FRENOT Leon

MPSI 1

le 18/06

DS 09

II- Analyse des données

1) SELECT id from medical where etat = "hernie discale";

2) Select nom, prenom from patient join medical on patient.id = idpatient where etat = "spondylolisthésis";

3) Select etat, count(*) from medical group by etat;

4) Numpy possède des outils permettant de facilement manipuler des tableaux de grande taille.

5) On a 8 bits = 1 octet

donc chaque valeur du vecteur est de 1 octet

et chaque valeur du tableau est de 4 octets

On a donc une taille totale de $4 \times 6N + N = 25N = 2,5\ M_o$

6) def separationParGroupe(data, etat):

```
    zero = []
    un = []
    deux = []
    for i in range(len(etat)):
        temp = data[i,:]
        if etat[i] == 0:
            zero.append(temp)
        else:
            if etat[i] == 1:
                un.append(temp)
            else:
                deux.append(temp)
    return([zero, un, deux])
```

7) ARGS1 : n, n, i+1+n*j

ARGS2 : groupes[k][:,i], groupes[k][:,j], marker = mark[k]

ARGS3 : data[:,i]

TEST : i != j

8) Les diagonnes de la diagonale permettent de voir la fréquence des valeurs des attributs

Les diagonnes hors-diagonale permettent de voir le lien entre les différents attributs en général et en fonction des états.

IV. Apprentissage et prédiction

1. Méthode KNN

9) $x_{norm\,j} = \dfrac{x_j - \min(x)}{\max(x) - \min(x)}$

10)
```
def min_max(X):
    min = X[0]
    max = X[0]
    for k in X:
        if min > k:
            min = k
        if k > max:
            max = k
    return (min, max)
```

11)
```
def distance(z, data):
    N = []
    n = len(z)
    t_min = []
    t_max = []
    for k in range(n):
        min, max = min_max(data[:, k])
        t_min.append(min)
        t_max.append(max)
    for x in data:
        S = 0
        for k in range(n):
            S += ((z[k] - X[k])/(t_max[k] - t_min[k]))**2
```

S = sort(s)

N.append(s)

return(N)

12) - La partie 1 de la liste T et la trie

- La partie 2 compte parmis les K plus proches voisins les différents états

- La partie 3 donne l'état le plus présent parmis les K plus proches voisins.

- T est la liste décrite au début de la question
- dist est la liste contenant la distance entre chaque n-uplet x et le n-uplet z
- select est le compte des états parmis les K plus proches voisins
- ind est l'état le plus présent parmis les K plus proches voisins.

13) La diagonale correspond au nombre de fois où l'algorithme a correctement prédit l'état (la case i,i correspondant à l'état i)

La première ligne indique que lorsque le patient était sain, l'algorithme a prédit 23 fois qu'il était sain, 4 qu'il avait une hernie discale et 7 fois qu'il avait une spondylolisthesis.

La première colonne indique que lorsque l'algorithme avait prédit que le patient était sain, 23 fois le patient était sain, 7 fois il avait une hernie discale et 5 fois il avait une spondylolisthesis.

Cette matrice nous permet de comparer l'état donné par l'algorithme et l'état réel pour un K donné.

14) On observe sur la courbe que le K optimal est autour de 20. Ce qui nous donnerais une efficacité de environ 74 %.

2. Méthode de classification naïve bayésienne

15)
```
def moyenne(x):
    S=0
    n=len(x)
    for k in x:
        S+=k
    return(S/n)

def variance(x):
    S=0
    n=len(x)
    m=moyenne(x)
    for k in x:
        S+=(x-m)**2
    return(S/n)
```

16)
```
def synthese(data, etat):
    data_0 = separation_par_groupe(data, etat)
    N=[]
    for X in data_0:
        N1=[]
        for Y in X:
            m=moyenne(Y)
            v=variance(Y)
            N1.append([m,v])
        N.append(N1)
    return(N)
```

17) def gaussienne(a, moy, v):

```
    return (exp(-((a-moy)**2)/2*v)/sqrt(2*pi*v))
```

18) def probabilité_Groupe(z, data, etat):

```
    synt = synthese(data, etat)
    tab = []
    for X in synt:
        P=1
        for k in range(len(X)):
            m, v = X[k]
            P*= gaussienne(z[k], m, v)
        tab.append(P)
    return(tab)
```

19) def prediction(z, data, etat):

```
    tab = probabilité_Groupe(z, data, etat)
    i=0
    max=tab[0]
    for k in range(1, len(tab)):
        if tab[k]>max:
            max=tab[k]
            i=k
    return(i)
```

20) On voit dans les probabilités bornées qu'elles sont distantes de puissances de 10, le logarithme permet donc de réduire la taille des objets étudiés sans perdre en précision.

2) Pourcentage de réussite : $\frac{\text{cas identifiés}}{\text{cas totaux}} \times 100$

% réussite KKN : $\frac{23+11+40}{23+4+7+2+11+1+5+2+40} \times 100 = 74\,\%$

% réussite gaussienne : $\frac{23+10+49}{23+9+8+9+10+7+10+7+49} \times 100 = 68{,}3\,\%$

Il semble que la méthode KKN soit plus pertinente ici.