3 Algorithmes d'apprentissage

Extrait du programme officiel :

Notions	Commentaires
Algorithme des k plus proches voisins avec distance euclidienne.	Matrice de confusion. Lien avec l'apprentissage supervisé.
Algorithme des k-moyennes.	Lien avec l'apprentissage non supervisé. La démonstration de la convergence n'est pas au programme. On observe des convergences vers des minima locaux.
Mise en œuvre	
La connaissance dans le détail des algorithmes de cette section n'est pas un attendu du programme. Les étudiants acquièrent une familiarité avec les idées sous-jacentes qu'ils peuvent réinvestir dans des situations où les modélisations et les recommandations d'implémentation sont guidées, notamment dans leurs aspects arborescents.	

Table des matières

I Intelligence Artificielle

L'Intelligence Artificielle est une branche de l'informatique sous les feux de la rampe actuellement.

Parmi les applications récentes, citons les moteurs de recherche, les programmes de jeux battant des êtres humains (`Deep Blue` pour les échecs, `AlphaGo` pour le jeu de go), la voiture autonome, la reconnaissance vocale, la reconnaissance faciale, la traduction instantanée, la détection de spams, jusqu'à tout récemment avec `ChatGPT` (*Generative Pre-trained Transformer for Chat*), une impressionnante intelligence artificielle de traitement du langage naturel.

Une des techniques les plus utilisées en intelligence artificielle est l'« Apprentissage Automatique » (*Machine Learning* en anglais), qui a connu un essor important ces dernières années grâce à la disponibilité d'importantes bases de données (*Big Data*) et une puissance de calcul démultipliée par la généralisation de la pratique du calcul parallèle.

Par exemple, la technique d'apprentissage statistique dite de « Réseau de Neurones Profond » (*Deep Neural Networks* en anglais, ou `DNN`) inventée dans les années 90 a connu un essor important ces dernières années (et apparaît même dans les nouveaux programmes de SII).

Dans ce chapitre, nous allons considérer deux techniques d'apprentissage automatique appelées « k plus proches voisins » (en anglais *k-Nearest Neigbors* ou `k-NN`) et « k moyennes » (en anglais *k-means*).

- La première appartient à la catégorie des algorithmes de classification **supervisée** : on dispose de données (dites d'apprentissage) pour lesquelles une classe a été attribuée (par exemple des images représentent un chien ou un chat). On cherche alors à construire une fonction qui permet de classifier de nouvelles données.
- La seconde appartient à la catégorie des algorithmes de classification **non supervisée** : elle consiste à partitionner des données en k classes sans qu'aucune de ces données n'ait été préalablement classifiée par un être humain.

II Apprentissage supervisé : l'algorithme des k plus proches voisins.

1 Présentation de l'algorithme

L'algorithme des k plus proches voisins est un algorithme de classification supervisée : il demande de connaître, pour un nombre important de données, la classe (catégorie) de chacune de ces données, et tente d'en déduire la classe d'une nouvelle donnée en s'intéressant à ses k plus proches voisins, dans un sens à définir.

Définition 1 : Algorithme des k plus proches voisins

Soit $k \in \mathbb{N}^*$, et $((x_i, c_i))_{1 \leqslant i \leqslant n}$ une base d'apprentissage : les $x_i \in \mathbb{R}^d$ sont les données, les c_i sont des classes associées à chaque donnée, dans un ensemble $\mathscr{C}$ de classes (de cardinal assez petit).

L'**algorithme des k plus proches voisins** associe à une nouvelle donnée $x \in \mathbb{R}^d$, la classe la plus représentée parmi les classes des k éléments de $(x_i)_{1 \leqslant i \leqslant n}$ les plus proches de x.

Cet algorithme dépend de deux choix, qui peuvent influer sur ses performances :
- celui de l'entier k ;
- celui de la définition de la distance entre les données.

Pour ce dernier, le programme stipule que seul le cas de la distance euclidienne habituelle est utilisée ici.

Algorithme k-NN

L'algorithme se découpe alors en trois phases :

1. Calculer la distance $d_i = \|x - x_i\|$ de x à chaque donnée x_i de la base (on peut se contenter de son carré).
2. Trier par ordre croissant ces distances d_i pour obtenir $d_{i_1} \leqslant d_{i_2} \leqslant \cdots \leqslant d_{i_n}$ et ne garder que les k premiers indices $i_1, \ldots, i_k$.
3. Déterminer la classe majoritaire parmi les $(c_{i_j})_{1 \leqslant j \leqslant k}$. En cas d'égalité, il faut décider d'un critère pour départager, par exemple en pondérant par un cœfficient inversement proportionnel à la distance à x. Lorsqu'il n'y a que deux classes, il peut aussi être intéressant de choisir k impair pour éviter les égalités.

On teste l'algorithme sur la base de donnée Iris : chaque fleur de la base est représentée par un point de coordonnée (longueur des sépales, largeur des sépales), dont la couleur correspond à l'espèce (setosa, versicolor ou virginica).

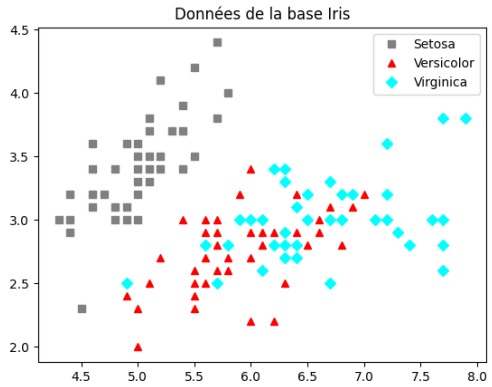


Figure 1 – Les données de la bases Iris, en dimension 2

Ici, $d = 2$. On teste, avec une nouvelle fleur, l'algorithme des 7 plus proches voisins. Voici le résultat.

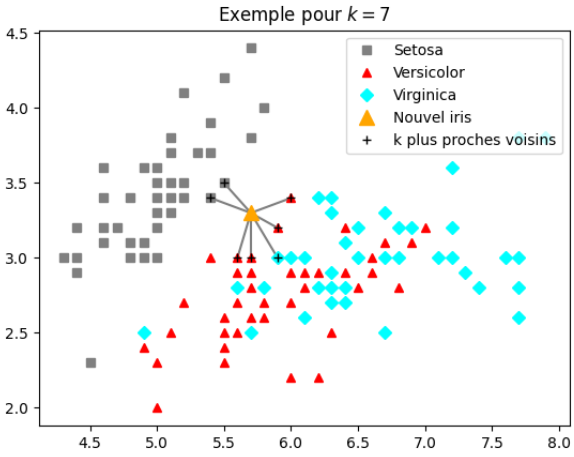


Figure 2 – Application de 7-NN

Voici, pour différentes valeurs de k (1, 3, 7, 13, 25, 31), le résultat de l'algorithme en chaque point :

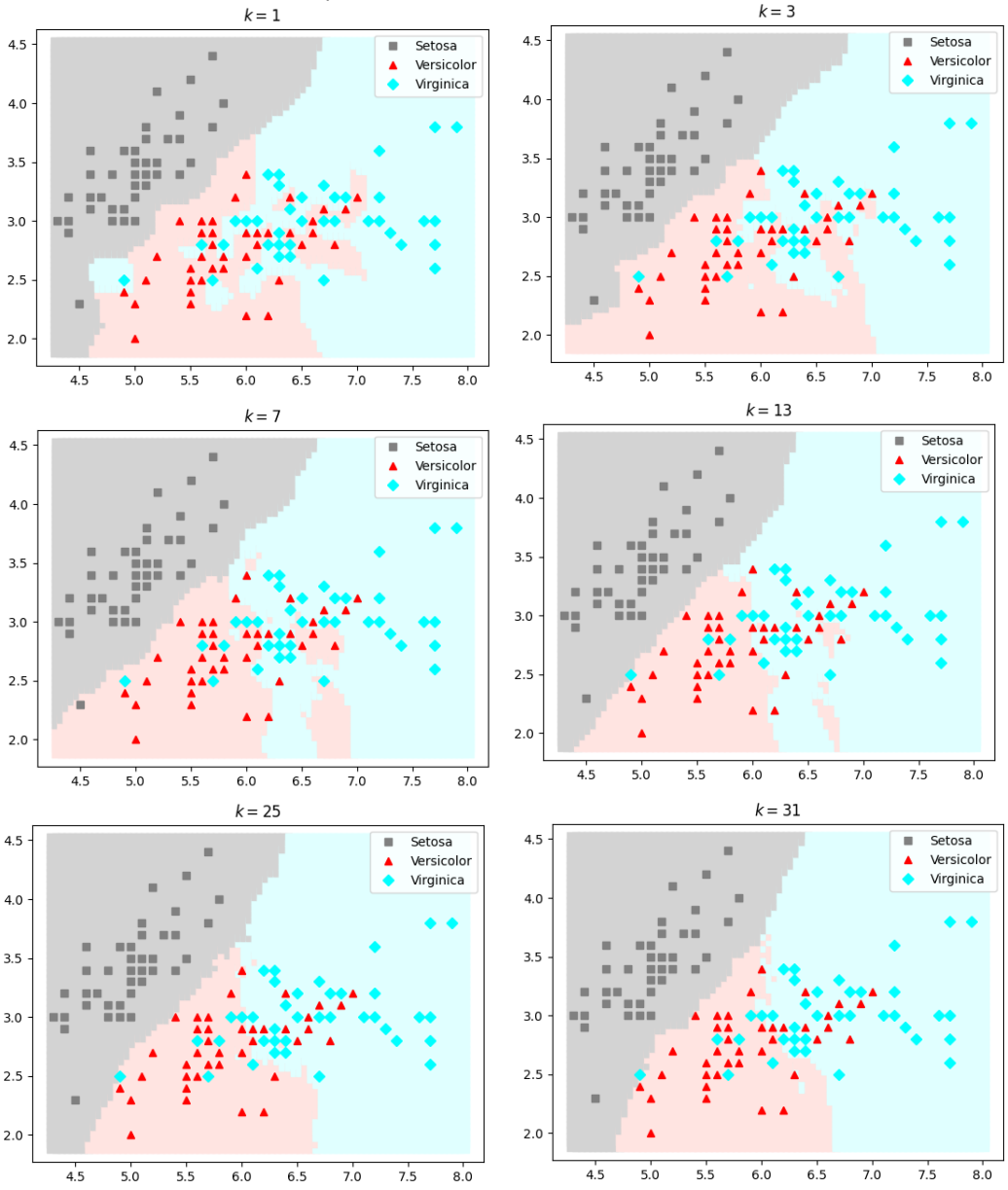


Figure 3 – Résultats en faisant varier k

2 Évaluation de la qualité de l'algorithme

Pour pouvoir évaluer la qualité de l'algorithme, on n'utilise pas l'intégralité des données pour l'apprentissage.

On va scinder ces données en une partie pour l'apprentissage, et une partie pour les tests. La règle est de ne jamais effectuer les tests sur les données d'apprentissage directement. On peut par exemple garder 70% des données pour l'apprentissage et 30% pour les tests.

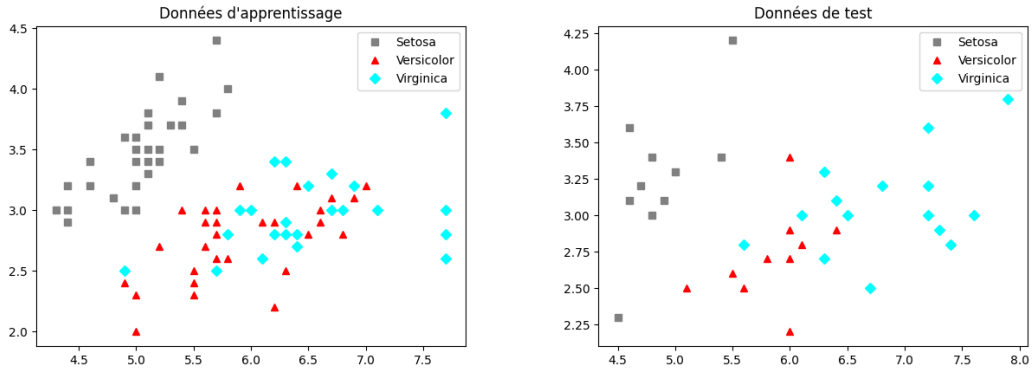


Figure 4 – Séparation des données d'apprentissage et des données de test

Définition 2 : Matrice de confusion

La **matrice de confusion** d'un résultat de classification sur une base de test est une matrice carrée d'ordre le nombre de classes, dont le cœfficient (i, j) compte le nombre d'observation pour laquelle la classe réelle est i et la classe estimée est j.

Ainsi, plus la matrice de confusion est proche d'une matrice diagonale, meilleur est l'apprentissage. Cela peut permettre de choisir une valeur de k bien adaptée.

$k=1$	setosa	versicolor	virginica
setosa	9	1	0
versicolor	0	5	5
virginica	0	6	9

$k=3$	setosa	versicolor	virginica
setosa	9	1	0
versicolor	0	5	5
virginica	0	6	9

$k=7$	setosa	versicolor	virginica
setosa	10	0	0
versicolor	0	5	5
virginica	0	4	11

$k=13$	setosa	versicolor	virginica
setosa	10	0	0
versicolor	0	6	4
virginica	0	2	13

$k=25$	setosa	versicolor	virginica
setosa	10	0	0
versicolor	0	9	1
virginica	0	3	12

$k=31$	setosa	versicolor	virginica
setosa	10	0	0
versicolor	0	9	1
virginica	0	3	12

3 Code python

Ici, on a choisi d'utiliser les facilités offertes par `python`. Pas nécessairement possible aux concours !

Python

```python
## Lecture des données dans un fichier csv
import csv
with open('iris.csv', newline='') as fichierIris:
    lecture = csv.reader(fichierIris, delimiter=',')
    # transformation de lecture en liste, omission de l'entête
    fleurs = list(lecture)[1:]

# On transforme les deux premières données en flottant :
for i, (longueur, largeur, espece) in enumerate(fleurs):
    fleurs[i] =  float(longueur),  float(largeur), espece

## Algorithme k-NN
def distance(iris1, iris2):
    """ iris1 -- liste de type [float, float, espèce] ou [float, float]
        iris2 -- même nature qu'iris1
    renvoie la distance euclidienne *au carré* entre les deux iris."""
    longueur1, largeur1 = iris1[:2]
    longueur2, largeur2 = iris2[:2]
    dx = longueur1 - longueur2
    dy = largeur1 - largeur2
    return dx ** 2 + dy ** 2

def kPlusProchesVoisins(iris, k, fleurs):
    """ iris -- liste de type [float, float]
        k -- entier naturel, compris entre 1 et le nombre total de fleurs
        fleurs -- liste de toutes les fleurs
    renvoie la liste des k plus proches voisins de iris."""
    nb_fleurs = len(fleurs)
    # On crée la liste des distances entre iris et les données du fichier.
    liste_distances = [distance(iris, fleur) for fleur in fleurs]
    # On ajoute la donnée "indice".
    indices = list(range(nb_fleurs))
    # On trie M suivant les distances, ordre croissant.
    indices.sort(key=lambda i: distance(iris, fleurs[i]))
    # On récupère uniquement les indices des k premiers éléments.
    I = indices[:k]
    # On renvoie la liste des fleurs correspondantes.
    return [fleurs[k] for k in I]

def classification(iris, k, fleurs):
    """ iris -- liste de type [float, float]
        k -- entier naturel, compris entre 1 et le nombre total de fleurs
        fleurs -- liste de toutes les fleurs
    renvoie l'espèce la plus fréquente parmi les k plus proches voisins de iris."""
    # Dictionnaire de comptage des espèces
    compte = {'Iris-setosa': 0, 'Iris-versicolor':0, 'Iris-virginica':0}

    # Liste des k plus proches voisins
    ppv = kPlusProchesVoisins(iris, k, fleurs)

    # Mise à jour compteurs
    for _, _, espece in ppv:
        compte[espece] += 1

    # On renvoie l'espèce réalisant majoritaire.
    return max(compte, key=lambda espece: compte[espece])
    # ou bien : max(compte, key=compte.get)
```

III Apprentissage non supervisé : l'algorithme des k-moyennes.

1 Présentation

L'algorithme que nous allons présenter dans cette partie est un algorithme d'apprentissage non supervisé : l'idée est de **partitionner des données** (en anglais : **clustering**) sans avoir de données déjà classées a priori.

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.

Un exemple où les données (des points de $\mathbb{R}^2$, ici) semblent se regrouper en trois classes :

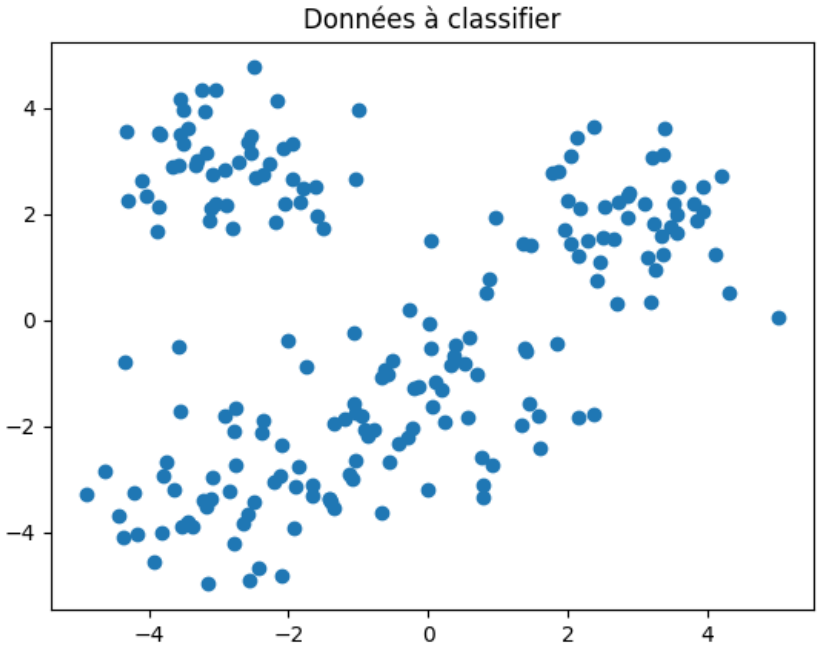


Figure 5 – Données semblant se partitionner en 3 classes

2 Centres, variances et inertie

Définition 3 : Centre

Le **centre** ou **isobarycentre** d'un ensemble X de n vecteurs est le vecteur

$$\overline{X} = \frac{1}{n} \sum_{x \in X} x.$$

Exercice 1

Écrire une fonction `centre(X)` renvoyant le centre de la liste éventuellement vide de vecteurs `X` (de même dimension `dim` : une variable globale).

Python

```python
def centre(X):
    """
X : liste de vecteurs (listes) de même dimension dim (variable globale) éventuellement vide.
Renvoie le vecteur isobarycentre des vecteurs présents dans X."""
    n = len(X)
    c = [0 for _ in range(dim)]
    if X == []:
        return X
    for x in X:
        for i in range(dim):
            c[i] += x[i]
    for i in range(dim):
        c[i] /= n
    return c
```

Dans toute la suite, on note d la distance euclidienne associée à la norme euclidienne $\|\cdot\|$.

Définition 4 : Variance

La **variance** ou **moment d'inertie** $V(X)$ d'un ensemble de vecteurs X est définie par

$$V(X) = \sum_{x \in X} \mathrm{d}\left(x, \overline{X}\right)^2 = \sum_{x \in X} \left\| x - \overline{X} \right\|^2.$$

La variance est un indicateur de dispersion : plus les points de X seront proche de leur centre $\overline{X}$, plus cette variance sera faible.

Définition 5 : Inertie

L'**inertie** d'une partition (*clustering*) de X en k parties $X_1, \ldots, X_k$ (classes ou *clusters*) est définie par

$$I = \sum_{i=1}^{k} V(X_i).$$

L'objectif est alors de trouver une partition de X minimisant l'**inertie** I.

On cherche donc une partition de X permettant d'avoir des données les plus proches possible du centre de leur classe.

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.

3 L'algorithme des k-moyennes

Algorithme k-MEANS

L'algorithme se déroule avec les étapes suivantes.

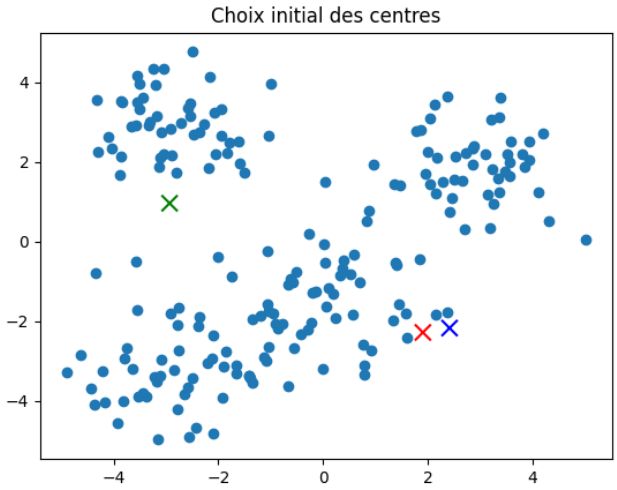
1. Soit $c_1, \ldots, c_k$ des vecteurs (par exemple choisis aléatoirement).
2. Associer chaque donnée x à la classe X_i telle que $\mathrm{d}(x, c_i)$ soit minimale.
3. Recalculer les centres des classes $c_i = \overline{X_i}$.
4. Si les centres ont changé, revenir à l'étape 2.

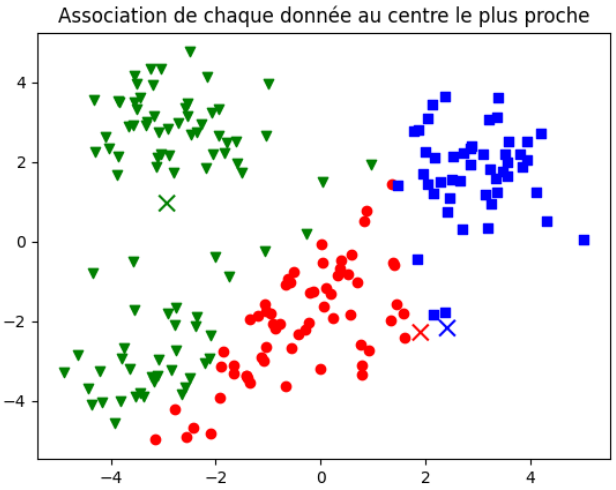
Remarque 1

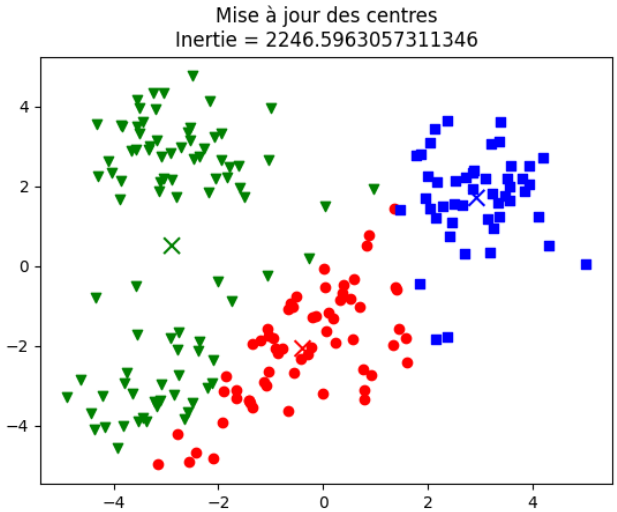
⚠ Dans l'algorithme des k-moyennes, k est le nombre de classes alors que dans l'algorithme des k plus proches voisins, k est le nombre de voisins.

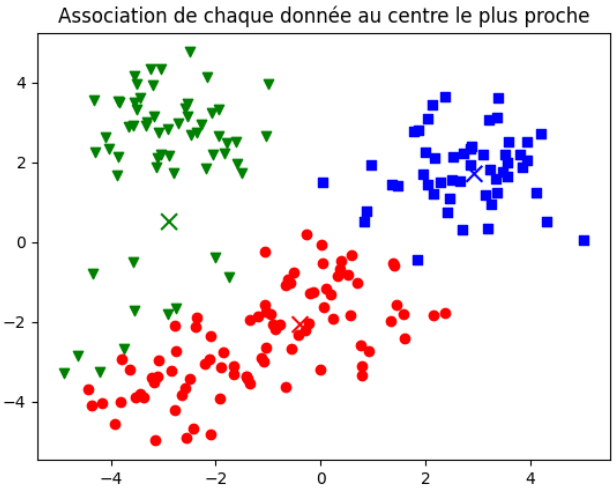
4 Exemple d'exécution

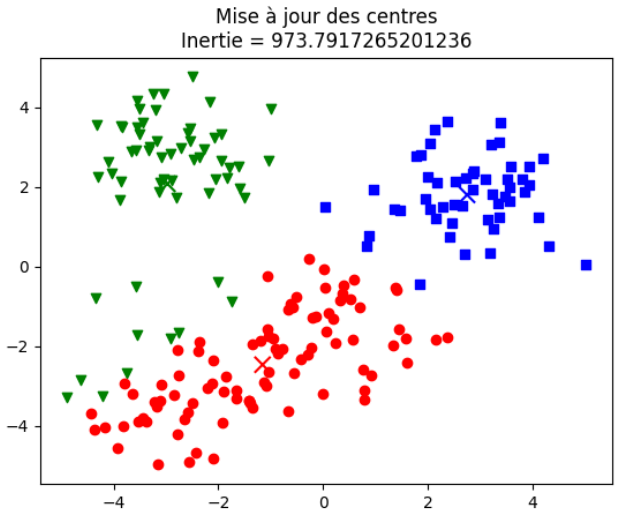
Exemple d'exécution de l'algorithme, les centres initiaux sont aléatoires.

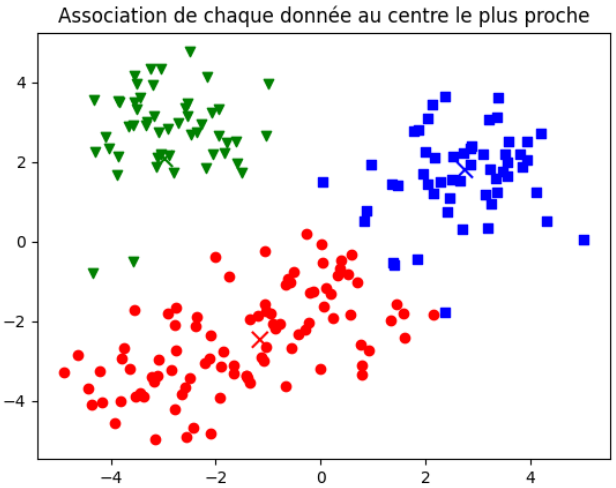


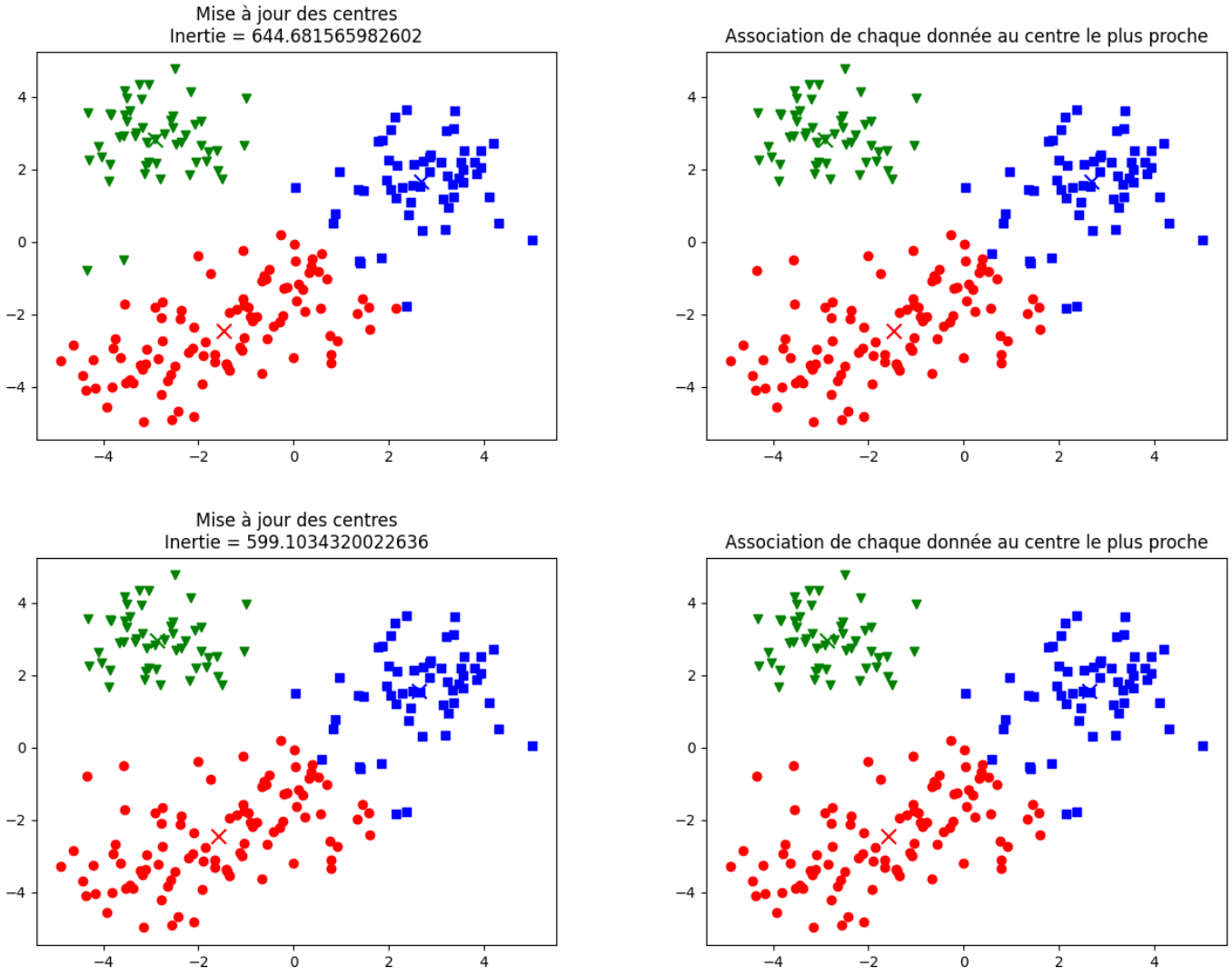












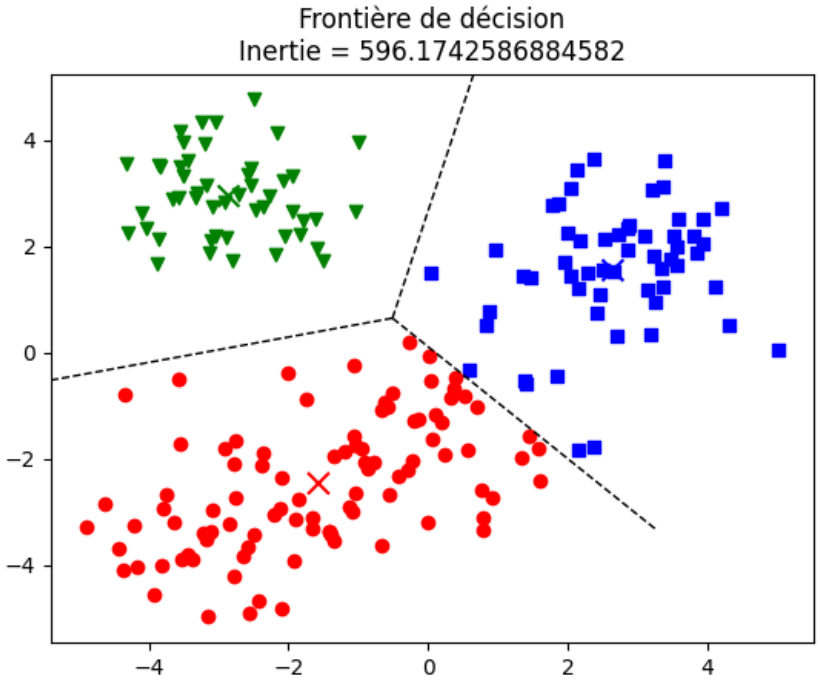


Figure 6 – Résultat de l'exécution

5 Terminaison

Théorème 1 : Hors programme

L'algorithme des k-moyennes termine.

Repose sur le fait qu'il n'y a qu'un nombre fini de k-partitions et que l'inertie, positive, décroît strictement.

Cependant, on n'est pas certain d'avoir atteint un minimum global de l'inertie, il s'agit peut-être d'un minimum local seulement.

6 Implémentation

Exercice 2

On utilise une liste `classes` de taille k telle que `classes[i]` est la liste des données de `X` associées à la classe i. Écrire une fonction `calculer_centre(classes)` renvoyant la liste des centres de chaque classe.

Python

```python
def calculer_centres(classes):
    """
classes : liste de listes de vecteurs.
renvoie la liste des centres de chaque classe."""
    centres = []
    for i in range(len(classes)):
        centres.append(centre(classes[i]))
    return centres
```

Exercice 3

Écrire des fonctions `d2(x, y)` calculant le carré de la distance euclidienne du vecteur `x` au vecteur `y` puis `plus_proche(x, centres)` renvoyant le numéro i 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.

Python

```python
def d2(x, y):
    """
x : vecteur (liste)
y : vecteur (liste) de même taille que x
renvoie le carré de la distance euclidienne de x à y."""
    s = 0
    for i in range(len(x)):
        s += (x[i] - y[i]) ** 2
    return s

def plus_proche(x, centres):
    """
x : vecteur
centres : liste de vecteurs
renvoie l'indice i tel que la distance de x à centres[i] soit minimale."""
    imin = 0
    for i in range(len(centres)):
        if d2(x, centres[i]) < d2(x, centres[imin]):
            imin = i
    return imin
```

Exercice 4

Écrire une fonction `calculer_classes(X, centres)` renvoyant une liste `classes` telle que `classes[i]` soit la liste des données de `X` dont le centre le plus proche est `centre[i]`.

Python
```python
def calculer_classes(X, centres):
    """
X : liste de vecteurs
centres  : liste de vecteurs
renvoie une liste classes telle que classes[i] soit la liste des données de X dont le centre le plus
proche est centre[i]."""
    classes = [[] for i in range(len(centres))]
    for x in X:
        classes[plus_proche(x, centres)].append(x)
    return classes
```

Exercice 5

Écrire une fonction `kmeans(X, centres)` appliquant l'algorithme des k-moyennes à `X` en partant de centres `centres` et renvoyant la liste des classes obtenues.

Python
```python
def kmeans(X, centres):
    """
X : liste de vecteurs
centres  : liste de vecteurs
renvoie une liste classes obtenue par l'algorithme des k-moyennes."""
    centres2 = None
    while centres != centres2:
        centres2 = centres
        classes = calculer_classes(X, centres2)
        centres = calculer_centres(classes)
    return classes
```

7 Choix des centres initiaux

L'inertie du clustering obtenu à la fin de l'algorithme dépend des centres initiaux.

Il y a plusieurs façons de les choisir. Par exemple :
- Aléatoirement dans l'espace.
- Aléatoirement parmi les données.
- Lancer l'algorithme plusieurs fois avec des centres initiaux différents et conserver la meilleure solution (celle d'inertie minimum).

8 Choix de k

Comment choisir le nombre k de classes ?

On peut calculer l'inertie obtenue pour différentes valeurs de k.

Cependant, plus k est grand, plus l'inertie diminue jusqu'à valoir 0 si k est égal au nombre de données (ce qui n'a aucun intérêt).

On choisit donc la plus grande valeur de k pour laquelle l'inertie diminue de façon significative.

On peut utiliser la méthode du coude : on choisit la valeur de k la plus grande pour laquelle l'inertie diminue de façon significative (3 ou 4 ici).

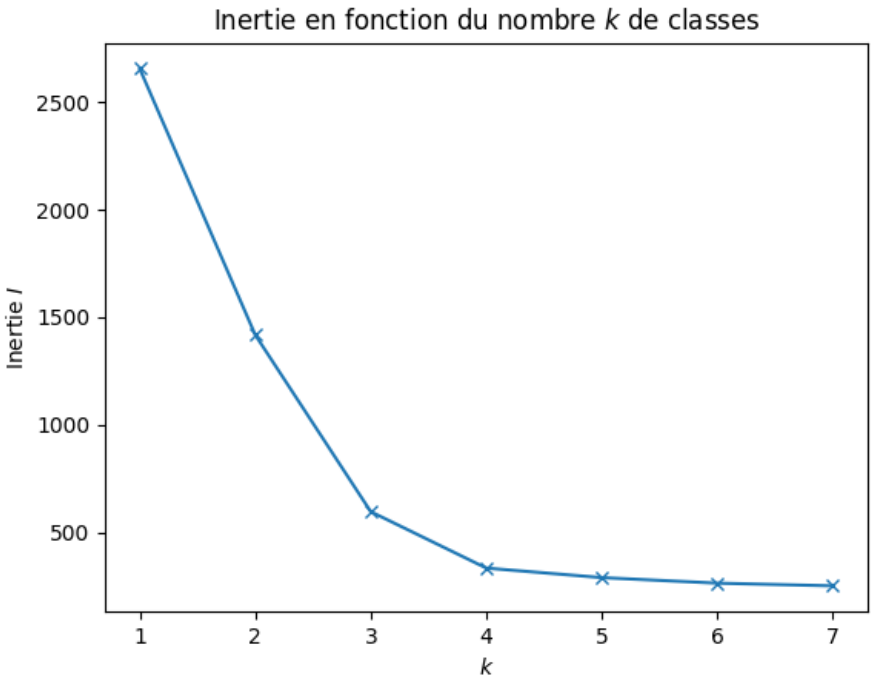


Figure 7 – Méthode du coude (*elbow method*) donnant $k = 3$ ou 4

9 Limites

L'algorithme des k-moyennes ne fonctionne que sur des données linéairement séparables (c'est-à-dire pouvant être séparées par des hyperplans).

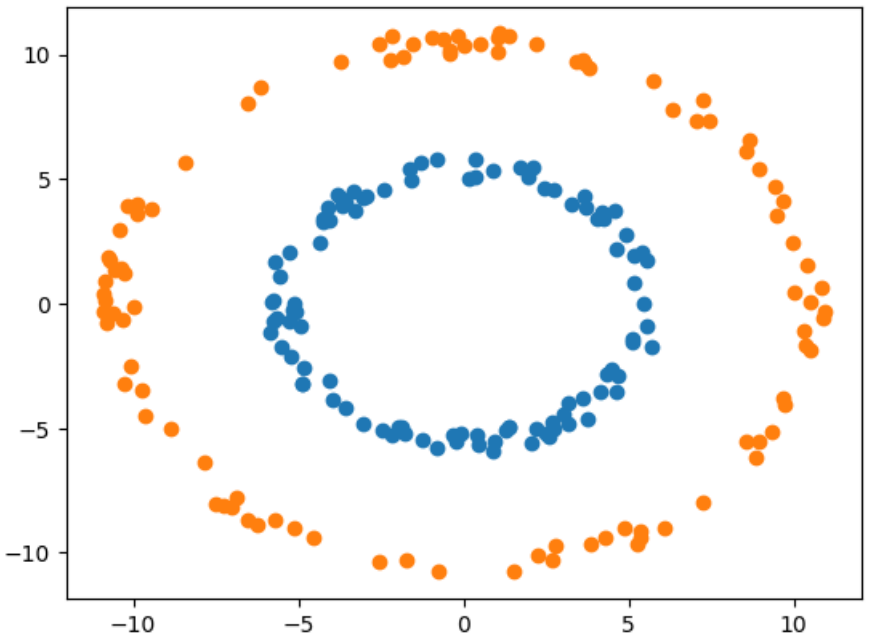


Figure 8 – Exemple de données non linéairement séparables