Analyse des données

1) SELECT idpatient FROM Medical where etat = 'hernie discale'

2) Select nom, prenom From Patient where id = (Select idpatient from medical where etat = 'spondylolisthésis').idpatient

3) Select count(idpatient) from medical group by etat

4) Numpy à un mode de stockage plus compact que les listes python.

5) Données représenté avec 8bits

Donc tableau : $N \cdot n \cdot 8$ bit $\Rightarrow \frac{N \cdot n}{1000000}$ Mo

vecteur : $N \cdot 1 \cdot 8$ bit $\Rightarrow \frac{N}{1000000}$ Mo

Donc total : $\boxed{N \cdot \frac{(n+1)}{1000000} \text{ Mo}}$

avec $N = 100000$ et $n = 6$

nous avons $\boxed{.7 \text{ Mo}}$

6)
```
def separationParGroupe(data, etat):
    sep = [[],[],[]]
    for i in range(data.shape[0]):
        sep[etat[i]].append(data[i])
    return sep
```

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

Args 2 : groupes[k, i, i], groupes[k, j], mark[k]

Args 3 : [d[i] for d in data]

Test : i != j

8) Les diagrammes hors diagonale donne les relations entre les diagrammes sur la diagonale.

Apprentissage et prédiction

9) $x\_norm_j = (x_j - min(x)) / (max(x) - min(x))$

10)
```
def min_max(x):
    minv, maxv = float("inf"), -float("inf")
    for v in x:
        if v < minv:
            minv = v
        elif v > maxv:
            maxv = v
    return minv, maxv
```

11)
```
def distance(z, data):
    dist = []
    for d in data:
        v = 0
        for i in range(len(d)):
            v += (d[i] - z[i])
        dist.append(v)
    return dist
```

12) Partie 1 : Calcule les distance entre les données du patient et la data puis les classes par ordre croissant.

Partie 2 : Pour les K éléments les plus proches du patient compte combien chaque état est représenté

Partie 3 : Trouver quel état est le plus représenté et le renvoyer comme résultat

13) La diagonale de la matrice donne le nombre de patients correctement répertorié pour chaque état.

Les autres positions donne l'information sur combien de fois chaque état a été confondue avec un autre.

14) L'efficacité pic à K=10 mais ne peut dépasser les 75% de réussite.

15) def moyenne(x):
```
    t=0
    for v in x:
        t+= v
    return v/x.shape[0]
```

~~def variance(x):~~
~~t, uc= moyenne(x)~~
~~for v in x~~

def variance(x):
```
    t, uc= 0, moyenne(x)**2
    for v in x:
        t+= v-uc
    return t/x.shape[0]
```

16)
```
def synthese(data, etat)
    G = [[],[],[],[]]
    gr = separation_en_Groupes(data, etat)
    for i in range(len(gr)):
        for j in range(len(gr[i])):
            G[i].append(moyenne(gr[i][j]), variance(gr[i][j]))
    return G
```

17)
```
def gaussienne(a, moy, v):
    return exp(-(a-moy)**2)/(2*v...))/((2*pi*(v))**.5)
```

18) ~~def probabilite_Groupe(z, data, etat):~~
~~ps = [0,0,0,0] [0] * len~~
~~s = synthese(data, etat)~~
~~for i in range(len(s))~~
~~fo~~

```
def probabilite_Groupe(z, data, etat):
    s = synthese(data, etat)
    ps = [1]* len(s)
    for i in range(len(s)):
        for j in range(len(s[i])):
            ps[i] = ps[i]* gaussienne(z[i], s[i][j][0], s[i][j][1])
    return ps
```

19)
```
def prediction(z, data, etat):
    ps = probabilité(z, data, etat)
    ~~cc = ps[0]~~
    ~~for v in ps~~
        ~~if v~~
    res, v = 0, ps[0]
    for i in range(len(ps)):
        if v < ps[i]:
            res = i
            v = ps[i]
    return i
```

21) Pour mettre en evidence la quantité ayant la plus haute probabilité et celle ayant la plus basse.

22) Réussite KNN = 74%

Réussite bayésienne = 82%

La méthode KNN à un taux de réussite bon pendant que la méthode bayésienne à un taux de réussite acceptable.