# Apprentissage Automatique

# I Présentation

## A Apprentissage Automatique

La capacité d'apprendre est essentielle à l'être humain (reconnaitre une personne, un objet, une situation,...). On distingue **deux types d'apprentissage** en général :

- l'apprentissage **par cœur** : on mémorise telles quelles des informations explicites.
- l'apprentissage **par généralisation** : on apprend à partir de quelques exemples des règles implicites qui vont permettre de reconnaitre de nouveaux exemples.
  *Exemple : lorsqu'on montre sur une page quelques carrés, quelques cercles et quelques triangles à un jeune enfant pour la première fois, celui-ci saura déterminer "naturellement" le type d'autres exemples des ces figures présents sur la 2e page, même si celles-ci ne sont pas de la même taille, de la même couleur ou au même emplacement.*

Il est facile de reproduire l'apprentissage par coeur sur un ordinateur, c'est plus difficile pour l'apprentissage par généralisation.

**L'apprentissage automatique est une tentative de reproduire cette faculté humaine d'apprentissage dans des systèmes artificiels**.

**Définition**

L'apprentissage automatique (ou *machine learning* en anglais) est la branche de l'intelligence artificielle qui concerne le développement d'algorithmes permettant de rendre une machine capable d'accomplir des tâches complexes sans avoir été explicitement programmée dans ce but.

Plus généralement, l'apprentissage automatique **se libère de la nécessité de formaliser des règles explicites** et se caractérise par la présence de deux phases :

— **la phase d'entrainement (ou d'apprentissage)** : on choisit un modèle et on lui fournit un grand nombre d'exemples/d'entrées bien choisis afin que ce dernier puisse "apprendre". Cette phase est réalisée avant l'utilisation pratique du modèle.

— **la phase d'inférence** : on rend le modèle entrainé autonome en lui permettant de traiter seul de nouvelles entrées.

## B Différents Types

Une première distinction porte sur la nature de la phase d'apprentissage.

En **apprentissage supervisé**, on a des données d'entrainements étiquetées $\mathcal{D} = [(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)]$. Autrement dit, on a récupéré $n$ **entrées** (notées chacune $x_i$) auxquelles on a déjà associé une **sortie cible** $y_i$ (l'étiquette) et on souhaite que l'algorithme devienne capable, après la phase d'apprentissage, de prédire correctement la cible sur une nouvelle entrée non étiquetée.

Illustration

chat renard oiseau chat chien


guepard


chat


chien


papillon


chien

*Ci-dessus :*
*Jeu de 10 données étiquetées*
*$x_1$ est la première image en haut à gauche. Son étiquette est $y_1$ = "chat"*
*Ci-contre : Nouvelle entrée (non étiquetée) dont il faut prédire la cible/l'étiquette*


?

L'apprentissage supervisé est généralement effectué dans l'un des deux contexte suivants :
- **Classification** : lorsque la cible est discrète (ne prend qu'un nombre fini de "valeurs") et représente une catégorie (ou une classe).
  *Exemple : algorithme de classification des images d'animaux (plusieurs catégories), algorithme de diagnostic médical qui doit déterminer si le patient est "malade" ou pas (2 catégories)*
- **Régression** : lorsque la cible prend une valeur réelle.
  *Exemple : algorithme de prédiction du cours d'une action en bourse*

Dans le II, nous étudierons en détails un exemple d'algorithme d'apprentissage supervisé : **L'algorithme des $k$ plus proches voisins**.

En **apprentissage non-supervisé**, les données d'entrées **ne sont pas étiquetées**, on les note $\mathcal{D} = [x_1, x_2, ..., x_n]$.
Le programme est en quelque sorte livré à lui-même : il doit identifier automatiquement des caractéristiques communes aux entrées/observations et modéliser la structure sous-jacente dans $\mathcal{D}$.
L'apprentissage non supervisé fait appel principalement à des algorithmes de **regroupement** (ou *clustering* en anglais) : on sépare $\mathcal{D}$ en un certain nombre fixé de groupes, de façon à ce que les membres d'un même groupe ont des données similaires entre eux et différentes de celles des autres groupes.
Autrement dit, on cherche une partition "cohérente" de $\mathcal{D}$ en un nombre fixé de groupes.

Illustration

Voici un jeu de 7 données sans étiquettes :

↪ Classification en 2 groupes :

Classe 1 Classe 2

Dans le III, nous étudierons en détail un exemple d'algorithme d'apprentissage non-supervisé : **l'algorithme des $k$ moyennes**.

# II Algorithme des $k$ plus proches voisins

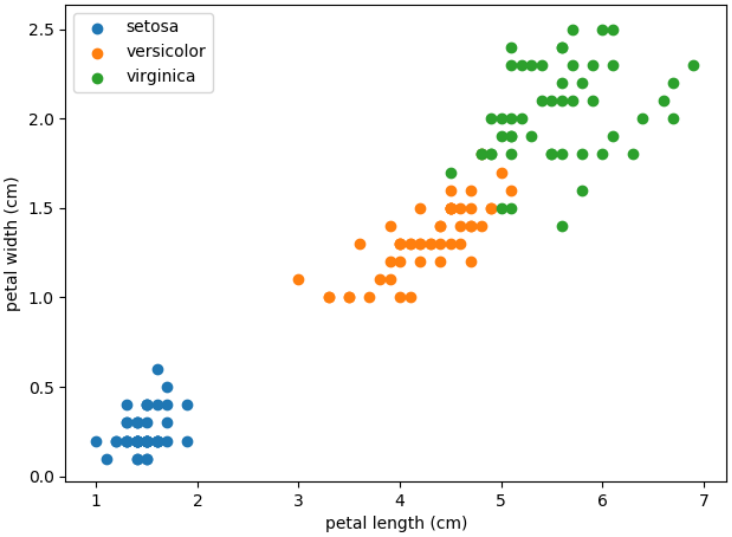
## A Mise en situation

Nous allons considérer l'exemple classique de la classification d'une fleur de type Iris en trois variétés numérotées 1,2,3 : setosa, versicolor et virginica.

Chaque fleur observée est associée à deux nombres, la largeur et la longueur de ses pétales, et peut donc être vue comme un élément de $\mathbb{R}^2$.

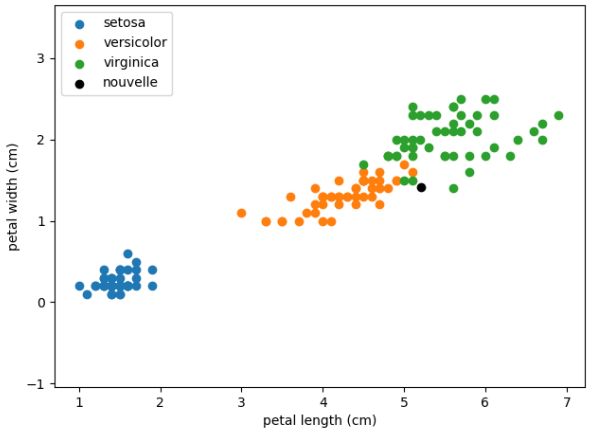
Autrement dit, le $i^e$ Iris étudié est caractérisé par un vecteur $x_i$ de $\mathbb{R}^2$ et par un nombre $y_i$ correspondant au numéro de sa classe.

On peut représenter ce jeu de données graphiquement (ci-contre) et observer que les 3 classes ont naturellement tendance à créer des amas.

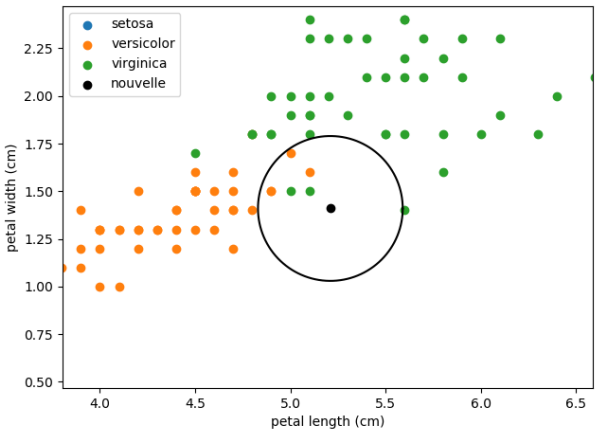



Imaginez désormais qu'en vous promenant dans les bois, vous trouviez un Iris. N'étant pas spécialiste,vous n'êtes pas capable de déterminer sa variété. Cependant, vous pouvez mesurer la longueur et la largeur de ses pétales.

Prédire à quelle classe appartient la nouvelle Iris constitue un problème de classification. L'algorithme des $k$ plus proches voisins propose une façon d'y répondre.




*nouvelle observation (Iris •)*




*zoom autour de la nouvelle Iris*

En cherchant ici les $k = 5$ plus proches voisins de la nouvelle Iris, il y a trois Iris Versicolor et 2 Iris Virginica : l'algorithme propose la classe Vercicolor pour notre nouvelle Iris.

# B Principe

L'algorithme des $k$ plus proches voisins (*kNN en anglais pour k-Nearest Neighbours*) est un algorithme de **classification supervisée**.

On dispose d'un jeu de données d'apprentissage $\mathcal{D} = [(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)]$ constitué de $n$ données appartenant chacune à une des $C$ classes.

Voici ce qu'on donne en entrée de l'algorithme :

- $x_i \in \mathbb{R}^p$ : un vecteur contenant $p$ valeurs caractéristiques appelées **descripteurs**. Dans notre exemple, il y 2 descripteurs, longueur et largeur des pétales.
- $y_i \in [|1, C|]$ : le numéro correspondant à la classe. Dans notre exemple, il y a $C = 3$ classes différentes, chacune correspondant à une variété.
- $x \in \mathbb{R}^p$ : la nouvelle observation (non étiquetée)
- $k \in \mathbb{N}^*$ : nombre de voisins à considérer

Afin de déterminer la classe de $x$, on cherche ses $k$ plus proches voisins parmi les $n$ observations et on assigne à $x$ **la classe majoritaire parmi celles de ses $k$ plus proches voisins**.

Présentation concrète de l'algorithme

1. Calculer la distance (euclidienne) $d_i = \|x - x_i\|$ de $x$ à chaque observation $x_i$
2. Trier par ordre croissant ces distances $d_i$ pour obtenir une liste triée : $d_{\sigma(1)} \leqslant d_{\sigma(2)} \leqslant ... \leqslant d_{\sigma(n)}$
3. Retenir l'ensemble $I = \{\sigma(1), ..., \sigma(k)\}$ des $k$ premiers indices
4. Déterminer la classe majoritaire parmi $(y_i)_{i \in I} = \left(y_{\sigma(1)}, ..., y_{\sigma(k)}\right)$
   (s'il n'y a pas d'éléments majoritaire, on convient d'un moyen arbitraire pour départager les ex-æquo)

# C Influence de la valeur de $k$

Le nombre $k$ de voisins à considérer est très important : il doit être judicieusement choisi car son influence sur la prédiction reste forte. A titre d'exemple, reprenons notre mise en situation avec la nouvelle Iris et appliquons l'algorithme des $k$ voisins en faisant varier $k$ pour prédire sa variété :

| Nombre de voisins $k$ | 1 | 3 | 7 | 10 | 20 | 50 | 100 |
|---|---|---|---|---|---|---|---|
| Variété prédite | virginica | virginica | versicolor | versicolor | virginica | versicolor | versicolor |

La prédiction peut changer selon la valeur de $k$ et il est incorrect de penser que plus la valeur de $k$ augmente, plus la prédiction est bonne.

**Choisir $k$ de façon optimale est difficile en général**. Voici quelques commentaires en guise d'éclairage :

- Si $k$ est trop petit : la réponse de l'algorithme est plus sensible au bruit des données d'entrainement. Imaginons par exemple qu'un unique iris setosa possède de très grands pétales et prenons $k = 1$. Le modèle prendra en compte cette Iris et risque de mal classer un iris trop proche de cette valeur aberrante alors que ce dernier n'est pas du tout représentatif de sa variété. Le modèle se généralise mal sur de nouvelles données : on parle de **sur-apprentissage**.
- Si $k$ est trop grand : En prenant $k = n$ (i.e. tout le jeu d'entrainement), toute nouvelle observation se verra assigner la classe majoritaire du jeu d'entrainement. La réponse de l'algorithme est toujours la même ! le modèle n'a rien appris : on parle de **sous-apprentissage**.

# D Évaluation de l'algorithme : matrice de confusion

Un algorithme d'apprentissage ne produisant pas toujours des résultats parfaits, il est utile de pouvoir l'évaluer.

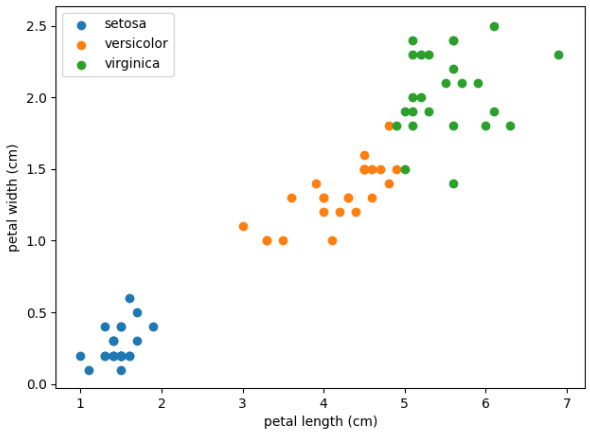
Dans cette optique, on utilise généralement un *jeu test*, différent du *jeu d'entrainement*, sur lequel les vraies étiquettes sont connues. On peut alors évaluer les performances de l'algorithme grâce à la **matrice de confusion** :

**Définition**

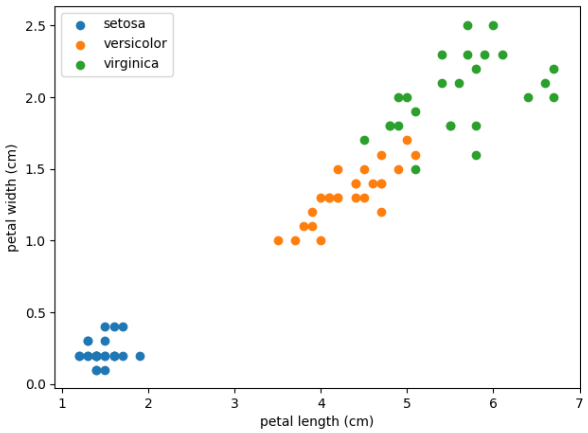
La matrice de confusion d'un résultat de classification sur un *jeu test* est une matrice carrée d'ordre $C$ (nombre de classes) dont le coefficient $(i, j)$ compte le nombre d'observations pour laquelle la vraie classe est $i$ et la classe estimée est $j$

En Pratique :

Reprenons notre exemple des Iris et cherchons à évaluer son efficacité. Il nous faut constituer un *jeu test* indépendant du *jeu d'apprentissage*. Pour cela, on découpe aléatoirement la base initiale composée de 50 échantillons de chaque classe, en deux bases de 25 échantillons par classe (ainsi, 75 échantillons composent chacun des deux jeux).




*jeu d'apprentissage*



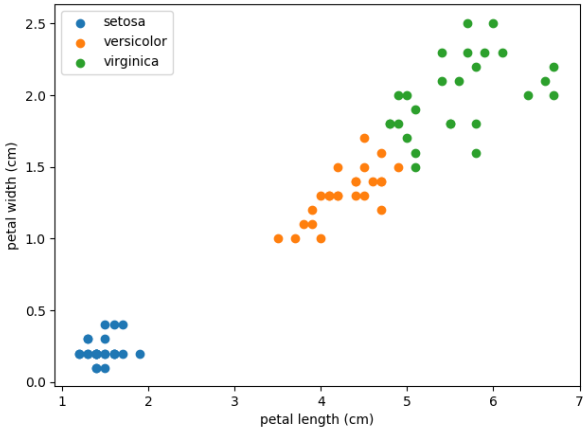

*jeu test*

Entrainé avec le *jeu d'apprentissage*, on applique alors l'algorithme $kNN$ (avec $k = 3$) au *jeu test* et on compare les prédictions obtenues avec les vraies étiquettes du *jeu test*. Voici les résultats :

| | setosa | versicolor | virginica |
|---|---|---|---|
| setosa | 25 | 0 | 0 |
| versicolor | 0 | 23 | 2 |
| virginica | 0 | 1 | 24 |

*matrice de confusion*

Interprétation : parmi les 25 versicolor testées, l'algorithme en a prédit 23 corrects et s'est trompé 2 fois en les étiquetant virginica.




*prédiction du jeu test par l'algorithme $kNN$*

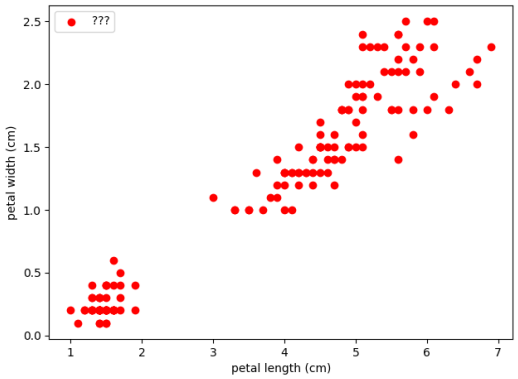
**L'efficacité de l'algorithme $kNN$ s'obtient en examinant la dispersion des valeurs de la matrice de confusion située en dehors de sa diagonale.**

*remarque : typiquement, un jeu de données d'entrées est plutôt séparé dans des proportions de 80% pour le jeu d'apprentissage et 20% pour le jeu test.*

# III Algorithme des $k$ moyennes

## A Mise en situation

Reprenons le jeu de données *Iris*, mais cette fois on se met à la place du premier botaniste qui a découvert les iris : les données des pétales sont collectées et c'est à lui que revient la tâche de les classer en variétés.
Nous disposons donc de la représentation ci-contre :




A partir de ces caractéristiques, on souhaite séparer nos iris en plusieurs nouvelles catégories. Graphiquement, on a l'impression que deux "groupes" semble se distinguer. Cette impression est à l'origine des questionnements suivants :

- Comment calculer automatiquement une décomposition/partition d'un ensemble en "groupes" distincts ?
- Quel est le nombre optimal de "groupes" à extraire ?

Tenter de répartir "correctement" des données non étiquetées dans des groupes distincts constitue un problème de **regroupement/partitionnement de données**. L'algorithme des $k$ moyennes propose une façon de répondre à la première question.

## B Principe

L'algorithme des $k$ moyennes *(k-means en anglais)* est un algorithme de **regroupement non supervisé** et permet de répartir/partitionner des données en $k$ clusters (groupes).

Voici ce qu'on donne en entrée de l'algorithme :

- un jeu de données $\mathcal{D} = [x_1, x_2, ..., x_n]$ constitué de $n$ données non étiquetées où les $x_i \in \mathbb{R}^p$ ($p$ descripteurs)
- $k \in \mathbb{N}^*$ : nombre de clusters souhaités.

Étant donnés $k$ groupes $G_1, G_2, ..., G_k$ formant une partition de $\mathcal{D}$, on va noter $\overline{G_1}, \overline{G_2}, ..., \overline{G_k}$ leurs barycentres respectifs. L'idée de l'algorithme des $k$ moyennes va être de faire évoluer la composition des groupes $G_1, ..., G_k$ et donc de leurs barycentres jusqu'à obtenir un résultat "satisfaisant".

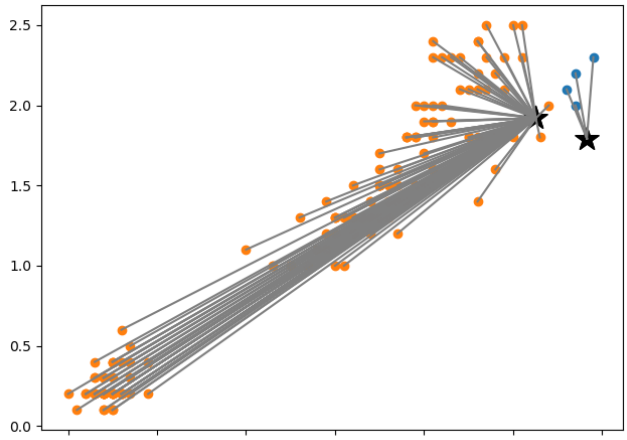
Présentation concrète de l'algorithme (phase d'apprentissage) :

1. On commence par initialiser $k$ points dans $\mathbb{R}^p$, généralement de manière aléatoire. Ces points seront les valeurs initiales des barycentres $\overline{G_1}, \overline{G_2}, ..., \overline{G_k}$.
2. On affecte chaque observation $x_i$ de $\mathcal{D}$ au barycentre $\overline{G_j}$ le plus proche. Les données d'entrainement sont ainsi réparties en $k$ nouveaux groupes.
3. On recalcule les barycentres (ou moyennes) de chacun de ces groupes et on met à jour les valeurs de $\overline{G_1}, \overline{G_2}, ..., \overline{G_k}$.
4. On réitère les étapes 2. et 3. jusqu'à ce que les postions des barycentres ne changent plus : on parle de *convergence* (même s'il s'agit plus précisément de stationnarité)

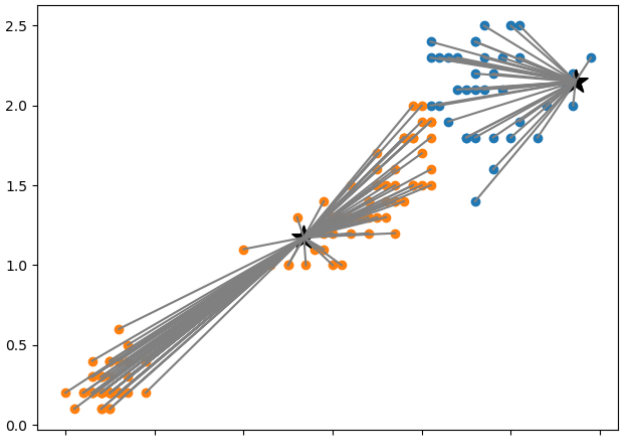
Ainsi, **lors de la phase d'entrainement, on tente soi-même d'étiqueter chaque donnée** $x_i$ : on lui donne comme étiquette le numéro du groupe dans lequel elle figure une fois obtenue la convergence de l'algorithme des $k$-moyennes.

Ensuite en cas de réception d'une nouvelle donnée $x$ (phase d'inférence), le modèle pourra lui attribuer une étiquette à l'aide de ce qu'il aura appris.

Illustration pas à pas



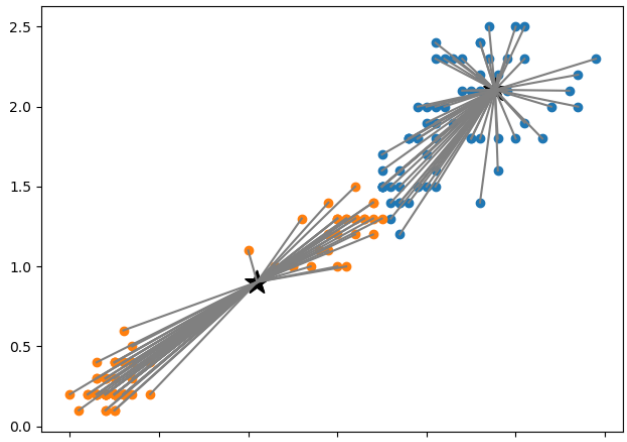

*phase 1*



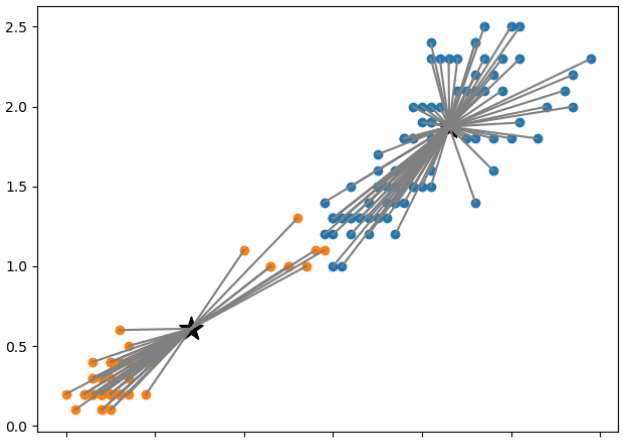

*phase 2*

Phase 1 : on initialise aléatoirement nos 2 barycentres (repérés par ★) et on relie chaque observation au barycentre le plus proche. On obtient 2 groupes, qui ont chacun un barycentre : on actualise alors nos 2 barycentres (ce sont les ★ représentés en phase 2). Puis on continue : on relie chaque observation au (nouveau) barycentre le plus proche.

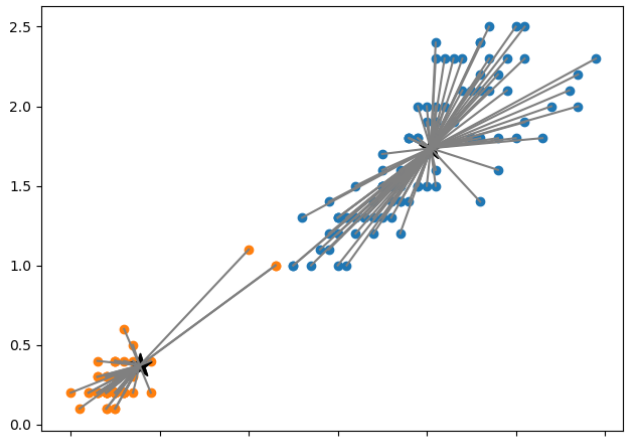
.....etc.....on itère le processus :



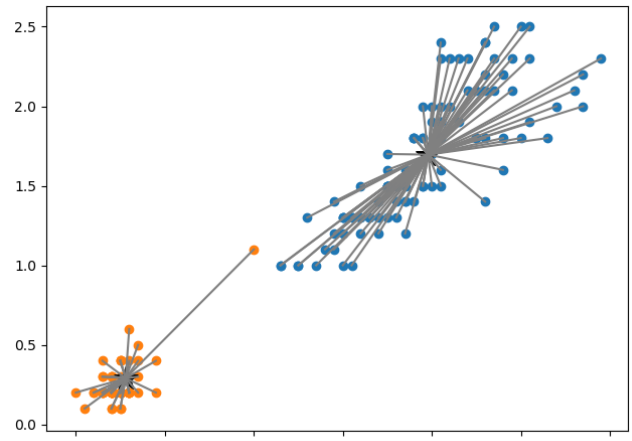

*phase 3*




*phase 4*




*phase 5*




*phase 6*

Phase 6 : le calcul des nouveaux barycentres ne modifie plus les positions ★, l'algorithme s'arrête et propose 2 groupes de classification (orange et bleu)

Désormais comparons avec les "vraies" étiquettes : on constate que l'algorithme a plutôt bien réussi à séparer les Iris setosa des autres variétés, il a juste commis une erreur.

Description théorique

On cherche en fait à minimiser, dans chaque groupe $G_j$, la distance (euclidienne) de ses points au barycentre $\overline{G_j}$. Cette dernière est donnée par :

$$V(G_1,...,G_k) = \sum_{j=1}^{k} \underbrace{\sum_{i \in G_j} \|x_i - \overline{G_j}\|^2}_{\text{variance intra-classe}} \qquad \text{où} \qquad \overline{G_j} = \frac{1}{Card(G_j)} \sum_{i \in G_j} x_i$$

L'algorithme des $k$ moyennes à pour objectif de trouver une bonne partition $G_1, G_2, ..., G_k$ du jeu de données $\mathcal{D}$ qui minimise $V(G_1,...,G_k)$.

On admet que l'algorithme se termine (en pratique, c'est très rapide) et qu'à chaque itération, la variance intra-classe diminue. Cependant, rien ne nous garantit que la solution obtenue soit la meilleure : **le minimum atteint est en fait local** et pas nécessairement global.

De plus, la solution obtenue dépend également de l'initialisation des barycentres : l'algorithme peut donc fournir des solutions différentes lorsqu'il est utilisé plusieurs fois.

## C Influence de la valeur de $k$

Le principal inconvénient de cet algorithme de classification non supervisée réside dans le choix du paramètre $k$ : comment savoir si le partitionnement obtenu représente bien la structure cachée des données ?

Là aussi, **choisir $k$ de façon optimale est difficile en général**. Voici quelques commentaires en guise d'éclairage :

- Si $k$ est trop petit :
  pour $k = 1$ par exemple, on forme un seul gros cluster, qui contiendra finalement toutes les données d'entrainement. On n'apprend rien sur la structure des données, on est en **sous-apprentissage**.
- Si $k$ est trop grand :
  pour $k = n$ groupes par exemple, cela revient à former des clusters formés d'un seul élément. Il n'y a donc plus de regroupement ! On est en **sur-apprentissage**.

# IV Épilogue

**Piège** :
même si les deux algorithmes présentés dans le chapitre utilisent le même nom pour la variable $k$, il ne s'agit pas du tout du même paramètre ! ! D'un côté, $k$ désigne un nombre de voisins. De l'autre, il désigne un nombre de groupes.

Lors d'un concours, on ne peut pas nous demander de savoir coder en détail de façon autonome l'algorithme $kNN$ et l'algorithme $k-means$. Par contre, on doit être au point sur leurs fonctionnements, leurs cadres, leurs objectifs .... et par conséquent savoir suivre une démarche qui utiliserait ces algorithmes. C'est ce que nous allons travailler en TD/TP.

Voici les termes exacts du programme au sujet de l'apprentissage automatique :

*"""Cette partie permet notamment de revisiter les notions de programmation et de représentation de données par un graphe, qui sont vues en première année, en les appliquant à des enjeux contemporains.*

*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."""*