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

1 Analyse des données

1. ```
   SELECT id_patient
   FROM MEDICAL
   WHERE etat = "hernie discale"
   ```

2. ```
   SELECT nom, prenom
   FROM MEDICAL
   JOIN PATIENT
   ON PATIENT.id = id_patient
   WHERE etat = "spondylolisthésis"
   ```

3. ```
   SELECT etat, COUNT(*)
   FROM MEDICAL
   GROUP BY etat
   ```

4. ```
   SELECT nom, prenom
   FROM MEDICAL
   JOIN PATIENT
   ON PATIENT.id = id_patient
   WHERE glissement_spon > (SELECT AVG(glissement_spon) FROM MEDICAL)
   ```

5. Sachant qu'un octet contient 8 bits, le stockage nécessite $100000(6 \times 4 + 1) = 2,5$ Go.

6. ```
   def separation_par_groupe(data, etat):
       L = [[], [], []]
       N, n = data.shape
       for i in range(N):
           L[etat[i]].append(data[i])
       return L
   ```

7. ```
   import matplotlib.pyplot as plt
   ```

8. ```
   n = len(data[0])
   for i in range(n):
       for j in range(n):
           ax1 = plt.subplot(n, n, i*n + j + 1)
           plt.xlabel(label_attributs[i])
           if i != j:
               for k in range(len(groupes)):
                   plt.ylabel(label_attributs[j])
                   ax1.scatter(groupes[k][:,i], groupes[k][:,j], marker = mark[k])
           else:
               plt.ylabel("Nombre de patients")
               ax1.hist(data[:,j])
   plt.show()
   ```
   ```

2 Apprentissage et prédiction

9. Les données utilisées par l'algorithme sont annotées : on connaît l'état des patients présents dans la base. Il s'agit donc d'apprentissage supervisé.

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

11.
```
def normaliser(X):
    min, max = min_max(X)
    for i in range(len(X)):
        X[i] = (X[i]-min)/(max - min)
```

12.
```
def distances(Z, data):
    N, n = data.shape
    D = zeros(N)
    for i in range(N):
        D[i] = sqrt(sum((data[i] - Z)**2))
    return D
```

13.
```
def f(L1, L2):
    n1 = len(L1)
    n2 = len(L2)
    i1 = i2 = 0
    L = []
    while i1 < n1 and i2 < n2:
        if L1[i1] < L2[i2]:
            L.append(L1[i1])
            i1 += 1
        else :
            L.append(L2[i2])
            i2 += 1
    return L + L1[i1:] + L2[i2:]
```

14. Il s'agit du tri fusion, de complexité quasi-linéaire (en $O(n \ln n)$).

15. Ce tri n'est pas en place, chaque appel récursif crée un nouveau tableau via la fonction `f`. Cela peut-être un problème car les tableaux considérés sont de grande taille dans ce contexte, donc une quantité importante de mémoire vive sera nécessaire.

16.
```
def KNN(data, etat, Z, K, nb):
    dist = distances(Z, data)
    T = tri([(dist[i],i) for i in range(len(dist))])
    select = [0]*nb
    for i in range(K):
        patient = T[i][1]
        e = etat[patient]
        select[e] += 1
    imax = 0
    for i in range(1,nb):
        if select[i] > select[imax]:
            imax = i
    return imax
```

17. Il s'agit du nombre de données de test correspondant à l'état i sur lesquelles l'état j a été prédit par l'algorithme.

18.
```
def test_KNN(data_test, etat_test, data, etat, K, nb):
    M = zeros((nb,nb))
    n = len(etat_test)
    for p in range(n):
        i = etat_test[p]
        j = KNN(data, etat, data_test[p], K, nb)
        M[i][j] += 1
    return M
```

19.
```
def K_optimal(data_test, etat_test, data, etat, nb):
    d = 2
    while trace(test_KNN(data_test, etat_test, data, etat, d, nb)) \
        <= trace(test_KNN(data_test, etat_test, data, etat, d+1, nb)):
        d = 2*d
    g = d//2
    while g < d-1:
        m = (g+d)//2
        if trace(test_KNN(data_test, etat_test, data, etat, m, nb)) \
            < trace(test_KNN(data_test, etat_test, data, etat, m + 1, nb)):
            g = m
        else:
            d = m
    if trace(test_KNN(data_test, etat_test, data, etat, g, nb)) \
        < trace(test_KNN(data_test, etat_test, data, etat, d, nb)): return d
    else: return g
```