Algorithmes pour l'apprentissage automatique

1 Introduction

L'apprentissage automatique, en anglais *machine learning*, est un domaine de l'intelligence artificielle ayant pour but de permettre à une machine d'« apprendre » à partir de données. Plus précisément, il s'agit de lui permettre d'améliorer son fonctionnement pour résoudre un problème donné sans le programmer explicitement pour résoudre ce problème.

On s'intéressera principalement à des problèmes de classifications : étant donnés des objets, il s'agit d'attribuer à chaque objet une catégorie, une classe. Par exemple, avec des mails, décider s'il s'agit d'un spam ou non.

Il existe également des problèmes de régression : il s'agit de prédire la valeur d'une caractéristique à partir de données sur des objets. Par exemple, prédire la valeur d'un logement en connaissant sa localisation, sa surface, la présence d'un garage à vélos, etc.

1.1 Différentes méthodes d'apprentissage

Généralement l'apprentissage automatique comporte deux phases : la phase d'apprentissage (ou d'entraînement) de l'algorithme et, une fois que celui-ci semble satisfaisant, celle de son utilisation pour faire des prédictions.

C'est la première phase qui va nous intéresser et on va étudier spécialement deux formes[1] d'apprentissage : celui supervisé et celui non supervisé.

Apprentissage supervisé

Pour ce type d'apprentissage, les données fournies à la machine sont *étiquetées*, c'est-à-dire que le classement des données a été fait en amont et est communiqué à la machine. Par exemple, un ensemble de photos où pour chacune est indiqué s'il s'agit d'un chat ou d'un chien.

L'algorithme est alors entraîné sur ces données étiquetées. Une fois qu'il semble satisfaisant, il est utilisé sur de nouvelles photos afin de fournir une prédiction pour chacune d'elle : s'agit-il d'un chat ou d'un chien ?

Comme exemple d'une telle méthode d'apprentissage, on s'intéressera à l'algorithme des k plus proches voisins, en anglais *k-nearest neighbors*, souvent abrégé en KNN.

Apprentissage non supervisé

Pour ce type d'apprentissage, les données ne sont pas étiquetées et le but de l'algorithme est de regrouper les données présentant certaines similarités pour former des classes, des groupes pertinents. En anglais on parle de *clustering*. Par exemple, sur une photo aérienne d'une région, découper des zones homogènes qui pourront être interprétées ensuite : forêts, champs, villes, etc.

Comme exemple d'une telle méthode d'apprentissage, on s'intéressera à l'algorithme des k-moyennes, en anglais *k-means*.

1.2 Quelques dates clés (culture)

1936 Alan Turing imagine une « machine universelle » qui applique des règles données. Cette *machine de Turing* est l'ancêtre théorique des ordinateurs qui apparaîtront quelques années plus tard.

1959 Arthur Samuel conçoit un programme jouant au jeu de dames et s'améliorant lui-même en jouant. La même année il popularise le terme de « machine learning ».

1997 L'ordinateur *Deep Blue* bat le champion du monde d'échecs Gary Kasparov.

2016 L'ordinateur AlphaGo bat Lee Seedol, l'un des meilleurs joueurs mondiaux de Go.

1. Il en existe d'autres, par exemple l'apprentissage par renforcement qui a été utilisé sur le jeu de Go.

2 Algorithme des k plus proches voisins

L'algorithme des k plus proches voisins, en anglais *k-nearest neighbors* souvent abrégé en KNN, a été introduit par les statisticiens Evelyn Fix et Joseph Hodges en 1951.

2.1 Principe de l'algorithme

Comme expliqué précédemment, on suppose que l'on dispose d'un ensemble d'objets dont on connaît toutes les caractéristiques, dont sa classe (ce sont les données étiquetées). On considère alors un nouvel objet `Obj` dont on connaît toutes les caractéristiques sauf sa classe : on souhaite prédire celle-ci à partir des caractéristiques et ce à l'aide des données étiquetées.

L'algorithme des k plus proches voisins repose sur un principe relativement simple : pour savoir à quelle classe appartient un objet `Obj`, on détermine ses k (à choisir) « voisins » les plus proches (dans un sens à préciser) et on prédit alors que `Obj` appartient à la classe majoritaire parmi ces k voisins.

2.1.1 La notion de voisin

Les données à propos des objets sont des données numériques diverses suivant ce que l'on étudie (longueur, prix, couleur d'un pixel, etc.). On calcule alors la distance entre deux objets comme s'il s'agissait de points dans l'espace (de dimension le nombre de données par objet) : on utilise la distance euclidienne.

Par exemple pour des objets ayant trois caractéristiques x, y et z, la distance entre `Obj1`(x_1, y_1, z_1) et `Obj2`(x_2, y_2, z_2) est donnée par

$$\text{dist}(\texttt{Obj1}, \texttt{Obj2}) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2}.$$

En pratique, on utilisera plutôt le carré de cette distance pour éviter le calcul de la racine carré puisqu'il s'agit juste de comparer des distances entre elles, pas d'interpréter les valeurs obtenues.

⇝ TD5 exercice 1

2.1.2 Déterminer les k voisins les plus proches

Une fois les distances de `Obj` à tous les autres objets calculées, pour déterminer les k voisins les plus proches de `Obj`, on peut trier tous les objets par distance croissante à `Obj` et retenir les k premiers (ou par distance décroissante et garder les k derniers).

⇝ TD5 exercice 2

2.1.3 Déterminer la classe majoritaire

Maintenant que les k voisins les plus proches ont été déterminés, il suffit de compter les classes représentées parmi eux. S'il y a une classe majoritaire stricte, c'est elle que l'on prédit pour `Obj`. S'il y a des égalités, on peut soit choisir une classe au hasard parmi ces ex aequo, soit choisir celle du voisin le plus proche parmi ces ex aequo.

⇝ TD5 exercice 3

2.1.4 Un exemple de mise en pratique

Supposons que l'on considère des objets dépendants de deux paramètres (disons x et y, peu importe ce qu'ils représentent) dont on cherche une caractéristique de couleur particulière pour laquelle il y a deux possibilités : noir ou blanc.

On dispose d' un certain nombre d'objets dont on connaît déjà la couleur comme illustré sur la figure 1. On a alors un nouvel objet dont on connaît les caractéristiques x et y mais pas la couleur : on le symbolise par une croix, cf figure 2. On souhaite prédire sa couleur à l'aide de l'algorithme des k plus proches voisins avec $k = 3$ (choix arbitraire).

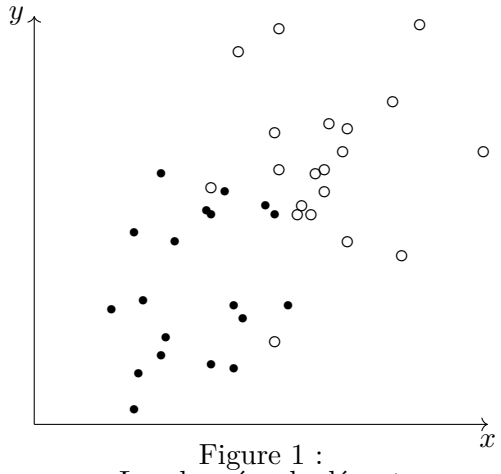

Figure 1 :
Les données de départ

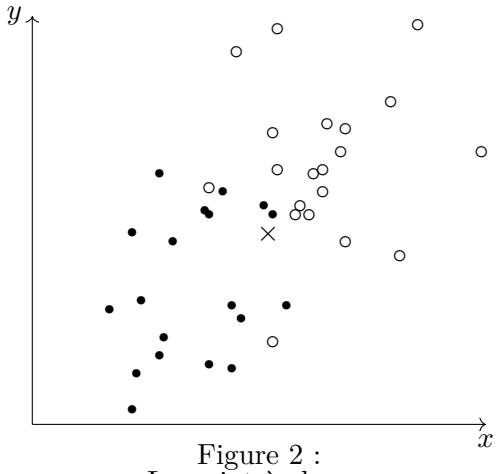

Figure 2 :
Le point à classer

Comme on avait choisi $k = 3$, on détermine alors les trois voisins les plus proches de la croix, comme illustré figure 3. On regarde alors la classe (ici la couleur) majoritaire parmi ces trois proches voisins et on prédit alors que la croix est de la couleur majoritaire, cf figure 4.

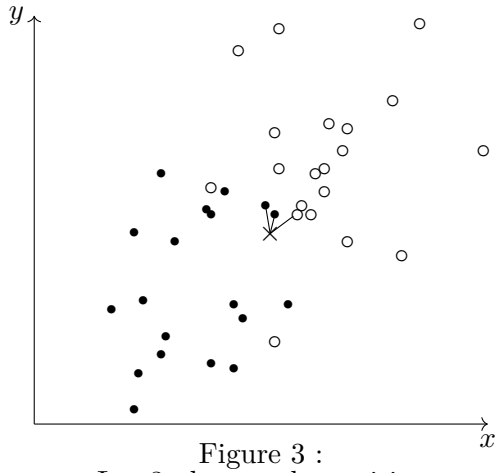

Figure 3 :
Les 3 plus proches voisins

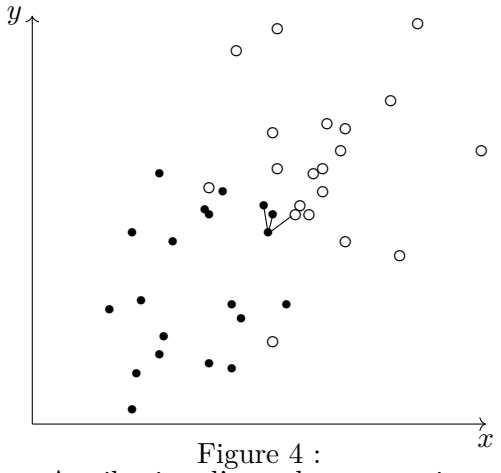

Figure 4 :
Attribution d'une classe au point

Remarquons que la valeur choisie pour k peut avoir un impact sur la classe prédite de certains points. Par exemple, si on avait choisi $k = 5$ au lieu de $k = 3$, la prédiction de couleur pour le point qui nous intéressait aurait été différente comme le montrent les figures 3b et 4b.

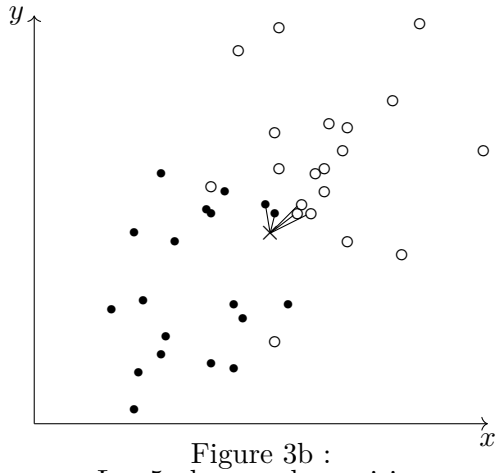

Figure 3b :
Les 5 plus proches voisins

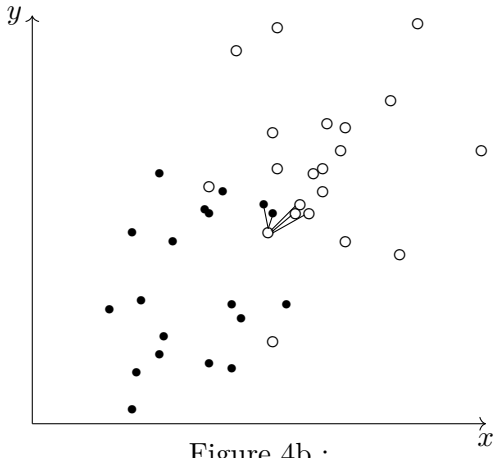

Figure 4b :
Attribution d'une classe au point

2.1.5 Résumé : les étapes de l'algorithme des k plus proches voisins

Au départ on dispose de points dans $\mathbb{R}^n$ (les données, leur classe est connue) et d'un point X dont on veut déterminer la classe. On choisit une valeur de k.

1. On calcule la distance de X à chaque point des données.
2. On détermine les k points des données les plus proches de X, par exemple en les triant selon leur distance à X.
3. On détermine la classe majoritaire parmi ces k points et on renvoie ce résultat.

2.2 Analyse des résultats et efficacité de l'algorithme

Imaginons que l'on souhaite développer un test pour prédire si une personne est atteinte ou non par une maladie. Avant de l'utiliser pour faire des prédictions, on doit l'évaluer : on doit au moins savoir quel est son niveau de bonne prédiction, mais aussi éventuellement savoir dans quels cas les erreurs sont les plus fréquentes.

2.2.1 Données d'entraînement et données de test

Afin de savoir si notre modèle est pertinent et va fournir des prédictions correctes, on doit le tester sur des données dont on connaît déjà le résultat.

Pour cela, on va réserver une partie, en général environ 20 %, des données étiquetées pour ces tests, on parlera de *données de test*. Il faut veiller à ce que ces données de test soient représentatives des données dont on dispose, en général on les prend donc au hasard parmi toutes les données étiquetées. Les 80 % restants sont les *données d'entraînement* : celles-ci vont permettre d'effectuer les prévisions.

Plus précisément, on va appliquer l'algorithme des k plus proches voisins à chacun des objets des données de test afin de prédire sa classe. On va chercher ses voisins les plus proches parmi les données d'entraînement et en déduire une prédiction de classe. On comparera alors ce résultat avec la classe connue pour cet objet afin de voir si la prédiction est bonne ou non. Une fois cela fait avec chaque objet des données de test, on dénombre les bonnes et mauvaises prédictions et on obtient le taux d'erreurs du modèle (ou sa précision).

⇝ TD5 exercice 4

Ce taux d'erreur peut par exemple permettre de choisir la valeur de k optimale. En effet, il suffit de répéter le processus de test avec différentes valeurs de k et constater que le taux d'erreurs est minimal pour une (ou plusieurs) valeur de k.

2.2.2 Matrice de confusion

Il est souvent utile d'être plus précis quant aux réussites et erreurs de prédiction. Pour reprendre l'exemple du test de détection d'une maladie, on peut se demander si nos erreurs viennent de malades non détectés par le test (on parle de « faux négatifs ») ou de personnes non malades alors que le test les donnait pour malades (on parle de « faux positifs »), les conséquences de ces deux types d'erreurs étant différentes. Pour cela, on peut présenter les résultats sous forme d'un tableau à double entrée :

	Test positif	Test négatif
Malade		
Pas malade		

Plus généralement mais de manière analogue, on peut présenter les résultats obtenus lors du classement de nos données de tests sous forme de tableau comportant autant de lignes et de colonnes qu'il existe de classes possibles :

— chaque ligne correspond à une classe réelle,

— chaque colonne correspond à une classe prédite.

Dans la case de la ligne i et de la colonne j, on inscrit le nombre d'objets de classe réelle i qui ont été estimés comme appartenant à la classe j. Ce tableau est appelé *matrice de confusion*.

⇝ TD5 exercice 5

2.2.3 Interprétation des résultats

Sur la diagonale de la matrice de confusion, on trouve les bonnes prédictions : la classe estimée correspond à la classe réelle. En dehors de la diagonale, on retrouve toutes les erreurs de prédiction et on peut alors chercher si certains cas posent particulièrement problème comme une classe pour laquelle les prédictions sont moins bonnes que pour les autres.

⇝ TD5 exercice 6

Dans le cas particulier de l'exemple du test pour la maladie, on définit couramment :

- la sensibilité : c'est le taux de vrais positifs, *i.e.* le nombre de prédictions positives sur le nombre de malades ;
- la spécificité : c'est le taux de vrais négatifs, *i.e.* le nombre de prédictions négatives sur le nombre de non-malades.

Ci-contre un extrait d'une notice d'autotest pour le COVID19 (le rôle des lignes et colonnes est échangé par rapport à la matrice de confusion).

Écouvillon nasal	RT-PCR		Total
	Positif	Négatif	
Positif	168	2	170
Négatif	5	262	267
Total	173	264	437
	Sensibilité	Spécificité	Précision globale
	97,1%	99,2%	98,4%
	95% CI: [93,4%-99,1%]	95% CI: [97,3%-99,9%]	95% CI: [96,7%-99,4%]

2.2.4 Résumé : savoir analyser les résultats

Pour connaître la fiabilité des prédictions de l'algorithme, on le teste sur des données dont la classe est connue et on la compare avec la prédiction obtenue.

Taux de réussite : c'est le pourcentage de bonnes prédictions. Il permet notamment de déterminer une valeur de k optimale : celle qui donne le meilleur taux de réussite.

Matrice de confusion : le coefficient $M_{i,j}$ de cette matrice correspond au nombre de fois où l'algorithme a prédit la valeur j alors que la valeur réelle est i. On souhaite donc que cette matrice soit le plus proche possible d'une matrice diagonale.

3 Algorithme des k-moyennes

L'algorithme des k-moyennes, en anglais *k-means*, a été introduit par Stuart Lloyd en 1957 et repose sur des idées du mathématicien Hugo Steinhaus.

3.1 Principe de l'algorithme

Comme expliqué précédemment, on dispose de données et on souhaite les regrouper en k groupes (des *clusters*), chaque groupe étant composé de données présentant des similarités entre elles. Ce nombre de groupes est choisi au départ : il peut correspondre au problème posé, par exemple $k = 10$ si on souhaite reconnaître des chiffres, ou être un simple choix arbitraire permettant juste de constituer différents groupes afin de faciliter le traitement des données.

Il s'agit donc de trouver des ressemblances parmi les données pour constituer ces groupes, on parle de *clustering*. Ces groupes seront représentés par une donnée centrale, les autres éléments de son groupe étant « proches » de ce centre.

Pour ce faire, l'algorithme des k-moyennes consiste à :

1. choisir k centres, généralement de manière aléatoire ;
2. affecter chaque donnée au centre le plus proche : on a ainsi créé k groupes ;
3. mettre à jour le centre de chaque groupe, d'où le nom de l'algorithme : on calcule ici k moyennes ;
4. répéter les deux étapes précédentes jusqu'à ce que les centres ne bougent plus.

3.2 Formulation mathématique

Supposons que l'on représente nos données comme des points dans $\mathbb{R}^m$ (les coordonnées correspondent à des caractéristiques de ces données, par exemple la couleur de chaque pixel d'une image). On dispose alors au départ de n points $p_1, \dots, p_n$. En notant d la fonction distance [2], on cherche à trouver k centres $c_1, \dots, c_k$ de groupes $G_1, \dots, G_k$ tels que la quantité

$$V = \sum_{i=1}^{k} \sum_{p_j \in G_i} \Big(d(p_j, c_i) \Big)^2$$

soit minimale. Cette quantité est appelée *inertie* ou *variance intra-classe* : elle est d'autant plus faible que chaque groupe est homogène, *i.e.* concentré proche de son centre.

Les deux étapes de l'algorithme que l'on répète peuvent alors se voir comme des minimisations de cette variance :

1. l'étape d'affectation au centre le plus proche minimise V pour des centres fixés ;
2. l'étape de mise à jour des centres consiste quant à elle à minimiser V pour des groupes fixés.

Cette formulation permet de démontrer que l'algorithme converge en un nombre fini d'étapes [3].

3.3 L'algorithme en pratique

On suppose que l'on dispose d'une liste de données `datas`, chaque donnée étant un tuple représentant ses coordonnées.

3.3.1 Initialisation des centres

L'algorithme des k-moyennes débute par un choix de k centres. Une manière de faire consiste à choisir aléatoirement k points parmi les données dont on dispose, *i.e.* k éléments distincts de la liste `datas`.

⇝ TD6 exercice 1

2. cf §2.1.1. Comme pour l'algorithme des k plus proches voisins, en pratique on considèrera le carré de la distance.
3. La démonstration précise est hors programme mais repose sur deux idées : d'une part il n'y a qu'un nombre fini de regroupements possibles et d'autre part à chaque étape V diminue.

3.3.2 Répartition dans les groupes

Les centres étant fixés, afin de répartir les données en k groupes, on affecte chacune au centre le plus proche. Pour cela, il suffit de calculer la distance entre le point considéré et chacun des k centres et de prendre le minimum (ou l'un des minimums en cas d'égalité).

⇝ TD6 exercice 2

3.3.3 Mise à jour des centres

Une fois tous les éléments de `datas` répartis dans les k groupes, on met à jour les centres en calculant l'*isobarycentre* de chaque groupe. Ce point est défini comme ayant pour coordonnées la moyenne des coordonnées de tous les points du groupe.

⇝ TD6 exercice 3

3.3.4 Fin de l'algorithme

Lors de l'étape précédente, les centres ont été modifiés donc il se peut que des points qui faisait partie d'un groupe de centre c_i soient maintenant plus proches d'un autre centre $c_j \neq c_i$. On répète donc l'étape de répartition dans les groupes avec les nouveaux centres obtenus à l'étape précédente. Une fois cela fait, on met à nouveau à jour les centres de chaque groupe puis on procède à une nouvelle répartition, etc.

On répète ainsi les étapes de répartition dans les groupes et de mise à jour des centres jusqu'à ce qu'il n'y ait plus de modifications d'une étape sur l'autre. En général, on fixe également un nombre maximum d'itérations pour éviter un cas de convergence trop longue à attendre.

⇝ TD6 exercice 4

3.4 Un exemple visuel avec $k = 3$

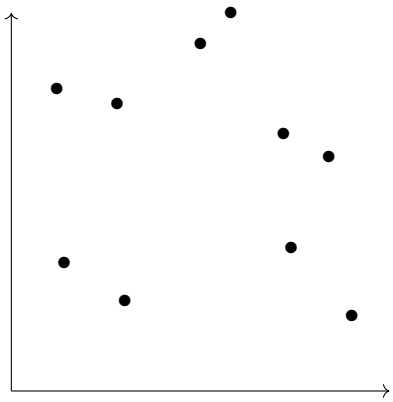

Figure 1 :
Les données de départ

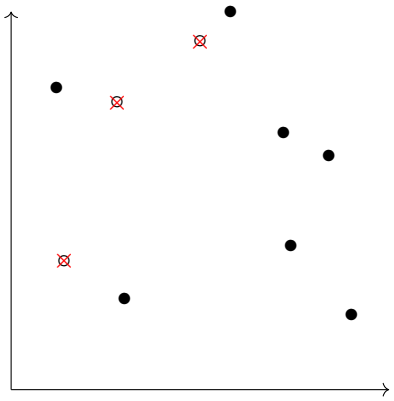

Figure 2 :
Choix aléatoire de 3 centres

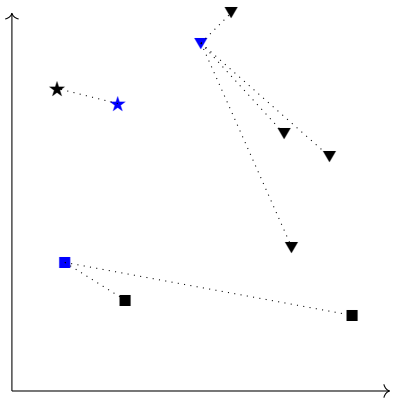

Figure 3 :
Répartition en groupes

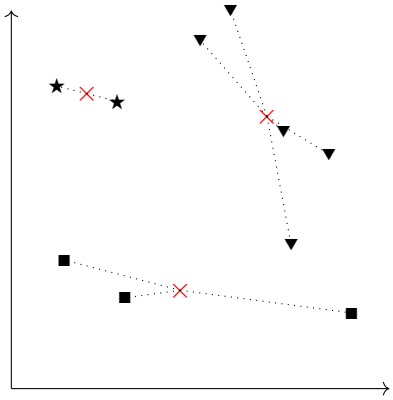

Figure 4 :
Nouveaux centres

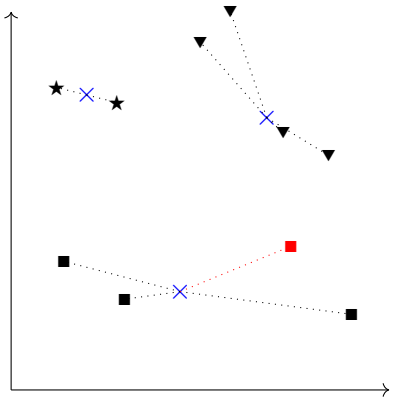

Figure 5 :
Nouvelle répartition

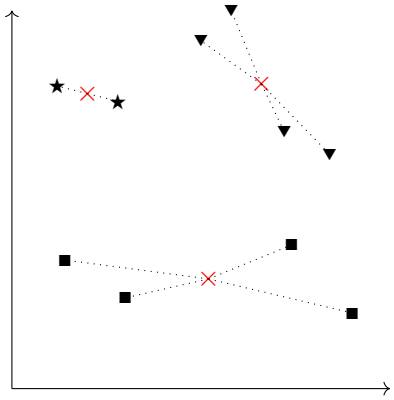

Figure 6 :
Nouveaux centres

Ensuite, il n'y a plus de modification : chaque point a pour centre le plus proche celui de son groupe actuel, autrement dit l'algorithme a convergé et on a obtenu une répartition en $k = 3$ groupes de données relativement proches.

3.5 Influence de l'initialisation

Le choix des centres initiaux peut avoir une influence forte sur les résultats obtenus. En effet, il se peut que l'algorithme converge vers un minimum local non global pour certains choix de centres de départ. On peut facilement illustrer cette influence de l'initialisation en reprenant l'exemple précédent mais avec deux initialisations différentes des centres :

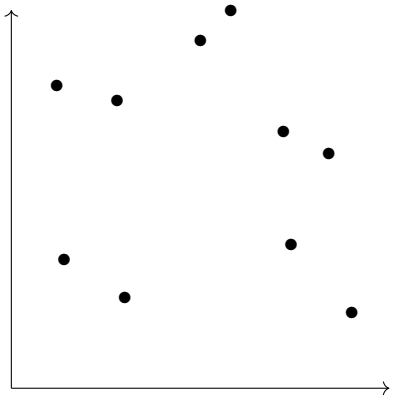

Figure 1a :
Les données de départ

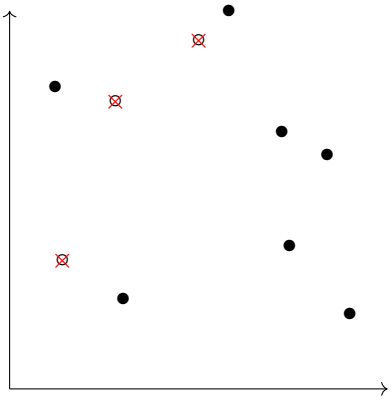

Figure 2a :
Choix aléatoires de 3 centres

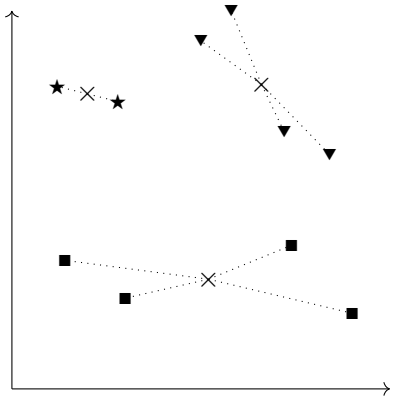

Figure 3a :
Résultat final

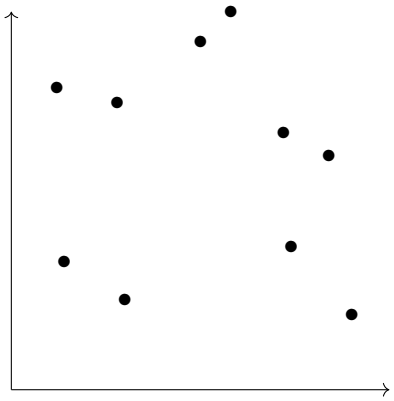

Figure 1b :
Mêmes données

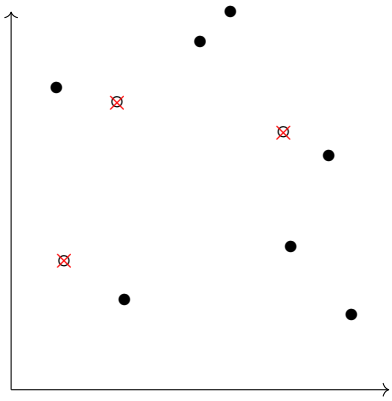

Figure 2b :
Autres centres initiaux

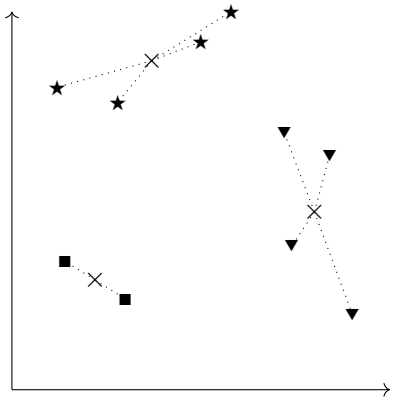

Figure 3b :
Autre résultat final

En pratique, pour éviter cela, on exécute plusieurs fois l'algorithme pour les mêmes données et la même valeur de k mais avec des initialisations de centres différentes et on garde le meilleur résultat obtenu (au sens de la variance minimale).

⇝ TD6 exercice 5