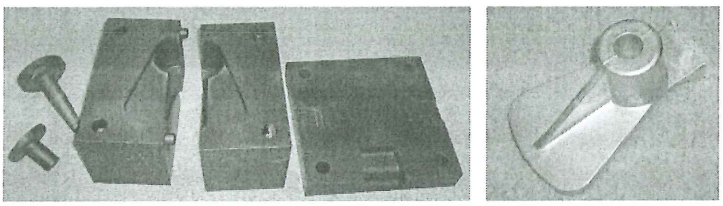
Détection d'erreurs dans une chaine de production de fonderie

On s'intéresse ici à une chaine de production de pièces de fonderie. On suppose qu'une telle chaine est essentiellement composée :

- de zones de circulation : ces zones permettent d'acheminer les pièces à fabriquer ou bien le liquide fondu entre les différents postes.
- de postes de fonderie à proprement parlé : ces postes permettent de couler le métal en fusion dans les moules afin d'en obtenir la pièce souhaitée.
- de postes de démoulage : ces postes permettent de sortir les pièces obtenues après solidification.



Pièce obtenue par moulage (à droite) et les moules utilisés (à gauche)

Dans ce type de chaine de production, une problématique récurrente est **la détection des pièces présentant des défauts** (le service responsable de cette mission au sein d'une entreprise est le "contrôle qualité"). Si un défaut est détecté, il faut alors mettre la pièce au rebut.

On souhaite, dans ce sujet, concevoir un algorithme d'intelligence artificielle pour :

- détecter un défaut dans une pièce.
- détecter une machine nécessitant une maintenance.

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`, toutes les deux de taille $n > 0$ où $n \in \mathbb{N}$, et calcule la distance euclidienne entre ces deux listes.

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

$$d = \sqrt{\sum_{i=0}^{n} (x1[i] - x2[i])^2}$$

1 Détecter un défaut dans une pièce.

L'intelligence artificielle doit établir une note dite de qualité pour chaque pièce : cette note est un flottant qui est obtenue en recroisant plusieurs informations liées à celle-ci (masse, images, température, etc.). La note de qualité sera égale à 100.0 pour une pièce parfaite et 0.0 pour une pièce ayant énormément de défauts.

Le service qualité a sélectionné aléatoirement N pièces fabriquées, et a attribué manuellement à chacune des pièces:

- une liste des défauts constatés ;
- la note de qualité choisie.

On suppose alors que les données associées à ces N pièces sont stockées sous forme numérique dans un `array` (de la bibliothèque `numpy`) appelé `defauts` de taille $N \times p$. La i-ème ligne, `defauts[i]`, représente les défauts constatés sur la i-ème pièce. On aura toujours, peu importe la valeur de i : `len(defauts[i]) = p`. Si aucune observation n'a été réalisée pour la i-ème pièce, alors `defauts[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 de qualité attribuée par le service qualité à la i-ème pièce.

On cherche à programmer l'**algorithme des k plus proches voisins** pour déterminer les notes de qualité de toutes les pièces sortants de l'usine.

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(defauts,x)` ci-dessous qui prend en argument la matrice contenant les défauts des N pièces et une liste `x` de longueur `p` représentant les défauts d'une nouvelle pièce, et qui renvoie une liste `distances` de longueur N où la i-ème case contient la distance de `x` à `defauts[i]`.

```
def distances_au_point(defauts,x):
    distances = []
    N = defauts.shape[0]
    for i in range(N):
        x_i = ..defauts[i].....
        distances.append(..dist(x, x_i))....
    return distances
```

On a défini une fonction `trier_pieces(distances)` qui prend en argument la liste `distances = distances_au_point(defauts,x)` 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_pieces(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 pièces triées par distance croissante à `x`, la liste contenant les notes de qualité des pièces déjà testées et le nombre de voisins à considérer, et qui renvoie la moyenne des notes de qualité des k plus proches voisins de `x`.

```
def moyenne_des_k_voisins(indices_tries,notes,k):
    indices_voisins = ..indices_tries[:k]
    note = 0
    for i in indices_voisins :
        note = note ..+ notes[i]
    note ..= note / k
    return note
```

Question.

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

On a prend les données suivantes :

```
1 import numpy as np
2 defauts = 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(defauts,x)` = [1.41, 2.23, 1, 5.47]

2 plus petites distances associées aux notes 1 et 4

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

2 Détection du besoin de maintenance sur une machine

Afin de faciliter la tâche du service qualité, un autre algorithme doit permettre de détecter une machine fonctionnant anormalement. Les machines de la chaine doivent donc être associées au groupe "*Fonctionnement normal*" ou au groupe "*Maintenance à prévoir*". Pour détecter un problème sur une machine l'algorithme dispose de nombreuses informations propres à chacune d'entre elles (appelés descripteurs) : tensions en divers points du circuit électrique, pression hydraulique, température de la machine, etc. On ne sait pas, a priori, la normalité des valeurs que doivent prendre les descripteurs.

On utilisera ici l'**algorithme des k moyennes** pour résoudre ce problème.

Question.

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

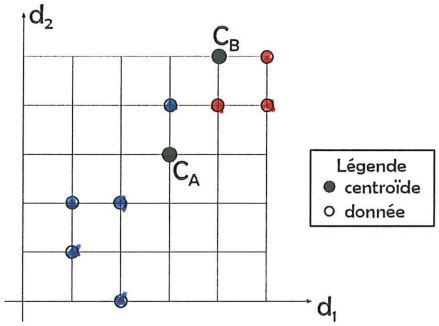
- ○ Algorithme de régression non-supervisé
- ● Algorithme de classification non-supervisé
- ○ Algorithme de régression supervisé
- ○ Algorithme de classification supervisé

Pour simplifier et représenter les données, on suppose que l'on dispose uniquement de deux descripteurs, notés d_1 et d_2. On cherche à regrouper les machines dans deux ensembles $\mathscr{A}$ ("*Fonctionnement normal*") ou $\mathscr{B}$ ("*Maintenance à prévoir*"). On désigne par C_A et C_B les centroïdes (barycentres) respectives des deux ensembles $\mathscr{A}$ et $\mathscr{B}$.

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 $\mathscr{A}$ et en rouge celles qui doivent être associées au groupe $\mathscr{B}$.



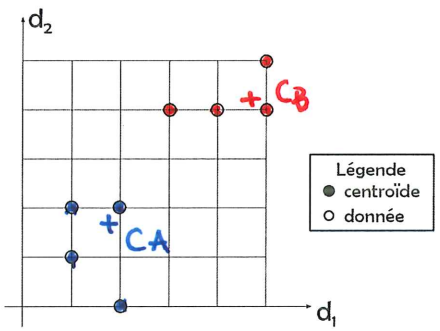
Question.

Calculer les nouvelles coordonnées des centroïdes C_A et C_B (faire apparaitre le calcul effectué !) puis tracer **précisément** leur nouvelle position.

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

$C_A = ((1+1+2+2+3)/5 ;$
$(0+1+2+2+4)/5)$
$= (1.8 ; 1.8)$

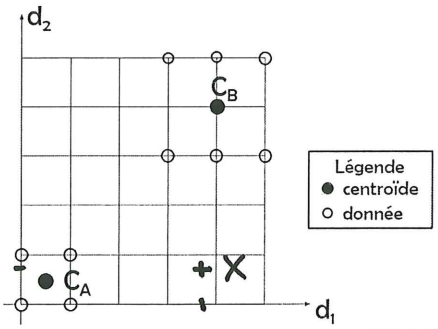
$C_B = ((4+5+5)/3 ;$
$(4+4+5)/3)$
$\simeq (4.7 ; 4.3)$



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 $\mathscr{A}$ et $\mathscr{B}$.

Une machine a pour couple de descripteurs $X = (d_1, d_2) = (3,70; 0,82)$. Dans quel groupe faut-il associer cette machine ? Faire apparaître votre calcul si besoin.



○ Groupe $\mathscr{A}$

● Groupe $\mathscr{B}$

Distance $XC_A = \sqrt{(0,5-3,7)^2 + (0,5-0,82)^2} \simeq 3,22$

" $XC_B \simeq 3,12$ $(< XC_A)$

3 Prévision des défauts sur une pièce

L'intelligence artificielle peut également, grâce à un réseau de neurones, déterminer si la pièce a une grande chance de présenter des défauts en sortie d'usine. Il classe donc ces pièces dans le groupe "pas de défaut" ou dans le groupe "avec défauts". Ce classement est basé sur des caractéristiques recueillies par 17 capteurs (température, masse, tension, etc.) équipant la chaine de production. 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 "pas de défaut" et le second au groupe "avec défauts". Par exemple, si la sortie est le couple $(1, 0)$, alors l'algorithme prévoit que la pièce ne présentera pas de défaut.

Pour entrainer le réseau, on dispose de fiches éditées par le service qualité avant la mise en place de l'algorithme. Sur chaque fiche, on retrouve l'association choisie pour chaque pièce (avec ou sans défaut) et la valeur des 17 capteurs lors de la fabrication. On considère qu'il existe $n_d = 25000$ fiches.

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 :

```
## Récupération des données
DATA = datasets.defauts()
y = DATA.target
x = DATA.data

## Définition du modèle
from sklearn.neural_network import MLPClassifier
mlp = MLPClassifier(hidden_layer_sizes=(10,6,4),activation='logistic')

## Préparation des données

from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y,shuffle=True, test_size=0.1)

mlp.fit(x_train,y_train)

y_pred = mlp.predict(x_test)

from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)

print('matrice de confusion =',cm) ## Affichage de la matrice de confusion
```

On fournit les informations suivantes :

Documentation.

`DATA = datasets.defauts()` 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.

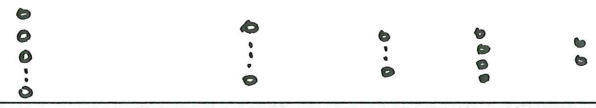
`mlp.fit(x_train,y_train)` entraîne le modèle.

`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.

17 neurones 10 6 4 2



Il y a :
- 10 + 6 + 4 + 2 = 22 biais
- et 262 poids

donc 284 paramètres au total.

Question.

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

- On utilise : 25000 × 90% = 22 500 données pour l'apprentissage.

- Ici le nombre de données d'apprentissage est largement supérieur au nombre de paramètre. Il n'y a donc pas de risque de surapprentissage.

Question.

La matrice de confusion est la suivante (les valeurs sont en milliers de pièces) :

		Prédictions	
		Pas de défaut	Avec défauts
Vraies	Pas de défaut	2.35	0.07
	Avec défauts	0.02	0.06

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

- La justesse du modèle est de ..96,4.. %.
- Les pièces avec défauts sont bien prédites à hauteur de ...75.. %.
- Il y a ..3,2.. % des pièces testées qui présentent réellement des défauts.
- ..54.. % des pièces prédites avec défauts sont en réalité sans défaut.

FIN