


## Big brother is watching you !

Ce sujet de devoir est complètement fictif ! Il n'est en lien avec aucune arrière pensée politique.

Dans son roman *1984*, George Orwell imagine un état autoritaire ayant installé une police de la pensée. Un algorithme d'intelligence artificielle, appelé *Big Brother*, est alors mis en place afin de surveiller les citoyens.

**Remarque.** On suppose que la commande `from math import sqrt` a déjà été saisie en amont de toutes les instructions que l'on écrira. On pourra donc utiliser `sqrt(x)` pour calculer la racine carrée du flottant  $x$ .

 **Question préliminaire.**

Écrire une fonction `dist(x1, x2)` qui prend en argument deux listes `x1` et `x2` de même taille et la calcule la distance euclidienne entre les deux.

```
def dist(x1, x2):
    S = 0
    for i in range(0, len(x1)):
        S += (x1[i] - x2[i])**2
    return sqrt(S)
```


### 1 Note d'insoumission

On s'intéresse à la manière dont *Big Brother* établit les notes d'insoumission des citoyens. La police a sélectionné aléatoirement  $N$  citoyens du pays, et a attribué à chaque individu une note d'insoumission en se basant sur les délits commis (plus la note est élevée, plus la personne est susceptible de se rebeller contre l'autorité).

On suppose que les données criminelles de ces  $N$  individus sont stockées sous forme numérique dans un array appelé `delits` de taille  $N \times p$ . La  $i$ -ème ligne, `delits[i]`, représente les données criminelles du  $i$ -ème individu. On aura toujours, peu importe la valeur de  $i$  : `len(delits[i]) = p`. Si aucune observation n'a été réalisée pour le  $i$ -ème individu, alors `delits[i]` sera le tableau `array([0,0,0,0 ...])` (de longueur  $p$ ).

On dispose également d'une liste `notes` de longueur  $N$  où la  $i$ -ème case contient la note d'insoumission attribuée par la police au  $i$ -ème individu.

On cherche à programmer l'algorithme des  $k$  plus proches voisins pour déterminer les notes d'insoumission du reste de la population.

 **Question.**

De quel type d'apprentissage s'agit-il ?

- Algorithme de régression non-supervisé
- Algorithme de classification non-supervisé
- Algorithme de régression supervisé
- Algorithme de classification supervisé

### Question.

Compléter la fonction `distances_au_point(delits,x)` ci-dessous qui prend en argument la matrice contenant les données criminelles des  $N$  individus et une liste  $x$  de longueur  $p$  représentant les données criminelles d'un nouvel individu, et qui renvoie une liste `distances` de longueur  $N$  où la  $i$ -ème case contient la distance de  $x$  à `delits[i]`.

```
1 def distances_au_point(delits,x):
2     distances = []
3     N = delits.shape[0]
4     for i in range(N):
5         x_i = delits[i]. .....
6         distances.append(.dist(x_i,x)). .....
7     return distances
```

On a défini une fonction `trier_individus(distances)` qui prend en argument la liste des distances de  $x$  à `delits[i]` et qui renvoie une liste `indices_tries` contenant les indices de la liste rangés par distance croissante. Par exemple, si `distances = [5,9,1]`, alors `trier_individus(distances)` renverra `[2,0,1]` : la case d'indice 2 présente la distance la plus petite, puis c'est la case d'indice 0 et enfin la case d'indice 1.

### Question.

Compléter la fonction `moyenne_des_k_voisins(indices_tries,notes,k)` qui prend en arguments la liste contenant les indices des observations triées par distance croissante à  $x$ , la liste contenant les notes d'insoumission des individus et le nombre de voisins à considérer, et qui renvoie la moyenne des notes d'insoumission des  $k$  plus proches voisins de  $x$ .

```
1 def moyenne_des_k_voisins(indices_tries,notes,k):
2     indices_voisins = indices_tries[:k]. .....
3     note = 0
4     for i in indices_voisins :
5         note = note + notes[i]. .....
6     note = note / k ..
7     return note
```

### Question.

On a écrit une fonction `k_plus_proches_voisins(delits,notes,x,k)` et on souhaite la tester.

On a prend les données suivantes :

```
1 import numpy as np
2 delits = np.array([[1,1,1],[1,2,3],[2,2,2],[0,3,6]])
3 notes = [1,5,4,10]
4 x = [2,2,1]
5 k = 2
```

Donner le résultat numérique obtenu après l'appel aux fonctions suivantes :

`distances_au_point(delits,x) = [1.41, 2.23, 1, 5.47]`

$k=2$  plus petites distances et notes associées

`moyenne_des_k_voisins(indices_tries,notes,k) = (1+4)/2 = 2.5`

## 2 Groupes de surveillance

Afin de faciliter la tâche aux agences de renseignement, *Big Brother* cherche à répartir les citoyens en plusieurs groupes homogènes. Pour cela, il dispose de nombreuses informations personnelles sur chaque citoyen ; à partir de ces descripteurs, il classera l'individu dans un des groupes, ce qui déterminera le niveau de surveillance à lui appliquer.

On utilise ici l'algorithme des  $k$  moyennes.

### Question.

De quel type d'apprentissage s'agit-il ?

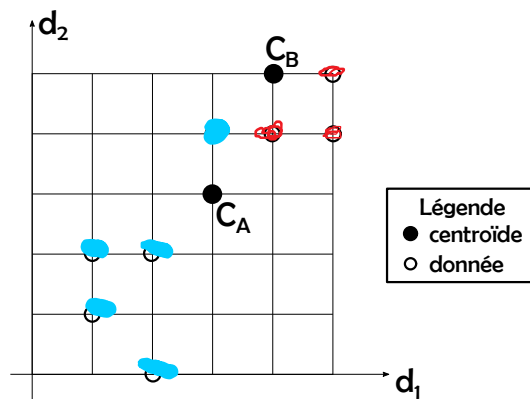
- Algorithme de régression non-supervisé
- Algorithme de classification non-supervisé
- Algorithme de régression supervisé
- Algorithme de classification supervisé

Pour simplifier et représenter les données, on suppose que l'on dispose de deux descripteurs, notés  $d_1$  et  $d_2$ . On se limite au regroupement dans deux ensembles  $\mathcal{A}$  et  $\mathcal{B}$ . On désigne par  $C_A$  et  $C_B$  les centroïdes (barycentres) respectives des deux ensembles  $\mathcal{A}$  et  $\mathcal{B}$ .

Sur les figures ci-dessous, on cherche à représenter l'évolution de la position des centroïdes. On représentera également l'appartenance de chaque entrée à sa centroïde associée. Dans la première phase, le positionnement de  $C_A$  et  $C_B$  est aléatoire.

### Question.

Colorier en bleu les données qui doivent être associées au groupe  $\mathcal{A}$  et en rouge celles qui doivent être associées au groupe  $\mathcal{B}$ .

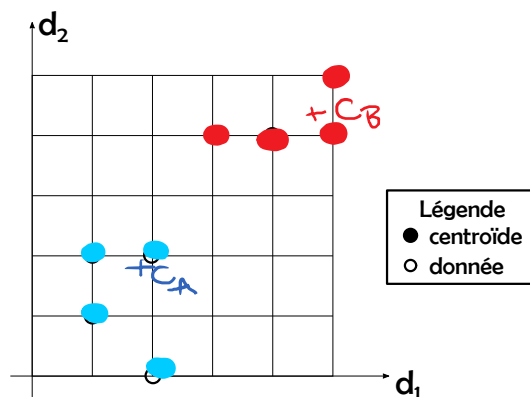


$$\text{Nouvelles coordonnées de } C_A = \left( \frac{(1+1+2+2+3)}{5} ; \frac{(0+1+2+2+4)}{5} \right) \\ = ( 1.8 ; 1.8 )$$

$$C_B = \left( \frac{(4+5+5)}{3} ; \frac{(4+4+5)}{3} \right) \\ = ( 4.7 ; 4.3 )$$

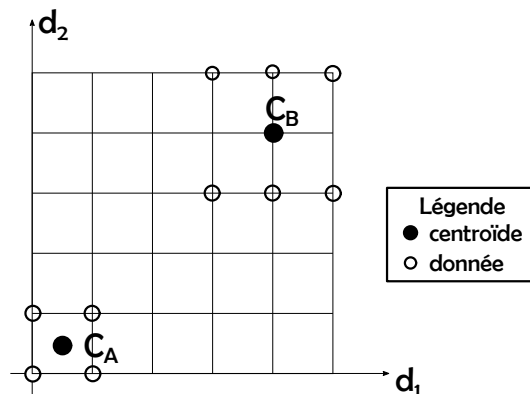
### Question.

Calculer les nouvelles coordonnées des centroïdes  $C_A$  et  $C_B$  puis tracer **précisément** leur nouvelle position.  
 Colorier alors en bleu les données qui doivent être associées au groupe  $\mathcal{A}$  et en rouge celles qui doivent être associées au groupe  $\mathcal{B}$ .



### Question.

Pour un autre jeu de données, on arrive à la situation représentée sur le graphique ci-dessous. On donne  $C_A(0,5;0,5)$  et  $C_B(4;4)$ , les coordonnées associées aux centroïdes des groupes  $\mathcal{A}$  et  $\mathcal{B}$ .  
 Un nouvel individu a pour couple de descripteurs  $X = (d_1, d_2) = (3,70;0,82)$ . Dans quel groupe faut-il associer cet individu ?



- Groupe  $\mathcal{A}$
- Groupe  $\mathcal{B}$

distance  $(X, C_A) = \sqrt{(0.5 - 3.7)^2 + (0.5 - 0.82)^2} \sim 3.22$   
 et de même distance  $(X, C_B) \sim 3.19 < 3.22$

## 3 Prédiction des délits

*Big Brother* peut également, grâce à un réseau de neurones, déterminer si l'individu a une grande chance de réaliser des délits dans sa vie ou non. Il classe donc cet individu dans le groupe "non-délinquant" ou dans un groupe "délinquant". Ce classement est basé sur des caractéristiques recueillies par les services de renseignements. Ces caractéristiques sont au nombre de 17. On peut y retrouver l'année de naissance de l'individu, le nom des parents, le lieu de naissance, la profession des parents, etc. Il y aura donc une couche d'entrée à 17 neurones.

La couche de sortie est composée de deux neurones. Le premier associé au groupe "non-délinquant" et le second au groupe "délinquant". Par exemple, si la sortie est le couple  $(1,0)$ , alors l'algorithme prévoit que l'individu sera un "non-délinquant".

Big Brother dispose d'une fiche de renseignement par citoyen. On considère qu'il existe  $n_d = 25$  millions de citoyens. Cette fiche de renseignement indique les caractéristiques d'entrée et la sortie associée (couple de deux valeurs indiquant si l'individu est un délinquant ou non).

### Question.

De quel type d'apprentissage s'agit-il ?

- Algorithme de régression non-supervisé
- Algorithme de classification non-supervisé
- Algorithme de régression supervisé
- Algorithme de classification supervisé

On envisage l'utilisation du code suivant :

```
1  ## Récupération des données
2  DATA = datasets.delinquance()
3  y = DATA.target
4  x = DATA.data
5
6  ## Définition du modèle
7  from sklearn.neural_network import MLPClassifier
8  mlp = MLPClassifier(hidden_layer_sizes=(10,6,4), activation='logistic')
9
10 ## Préparation des données
11
12 from sklearn.model_selection import train_test_split
13 x_train, x_test, y_train, y_test = train_test_split(x, y, shuffle=True, test_size=0.1)
14
15 mlp.fit(x_train, y_train)
16
17 y_pred = mlp.predict(x_test)
18
19 from sklearn.metrics import confusion_matrix
20 cm = confusion_matrix(y_test, y_pred)
21
22 print('matrice de confusion =', cm) ## Affichage de la matrice de confusion
```

On fournit les informations suivantes :

### Documentation.

`DATA = datasets.delinquance()` contient toutes les données (entrées et sorties). `y = DATA.target` permet d'accéder aux sorties (tableau de taille  $n_d \times 2$ ) et aux entrées `x = DATA.data` (tableau de taille  $n_d \times 17$ )

`MLPClassifier` est le modèle de réseau de neurones utilisé. `hidden_layer_sizes = (10,6,4)` indique qu'il y a 3 couches cachées de 10 puis 6 puis 4 neurones. `activation='logistic'` signifie qu'une fonction sigmoïde est utilisée comme fonction d'activation.

`train_test_split` permet de séparer les données en données d'apprentissage (entrée `x_train` ; sortie `y_train`) et données de test (entrée `x_test` ; sortie `y_test`). `test_size=0.1` indique que 10% des données seront utilisées pour le test et 90% pour l'apprentissage.

`mlp.fit(x_train, y_train)` entraîne le modèle.

confusion\_matrix(y\_test, y\_pred) renvoie la matrice de confusion où la lecture en ligne donne les vraies valeurs et en colonne les valeurs prédites par le modèle.

### Question.

Combien de paramètres (poids et biais) composent le modèle ? Donner la réponse sans justification.

$N_{\text{paramètres}} = 10 \times (17+1) + 6 \times (10+1) + 4 \times (6+1) + 2 \times (4+1)$   
Il y a donc 284 paramètres.

### Question.

Combien de données seront utilisées pour l'apprentissage ? L'algorithme va-t-il *surapprendre* ? Justifier.

Il y a ici  $N_{\text{données, apprentissage}} = 0.9 \times 25 = 22.5$  millions de jeu de données pour l'apprentissage.

On a  $N_{\text{données, apprentissage}} \gg N_{\text{paramètres}}$  donc il n'y a, a priori, pas de risque de surapprendre.

### Question.

La matrice de confusion est la suivante (les valeurs sont en millions d'individus) :

		Prédictions	
		Non-Délinquant	Délinquant
Vraies	Non-Délinquant	2.35	0.07
	Délinquant	0.02	0.06

Compléter les phrases ci-dessous avec la valeur numérique adaptée (en %).

- La justesse du modèle est de 96.4 %.
- Les délinquants sont bien prédits à hauteur de 75 %.
- Il y a 3.2 % de la population qui est délinquante.
- 54 % des délinquants prédits sont en réalité des non-délinquants.

FIN