'échantillon de test pour une valeur de K donnée. Pour simplifier, on fait l'hypothèse que cette proportion est une fonction croissante puis strictement décroissante de K. Écrire une fonction `K_optimal` prenant en argument `data_test`, `etat_test`, `data`, `etat` et `nb`, et renvoyant la valeur de K donnant la proportion maximale de prédictions correctes. La fonction devra faire un nombre d'appel à `test_KNN` logarithmique en ce K optimal (et ne pas faire d'appels à `KNN` par ailleurs).

Annexe - Éléments de syntaxe Python

- `M = array(([1,2,3], [3,4,5]))` donne un tableau `numpy` à deux lignes et 3 colonnes
- `M[1,2]` donne 5
- `M[:,i]` extrait la colonne d'indice i
- `M[i,:]` extrait la ligne d'indice i
- `M[:,0:2]` extrait les deux premières colonnes de `M` (ici, `array([[1, 2], [3, 4]])`)
- `zeros((2,3))` renvoie un tableau de 0 de 2 lignes et 3 colonnes
- `M.shape` donne les dimensions de `M` (ici, $(2,3)$)
- `trace(M)` renvoie la trace de la matrice `M`
- `plt.plot(x,y)` trace une courbe passant par les points dont les abscisses sont dans `x` et les ordonnées dans `y`
- `plt.xlabel(titre)` ajoute un titre à l'axe des abscisses
- `plt.ylabel(titre)` ajoute un titre à l'axe des ordonnées
- `ax1 = plt.subplot(a, b, k)` permet de sélectionner dans la variable `ax` la fla $k-$ième figure parmi un tableau de figures de dimensions a (nombre de lignes) $\times$ b (nombre de colonnes). Les figures sont numérotées en allant de gauche à droite et de haut en bas.
- `ax1.scatter(datax, datay, marker = mark[k])` premet de tracer sur la sous-figure `ax1` un nuage de points d'abscisses `datax` et d'ordonnées `datay`, en traçant chaque point avec le symbole `mark[k]`
- `ax1.hist(datax` permet de tracer l'histogramme des données de `datax` dans la sous-figure `ax1`