Informatique du Tronc Commun

Chapitre n°4 Intelligence artificielle

Prix Nobel de physique 2024

Le prix Nobel de physique 2024 a été attribué mardi 8 octobre à : John J. Hopfield (américain) et Geoffrey E. Hinton (canado-britanique), pour leurs contributions exceptionnelles dans le domaine de la science des réseaux neuronaux et de l'intelligence artificielle.
Ces deux scientifiques ont utilisé les outils de la physique pour développer des méthodes qui sont à la base du machine learning (apprentissage machine). John Hopfield a créé une mémoire associative qui peut stocker et reconstruire des images et d'autres types de motifs dans une base de données. Geoffrey Hinton a inventé une méthode qui peut, de manière autonome, analyser des données pour en sortir des propriétés, des éléments spécifiques, et ainsi réaliser différentes taches en identifiants ces éléments particuliers dans les images.


Ill. Niklas Elmehed © Nobel Prize Outreach
John J. Hopfield


Ill. Niklas Elmehed © Nobel Prize Outreach
Geoffrey E. Hinton

Lorsque l'on parle d'intelligence artificielle, on fait souvent référence au machine learning qui utilise un réseau de neurones. Cette technologie a été inspirée de la structure du cerveau. Dans un réseau de neurones artificiels, les neurones sont représentés par des "nœuds" qui peuvent prendre différentes valeurs. Ces nœuds s'influencent mutuellement via des connexions, semblables aux synapses. Le réseau est entraîné, et par ce biais développe, par exemple, des connexions plus fortes entre certains nœuds.

Introduction

L'expression *intelligence artificielle* a été créée par les américains John McCarthy et Marvin Lee Minsky en 1959. Il s'agit de faire exécuter par des machines des tâches habituellement exécutées par des humains, qui nécessitent des capacités de raisonnement, de mémoire et d'apprentissage. En particulier, on souhaite rendre une machine capable d'accéder à certaines connaissances sans avoir été programmée explicitement pour cela, autrement dit capable d'apprendre par elle-même.

L'expression *apprentissage automatique* (ou *machine learning* en anglais), a été créée par Arthur Samuel en 1959.

D'après Tony Michell (1988), on peut dire qu'une machine apprend si après certaines expériences elle a augmenté sa capacité à réaliser une certaine tâche.

Tous les jours vous utilisez des outils qui utilisent l'intelligence artificielle :

— l'étiquetage de photos, qui repose sur la détection d'objets dans des images. La recherche de photos se fait ensuite sur le résultat de cette détection. Pour essayer, tapez "pizza" dans Google images !
— la traduction automatique *(INTERDITE en cours, colles, devoir d'anglais !!)* ;
— la prédiction d'une grandeur, par exemple d'un temps de trajet sur une appli de navigation ;
— la reconnaissance et la synthèse vocale de l'application de navigation ;
— le système de recommandation de Netflix.

Objectifs

Les objectifs de ce chapitre sont de :

— découvrir ce qu'est l'intelligence artificielle ;
— expliquer et utiliser l'algorithme d'apprentissage supervisé des k plus proches voisins (KNN) ;
— expliquer et utiliser l'algorithme d'apprentissage non supervisé des k moyennes. (Kmeans)

Pré-requis

— Algorithme gloutons,
— Manipulation de listes.

Plan du cours

I Intelligence artificielle

I.1 Qu'est-ce que l'intelligence artificielle ?

Définition : Intelligence artificielle

Sous le terme d'intelligence artificielle se cachent un ensemble de techniques permettant à des machines d'accomplir des tâches et de résoudre des problèmes normalement réservés aux humains, comme par exemple reconnaître et localiser des objets dans une image.

Depuis quelques années, on associe presque toujours l'intelligence aux capacités d'apprentissage : c'est grâce à l'apprentissage qu'un système intelligent, capable d'exécuter une tâche, peut améliorer ses performances avec l'expérience.

Définition : Apprentissage automatique

L'apprentissage automatique est un domaine de l'intelligence artificielle. Il s'agit de permettre à une machine de progresser, d'apprendre par elle-même à améliorer son fonctionnement dans un cadre de résolution d'un problème, mais sans jamais la programmer explicitement pour résoudre un problème.

À partir de données, un modèle est construit. Par exemple, pour apprendre à reconnaître un élément dans une image, on fournit à la machine des images avec la présence ou non de l'élément, et un modèle est élaboré. Ce modèle permet à la machine, lorsqu'on lui fournit une image nouvelle de dire si l'élément est présent ou non.

On distingue deux types d'apprentissage : l'*apprentissage supervisé* et l'*apprentissage non supervisé*.

I.2 Apprentissage supervisé

Définition : Apprentissage supervisé

L'apprentissage supervisé consiste pour un opérateur à montrer à la machines des milliers voire des millions d'exemples étiquetés avec leur catégorie, qui permettront à la machine de déterminer elle-même les paramètres pertinents pour classer chaque objet dans la catégorie qui lui correspond. Une fois cette phase d'apprentissage terminée, la machine doit être capable de généraliser à des objets pas encore vus.

I.3 Apprentissage non supervisé

Définition : Apprentissage non supervisé

L'apprentissage non supervisé est plus ambitieux, mais il est aussi plus proche de notre propre modèle d'apprentissage, basé sur l'observation. Il consiste à faire en sorte qu'à partir d'un ensemble de données (non étiquetées) la machine soit capable de créer ses propres catégories, et si possible que ces catégories soient pertinentes pour nous.

Dans la suite nous allons nous intéresser à deux algorithmes :

— l'algorithme des k plus proches voisins utilisé en apprentissage supervisé ;

— l'algorithme des k-moyennes utilisé en apprentissage non supervisé.

II Apprentissage supervisé : algorithme des k plus proches voisins

Nous nous intéressons à un algorithme d'apprentissage supervisé : celui des k plus proches voisins (en anglais KNN pour k-nearest neighbors).

On possède un ensemble de données classées dans un certain nombre de catégories, et une donnée dont on souhaite déterminer la catégorie.

> **Exemple 1.** On dispose d'un ensemble d'iris caractérisés par la longueur et largeur de leurs sépales et de leurs pétales, associés à leur variété.
>
> On souhaite déterminer la variété d'un iris dont on connait la longueur et largeur de ses sépales et de ses pétales.

II.1 Algorithme des k plus proches voisins

Nous allons considérer un exemple plus simple, un ensemble de 200 points de coordonnées (x, y) dans le plan qui peuvent appartenir à deux catégories : bleu ou rouge. Nous fournissons à la machine le jeu de ces 200 données d'apprentissage.

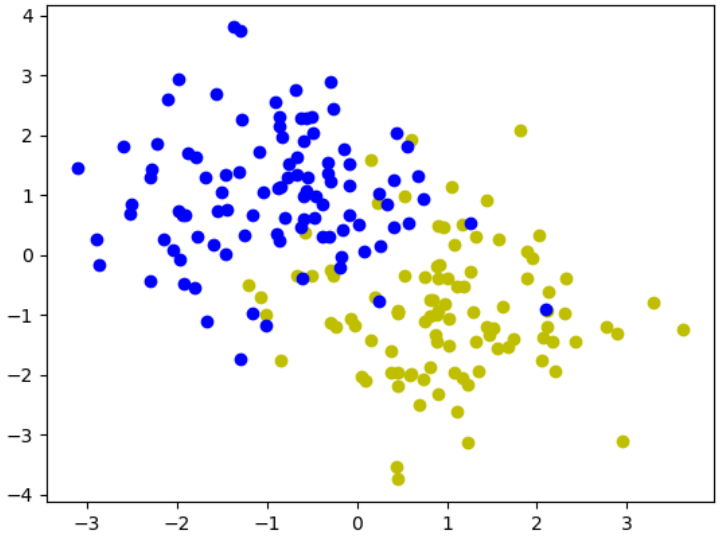

FIGURE 1 – Visualisation des données

Les données d'apprentissage sont constituées d'une liste de listes de format $[[p_0, c_0], [p_1, c_1], ...]$:

— p_i est les coordonnées du point i, données sous la forme d'une liste de deux éléments $p_i = [x_i, y_i]$;

— c_i est un entier qui indique la couleur du point : 0 pour bleu, et 1 pour rouge.
La couleur c_i est le rang de la liste `categories=["bleu","red"]` dont la valeur indique la couleur du point.

Une fois ces données acquises, nous souhaitons que la machine attribue à un point P donné de coordonnées (x, y) (et de catégorie inconnue) une catégorie rouge ou bleu.

L'algorithme des k plus proches voisins prend en entrée :

— la liste précédente des données d'apprentissage,

— un point `p` du plan, constitué d'une liste de ses deux coordonnées `[x,y]`,

— un entier `k` le nombre de voisins considérés.

La notion de « k **plus proches voisins** » est prise **en terme de distance entre le point `P` et les points des données d'apprentissage**.

Principe de l'algorithme des k plus proche voisin

La méthode des k plus proches voisins consiste à **déterminer la catégorie des k données d'entraînement les plus proches de la donnée inconnue, et à attribuer à cette inconnue la catégorie majoritaire parmi les k voisins les plus proches.**

Nous détaillons ici l'algorithme des k plus proches voisins pour déterminer la catégorie d'un point dont les coordonnées sont connues en utilisant un ensemble de données d'apprentissage.

Méthode : Algorithme des K plus proches voisins

(1) Calculer les distances séparant l'ensemble des points P_i des données avec le point P dont on cherche à déterminer la catégorie.
On obtiendra une liste de listes de deux valeurs `distance=[....,[di,ci],...]`. Pour chaque liste, le premier élément est la distance d_i séparant le point P_i des données et le point P, et le deuxième élément est la couleur du point P_i.

(2) Classer la liste des distances par ordre de distance croissante. On pourra utiliser la méthode `L.sort()` de `python` qui trie la liste `L` par ordre lexicographique du premier élément (c'est ce qu'on veut ici).

(3) Parcourir les k plus proches voisins de P, et compter le nombre de voisins dans chacune des catégories.

(4) Renvoyer la catégorie de P comme étant la majoritaire parmi ses k plus proches voisins.

II.2 Distance

Nous devons donc commencer par définir la distance. Nous utiliserons la distance euclidienne (nous pourrions décider d'utiliser d'autres distances).

Définition : Distance euclidienne

La distance entre les deux vecteurs de $\mathbb{R}^m$, $X = \begin{pmatrix} x_0 \\ \vdots \\ x_{m-1} \end{pmatrix}$ et $Y = \begin{pmatrix} y_0 \\ \vdots \\ y_{m-1} \end{pmatrix}$ est

$$d(X,Y) = \sqrt{\sum_{i=0}^{m-1} (x_i - y_i)^2}$$

```python
def distance(p1:list,p2:list)->float:
    """
    Entrée :
    p1, p2 :  listes de m valeurs, représentant les coordonnées de chaque point en dimension m
    Renvoi :
    d : distance euclidienne en dimension n entre les deux points
    """
    ........................................
    ........................................
    ........................................
    ........................................
    ........................................
    ........................................
```

II.3 Implémentation en python

1. Fonction qui renvoie la liste de listes de deux valeurs `distance=[....,[di,ci],...]`

```
def liste_distances(donnees,P):
    """
    Entrées :
    donnees : liste des données [[pi,ci]], avec pi la liste des coordonnées des données, et ci la catégorie de la donnée
    P : un point dont on souhaite déterminer la catégorie, liste de ses coordonnées
    Renvoi : la liste  de listes [[di,ci]] des distances de P avec pi
    """
    distances=[]
    for i in range(len(donnees)):


    return distances
```

2. Fonction qui classe la liste des distances par ordre de distances croissantes, et renvoie le nombre de voisins dans chacune des catégories parmi les k plus proches voisins de P.

```
def kvoisins(dist:list,categories:list,k:int)->list:
    """
    Entrées :
    dist : liste des distances [[di,ci]]
    categories : liste des catégories
    Renvoi : la liste du nombre de voisins, au rang i est le nombre de voisins de categories[i] présents parmi les k plus proches voisins du point à déterminer
    """
    dist.sort() # trie la liste dist par ordre croissant du premier élément
    L_nbre=[0]*len(categories) # au rang i est le nombre de voisins de categories[i] présents à proximité de P
    for i  ..................................
        ....................................
    return ..............
```

3. Fonction qui renvoie le rang du maximum d'une liste.

```
def rang_max(L:list)->int:
    """
    Renvoi le rang du maximum de la liste L
    """
    ....................................
    ....................................
    ....................................
    ....................................
    ....................................
```

4. Fonction qui renvoie la catégorie du point P.

```
def knn(donnees:list,P:list,categories:list,k:int)->int:
    """
    renvoi la catégorie du point P comme étant la catégorie majoritaire parmi ses k plus proches voisins
    """
    ....................................
    ....................................
    ....................................
    ....................................
```

II.4 Matrice de confusion

Pour tester quantitativement la qualité de l'apprentissage, nous utilisons une **matrice de confusion**.

Pour cela, nous séparons les données en deux groupes : des données d'apprentissage et des données qui permettent de tester la qualité de l'apprentissage.

Pour chaque donnée de test, nous appliquons l'algorithme des k-plus proches voisins en s'appuyant sur les données d'apprentissage, et nous comparons la couleur prédite avec la couleur réelle.

Selon les résultats, les points tests vont se répartir en quatre catégories :

— le point est rouge et identifié comme tel,

— le point est bleu et identifié comme tel,

— le point est rouge, mais a été reconnu (à tort) comme bleu,

— point est bleu, mais a été reconnu (à tort) comme rouge.

Ces résultats sont alors regroupés dans une matrice, appelée **matrice de confusion** :

		Couleur prédite	
		Bleu	Rouge
Couleur réelle	Bleu	vrai bleu	faux rouge
	Rouge	faux bleu	vrai rouge

Définition : Matrice de confusion

Une matrice de confusion est construite ainsi :

— chaque ligne correspond à la classe réelle,

— chaque colonne correspond à la classe prédite par l'algorithme.

À l'intersection d'une ligne i et d'une colonne j, on trouve le nombre d'éléments de classe réelle i et de classe prédite j.

On modifie légèrement la fonction `knn` pour qu'elle renvoie le numéro de la catégorie majoritaire (`imax`) et non le nom de la catégorie.

```
def matrice_confusion(apprentissage,test,categories,k):
    """
    Entrée :
    apprentissage : liste des données d'apprentissage
    test : liste de liste des données utilisées pour le test
    k : nombre de voisins choisis pour identifier la catégorie
    Renvoi :
    mat : matrice de confusion
    """
    mat=[[0 for i in range(len(categories))] for j in range(len(categories))]   # initialisation de la matrice
    for i in range(len(test)):
        j = ..................................  # rang de la categorie prédite
        ri = ................................. # categorie réelle
        ..................................... # remplissage de la case (i,j)
    return mat
```

Cette matrice de confusion peut aider à **choisir la meilleure valeur de k : celle qui permet d'avoir la somme sur la diagonale la plus élevée, et les points hors de la diagonale les plus proche de zéro.**

Pour un ensemble de $N = 500$ points (semblables aux données précédentes), 3/4 ayant servis de données d'apprentissage et 1/4 de données tests, on obtient les matrices de confusion M_k suivantes :

$$M_1 = \begin{pmatrix} 79 & 8 \\ 5 & 33 \end{pmatrix} \quad M_{50} = \begin{pmatrix} 81 & 3 \\ 3 & 38 \end{pmatrix} \quad M_{200} = \begin{pmatrix} 82 & 8 \\ 2 & 33 \end{pmatrix} \quad M_{300} = \begin{pmatrix} 84 & 41 \\ 0 & 0 \end{pmatrix}$$

III Apprentissage non supervisé : algorithme des k-moyennes

Nous allons maintenant nous intéresser à un problème plus complexe : comment comprendre la structure des données sans aider la machine en lui donnant des données d'apprentissage étiquetées ? En pratique il s'agit pour la machine de regrouper les données proches au sein d'une même classe, à charge pour l'humain d'interpréter ensuite ce regroupement.

Nous considérons un ensemble de points qui a l'allure ci-dessous (figure 2). Nous observons que les points semblent se regrouper dans trois classes.

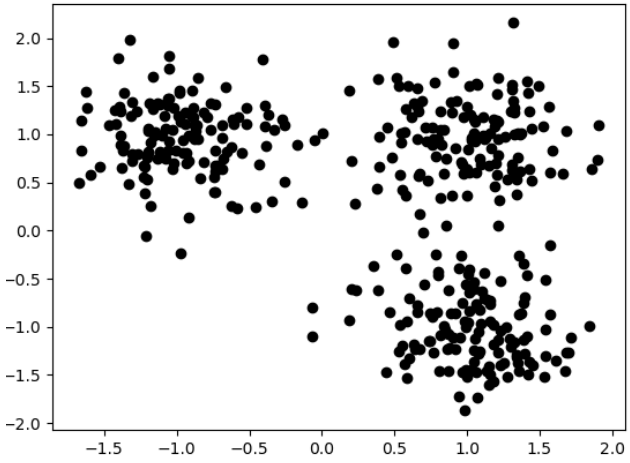

FIGURE 2 – Les données semblent se regrouper en trois classes

L'objectif est de permettre à la machine d'identifier ces trois classes. Nous allons tout de même l'aider en lui imposant le nombre de classes (3 ici). Mais le nombre de classe n'est pas toujours facilement identifiable et il est nécessaire d'essayer plusieurs nombres de classes.

III.1 Définitions

Définitions : Barycentres et moments d'inerties

On note $C_0, ..., C_{k-1}$ les k classes, et on définit :

— le barycentre μ_j de la classe C_j : $\mu_j = \dfrac{1}{\text{card}(C_j)} \displaystyle\sum_{x \in C_j} x$

— le moment d'inertie m_j de la classe C_j : $m_j = \displaystyle\sum_{x \in C_j} \|x - \mu_j\|^2$

L'objectif de la machine est de minimiser la somme des moments d'inertie $\displaystyle\sum_{j=1}^{k} m_j$.

III.2 Algorithme des k-moyennes

Le calcul de la classification optimale, c'est-à-dire minimisant la somme des moments d'inertie, est un problème très coûteux en temps, aussi allons nous employer un algorithme glouton. Ce dernier nous assurera d'obtenir in fine une « bonne » solution, mais pas forcément la meilleure.

Méthode : Algorithme des k-moyennes

L'algorithme des k-moyennes consiste à réaliser la succession d'opérations suivantes :

(1) on choisit aléatoirement k barycentres $\mu_0, ..., \mu_{k-1}$;

(2) chacun des points du nuage est associé au barycentre $\mu_{j_{\min}}$ le plus proche ; on crée ainsi k classes $C_0, ..., C_{k-1}$;

(3) on calcule les barycentres $\mu_0, ..., \mu_{k-1}$ de ces classes, qui remplacent les valeurs précédentes (des barycentres) ;

(4) on reprend le calcul à partir de l'étape 2*.

Nouas admettons que cet algorithme converge, autrement dit qu'à partir d'une certaine étape les centres $\mu_0, ..., \mu_{k-1}$ ne se déplacent plus et donc que les classes $C_0, ..., C_{k-1}$ sont stabilisées.

Mais répétons-le, **cet algorithme donne une configuration pour laquelle la somme des moments d'inertie est un minimum local, mais pas forcément le minimum global. Le résultat dépendant notamment du point de départ** (choix aléatoire des centres).

III.3 Implémentation en python

Nous allons implémenter l'algorithme à l'aide de plusieurs fonctions :

1. Fonction `barycentre` qui prend en entrée une liste de listes représentant une liste de données représentées par leurs coordonnées, et qui renvoie les coordonnées du barycentre de ces points sous la forme d'une liste.

```
def barycentre(L:list)->list:
    """
    renvoi la liste des coordonnées du barycentre de L
    """
    m=len(L[0]) # dimension de l'espace
    coord_G=m*[0] # initialisation des coordonnées du barycentre
    for i in range(.......) : # on calcule la coordonnés x_{Gi} de G
        for j in range(............): # parcours de toutes les données
            ................................ # on somme les coordonnées
        ......................... # calcul de la moyenne.
    return coord_G
```

2. Fonction `initialisation(X:list,k:int)` qui initialise les `k` centres à partir des données `X`.

```
def initialisation(X:list,k:int)->list:
    """
    Après mélange des données de X, renvoi les k premiers éléments
    """
    shuffle(X) # on mélange les données X pour effectuer un "vrai" tirage au
   sort
    return X[:k] # k premiers éléments de k
```

3. Fonction `calculer_centre(classes:list)->list` qui renvoie la liste des centres de chaque classe.

```
def calculer_centre(classes:list)->list:
    """
    Entrée : classes : liste des classes
    Sortie : centres liste des coordonnées des centres de chaque classes
    """
    k=len(classes)
    centres=[]
    for i in range(k): # pour chaque classe, on calcule son barycentre
        ........................................
    return centres
```

4. Fonction `plus_proche(x:list,centres:list)->int` qui renvoie le numéro de la classe la plus proche de `x` parmi `centres`, c'est-à-dire la classe telle que la distance de `x` à `centres[i]` soit minimale.

```
def plus_proche(x:list,centres:list)->int:
    """
    Entrées :
    x : une donnée
    centres : liste des k centres
    Renvoi :
    imin : numéro de la classe dont le centre est le plus proche de x
    """
    # Calculs des distances centre x et centres
    distances=[]
    k=len(centres)
    ........................................
    ........................................
    # Détermination du rang de la distance minimale
    ................
    ................
    for i in range(k):
        ................
            ................
            ................
    return imin # classe dont le centre est le plus proche de X
```

5. Fonction `calculer_classes(X:list,centres:list)->list` qui renvoie une liste `classes` telle que `classes[i]` soit la liste des données de `X` dont le centre le plus proche est `centre[i]`.

```
def calculer_classes(X:list,centres:list)->list:
    """
    Entrées :
    X : liste des données
    centres : liste des k centres
    Renvoie :
    classes : liste de k listes regroupant les données de X dans les k classes (listes)
    """
    k=len(centres) # nombre de classes
    classes=.......................................... # initialisation à une liste de k listes vide : une pour chaque classe
    for j in range(len(X)): # parcours des données
        imin=.......................................... # centre le plus proche de X[j]
        ..................................... # on ajoute la donnée j à la classe déterminée
    return classes
```

6. Fonction `k_moyennes(X:list,k:int)->list` appliquant l'algorithme des k-moyennes à `X`.

```
def k_moyennes(X:list,k:int)->list:
    """
    Algorithme des k-moyennes appliqué à la liste des données X
    Entrées :
    X : liste des données
    k : nombre de classes
    Renvoie :
    classes : liste de k listes regroupant les données de X dans les k classes (listes)
    """
    centres=.....                    # initialisation des centres
    classes= .....                   # calcul des classes correspondant à ces centres
    centres_new=.....                # calcul des nouveaux centres
    while centres_new!=centres: # tant que les nouveaux sont différents des anciens
        classes=.....                # calcul des nouvelles classes
        # calcul des nouveaux centres affectés à centres_new, tout en affectant à centres la liste précédente centres_new
        ..................................................
    return classes
```

Le nombre d'itérations nécessaires pour converger dépend des données, du nombre de classes imposé, et du premier choix aléatoire des trois barycentres. Sur le jeu de données ci-dessous, le nombre d'itérations nécessaire pour que l'algorithme converge a varié entre 3 et 12, cela dépend du choix aléatoire des trois barycentres lors de l'initialisation. Notamment, les fois où deux (ou trois) barycentres étaient proches ont nécessité davantage d'itérations.

On constate que l'algorithme converge vers des classes correctement identifiées, avec seulement quelques points mal identifiés.

Les erreurs (nombres et natures) dépendent des données, du nombre de classes imposé, et du premier choix (aléatoire) des trois barycentres.

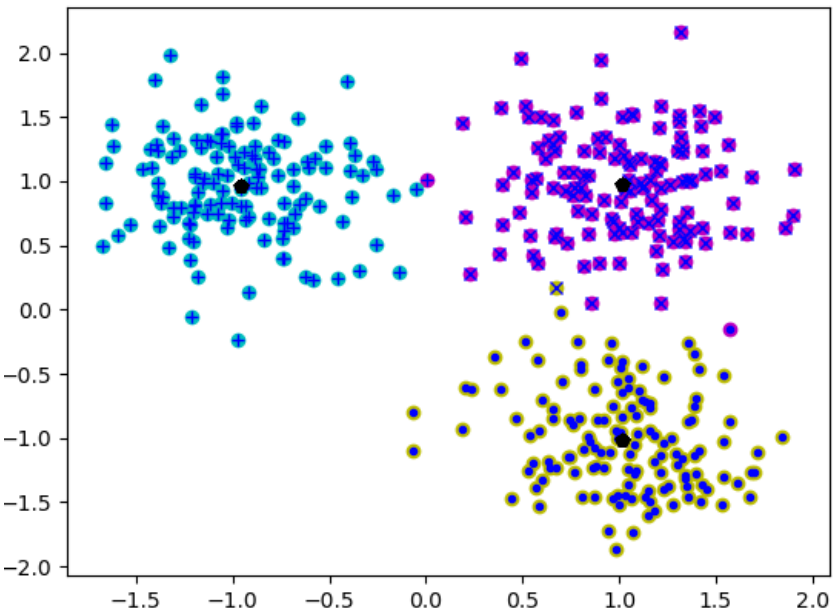

FIGURE 3 – Les données de départ sont représentées, avec des points de trois couleurs différentes pour distinguer les groupes.
Les trois classes identifiées par l'algorithme des k-moyennes sont représentées par des +, × et • bleus.
Les barycentres des classes identifiées sont représentés par des pentagones noirs.