CHAPITRE 3

Algorithmes d'apprentissage

Jérôme VON BUHREN
http://vonbuhren.free.fr

CPGE - Filière Scientifique - 2e année

L'**apprentissage automatique** (machine learning en anglais) est une branche de l'intelligence artificielle et de l'informatique qui vise à apprendre aux machines à résoudre certaines tâches en tirant des enseignements à partir de données fournies et en s'améliorant progressivement avec l'expérience. L'idée directrice est de tenter d'imiter le processus d'apprentissage des êtres humains. On peut par exemple apprendre à une machine à détecter la présence ou non d'un objet donné dans une image.

On distingue deux types d'apprentissage.

- L'**apprentissage supervisé** consiste à fournir à une machine de nombreuses données étiquetées par leur catégorie (phase d'apprentissage) ; ces dernières permettant à la machine de prédire la catégorie d'une nouvelle donnée sans étiquette.
- L'**apprentissage non supervisé** consiste à fournir à une machine des données sans étiquette, puis la laisser déterminer ses propres catégories en regroupant les données proches entre elles.

Dans ce chapitre nous allons présenter et étudier deux algorithmes d'apprentissage : l'algorithme des k plus proches voisins (supervisé) et l'algorithme des k-moyennes (non supervisé).

On considère une collection d'objets $x_1, \dots, x_n$ appartenant à un ensemble $\mathscr{O}$. On suppose qu'à chaque objet x_i est associé une étiquette y_i appartenant à un ensemble $\mathscr{E}$. Dans la suite, on note l'ensemble des données

$$\mathscr{D} = \{(x_1, y_1), \dots, (x_n, y_n)\}.$$

L'objectif est d'être capable de prédire automatiquement l'étiquette $y \in \mathscr{E}$ de tout nouvel objet $x \in \mathscr{O}$ « du mieux possible » en s'appuyant sur le jeu de données $\mathscr{D}$. Il s'agit d'un problème de classification.

Exemple 1

Un ordinateur dispose dans un dossier « chats » de nombreuses photos de chats et dans un dossier « chiens » de nombreuses photos de chiens. En utilisant ces données, on souhaite que l'ordinateur puisse ranger automatiquement une nouvelle photo de chat ou de chien dans le bon dossier.

On fixe un entier $k \in [\![1, n]\!]$. Rappelons que la distance euclidienne entre deux points $x, x' \in \mathbb{R}^d$ est définie par

$$\delta\left(x, x'\right) = \sqrt{\sum_{i=1}^{d} \left(x'_i - x_i\right)^2}.$$

L'algorithme des k plus proches voisins nécessite la construction d'une fonction $f : \mathscr{O} \to \mathbb{R}^d$ permettant d'associer à chaque objet des données numériques et vérifiant la propriété suivante : pour tout couples d'objets $(x, x') \in \mathscr{O}^2$, si les vecteurs $f(x)$ et $f(x')$ sont proches pour la distance euclidienne, alors les étiquettes y et y' ont tendance à être identiques. La construction d'une telle fonction f est un problème difficile en général : il ne sera pas abordé dans ce cours. On suppose dans la suite que $\mathscr{O} = \mathbb{R}^d$.

Algorithme des k plus proche voisin

Entrées : les données $\mathscr{D}$, l'entier k, un nouvel objet x.

1) On détermine les k éléments $x_{i_1}, \dots, x_{i_k}$ les plus proches pour la distance euclidienne de x parmi $\{x_1, \dots, x_n\}$.
2) L'algorithme renvoie l'étiquette majoritaire parmi $y_{i_1}, \dots, y_{i_d}$.

Remarques 1

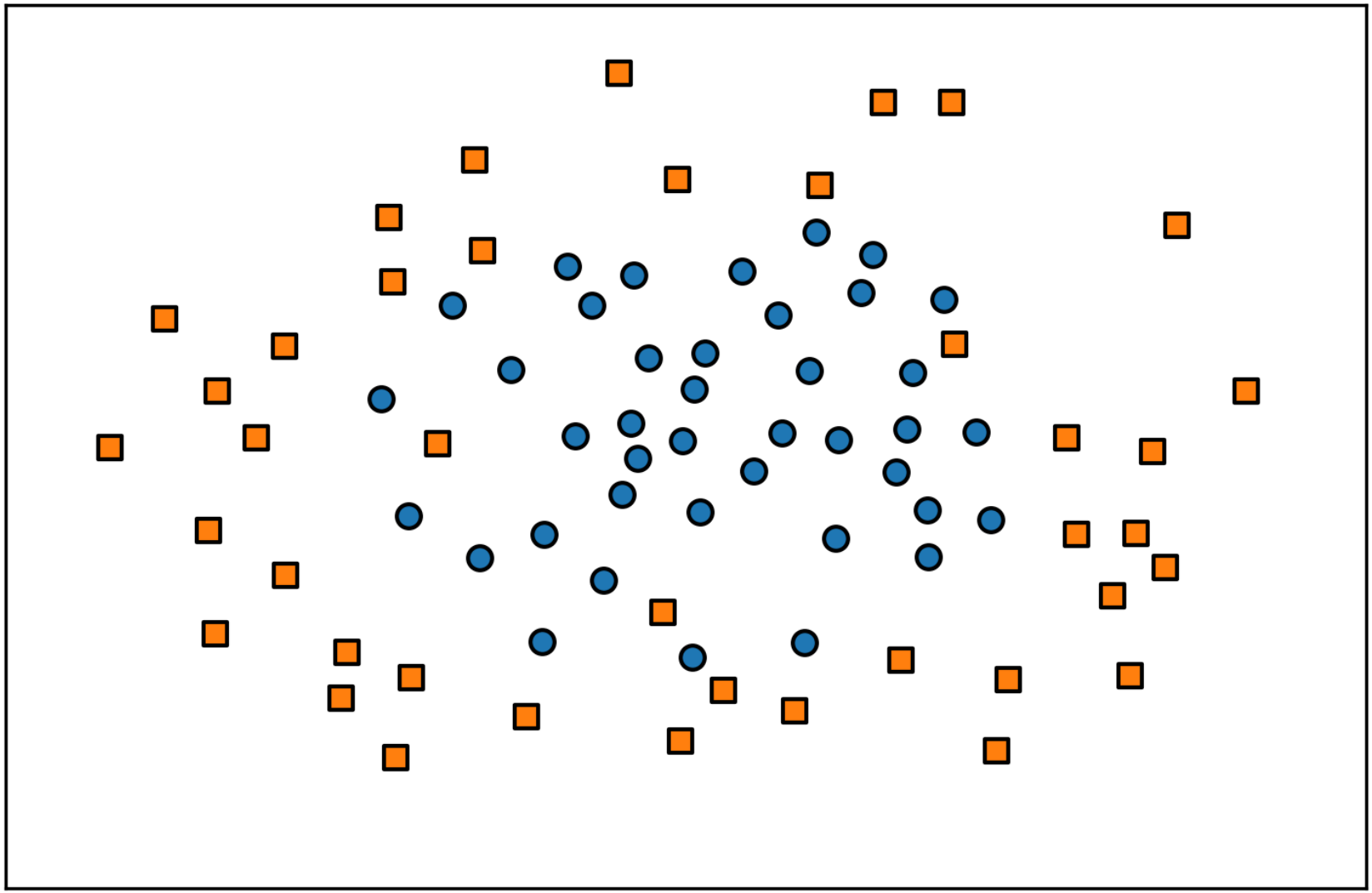
a) En cas d'égalité entre deux étiquettes, il faut préciser une règle pour choisir laquelle sera retenue.

b) L'algorithme construit une fonction $h : \mathscr{O} \to \mathscr{E}$, appelée fonction de prédiction, qui a chaque objet associe une étiquette. Attention : cette fonction dépend de k et du jeu de données utilisé pour la construire.

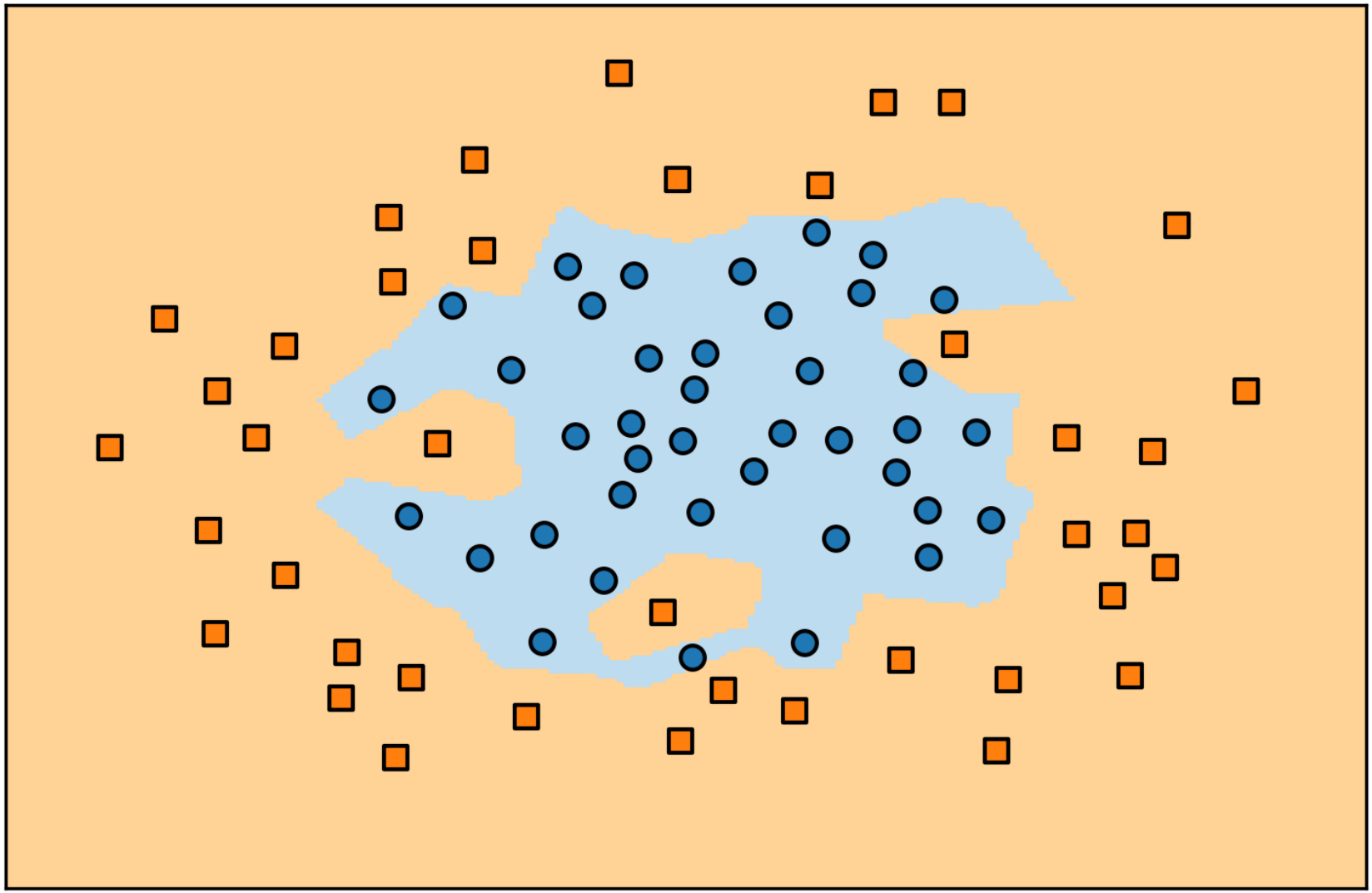
En pratique, on peut utiliser le code ci-dessous pour récupérer la liste des k éléments les plus proches de x parmi les objets $X = \{x_1, \ldots, x_n\}$ du jeu de données.

```
1  # Distance euclidienne entre deux vecteurs x1 et x2
2  def delta(x1, x2) -> float:
3          s = 0
4          for i in range(len(x1)):
5                  s += (x1[i]-x2[i])**2
6          return(s**(1/2))
7
8  # Liste des k éléments les plus proches de x
9  def plus_proches_voisins(k:int, x:float) -> list:
10         # On trie les données en fonction de leur distance à x
11         lst = sorted(X, key=lambda c: delta(c,x))
12         # On retourne les k premiers éléments de la liste
13         return lst[:k]
```

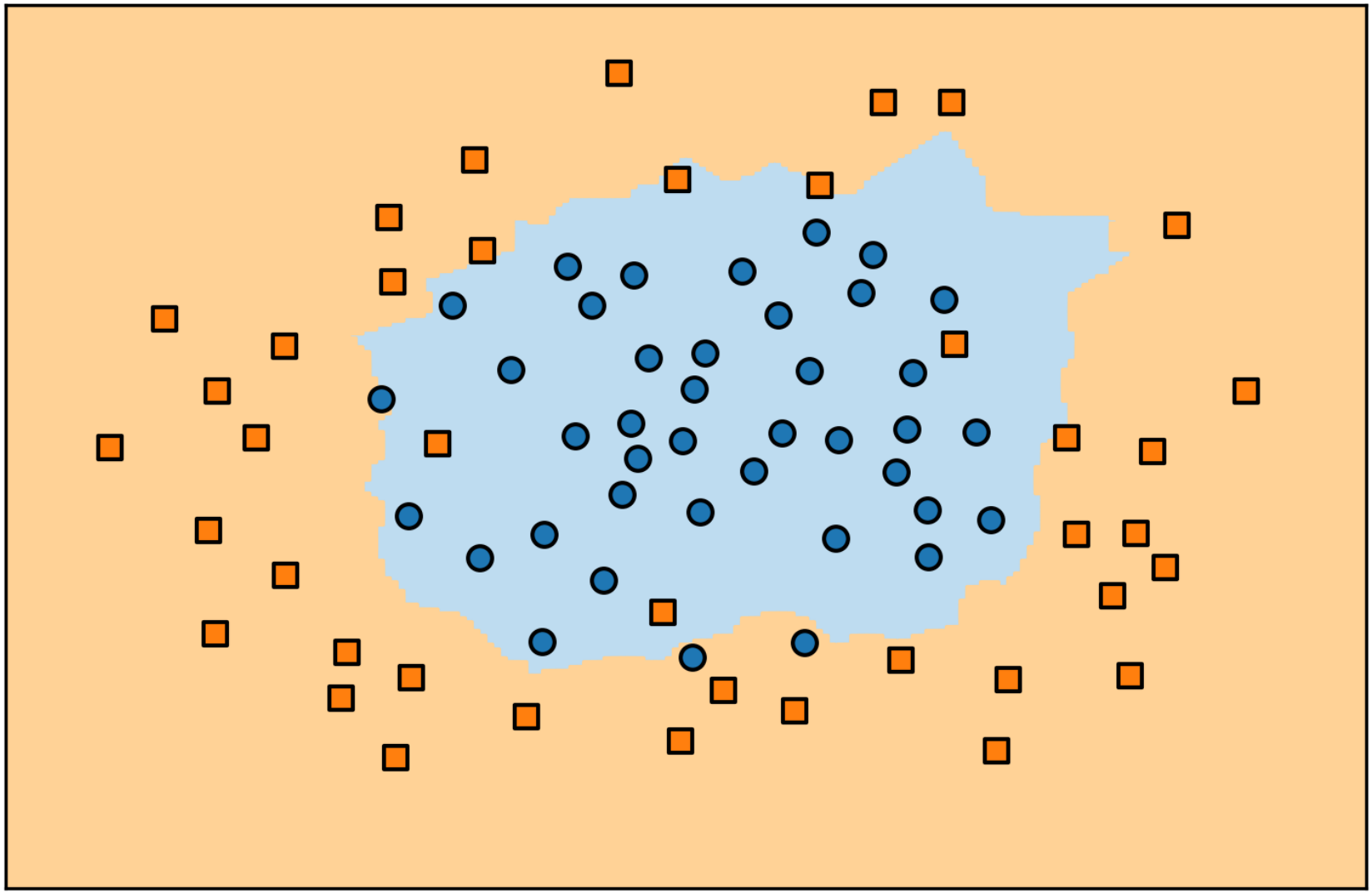
Dans l'exemple suivant, on considère un jeu de données constitué de 80 points de $\mathbb{R}^2$ étiquetés par 1 (représenté par des ronds bleus ci-dessous) et 2 (représenté par les carrés oranges ci-dessous). En appliquant l'algorithme des k plus proches voisins pour $k \in \{1, 6, 13\}$ à ces données, on obtient les fonctions de prédiction suivantes.



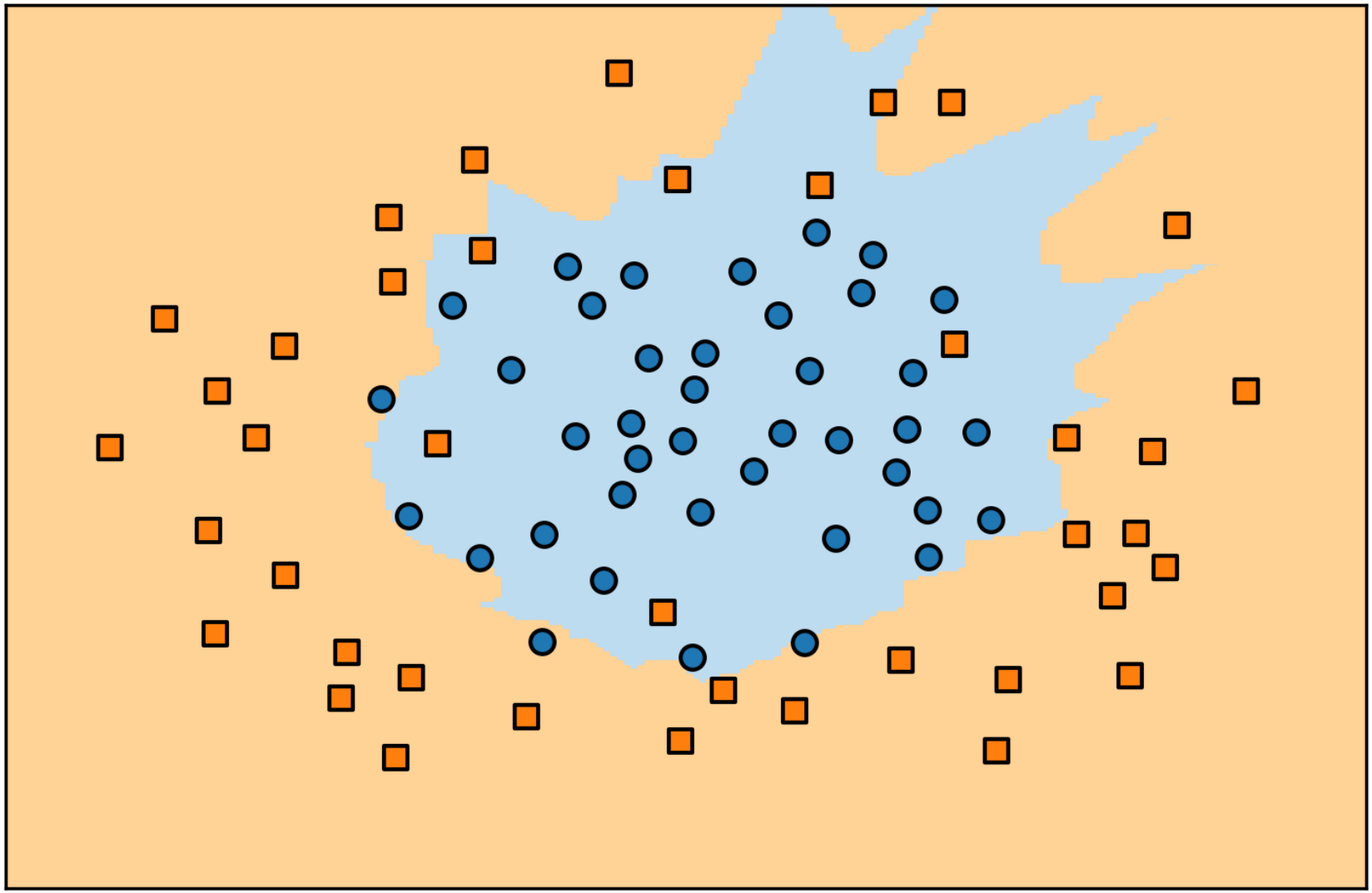
Un jeu de 80 données



Fonction de prédiction pour $k = 1$



Fonction de prédiction pour $k = 6$



Fonction de prédiction pour $k = 13$

Il est naturel de se demander si la fonction de prédiction $h : \mathscr{O} \to \mathscr{E}$ construite par l'algorithme des k plus proches voisins est de bonne qualité, i.e. si elle prédit correctement l'étiquette des nouvelles données. De plus, on peut également s'interroger sur le choix de la valeur de k à effectuer pour obtenir une bonne fonction de prédiction.

Pour ce faire, l'usage est de scinder l'ensemble des données $\mathscr{D} = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ en deux sous-ensembles :

- un ensemble de données $\mathscr{D}_a$ réservées à l'apprentissage (aussi appelé échantillon d'apprentissage) ;
- un ensemble de données $\mathscr{D}_t$ destinées à tester la qualité de l'apprentissage (aussi appelé échantillon de test).

En pratique, on réserve environ 80% des données pour l'échantillon d'apprentissage et 20% des données pour l'échantillon de test. Une fois la séparation des données effectuée, on utilise les données d'apprentissage pour construire la fonction de prédiction.

Définition (Matrice de confusion)

On numérote les étiquettes possibles de sorte que $\mathscr{E} = \{e_1, \dots, e_p\}$.
La matrice de confusion $M \in \mathscr{M}_p(\mathbb{R})$ est définie par : pour tout $(i,j) \in [\![1,p]\!]^2$, le coefficient $m_{i,j}$ de M est le nombre de données de l'échantillon de test d'étiquette e_i et prédites avec l'étiquette e_j.

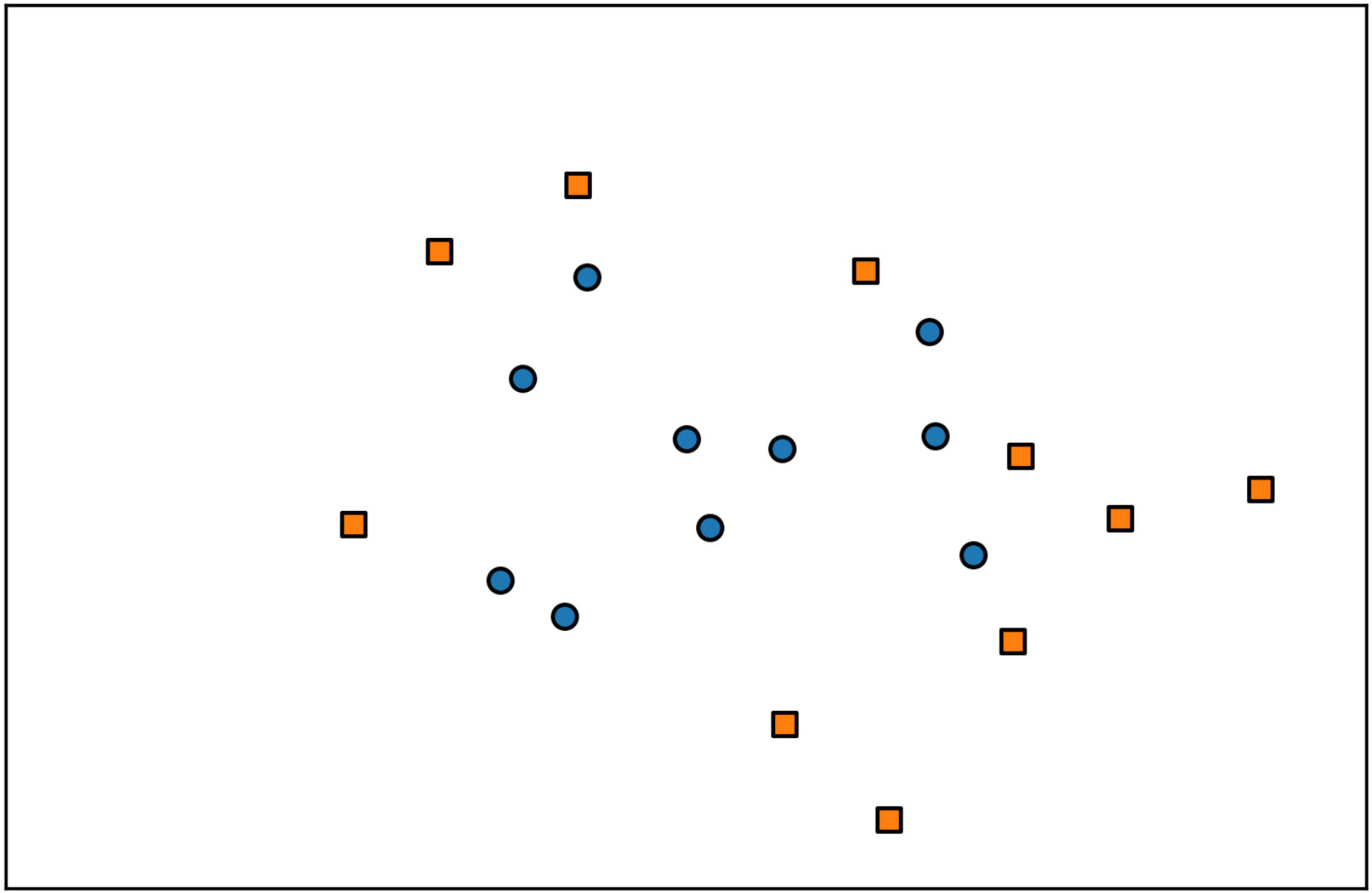
Remarques 2

a) La somme des coefficients de tous les coefficients de M est égale à la taille de l'échantillon de test.

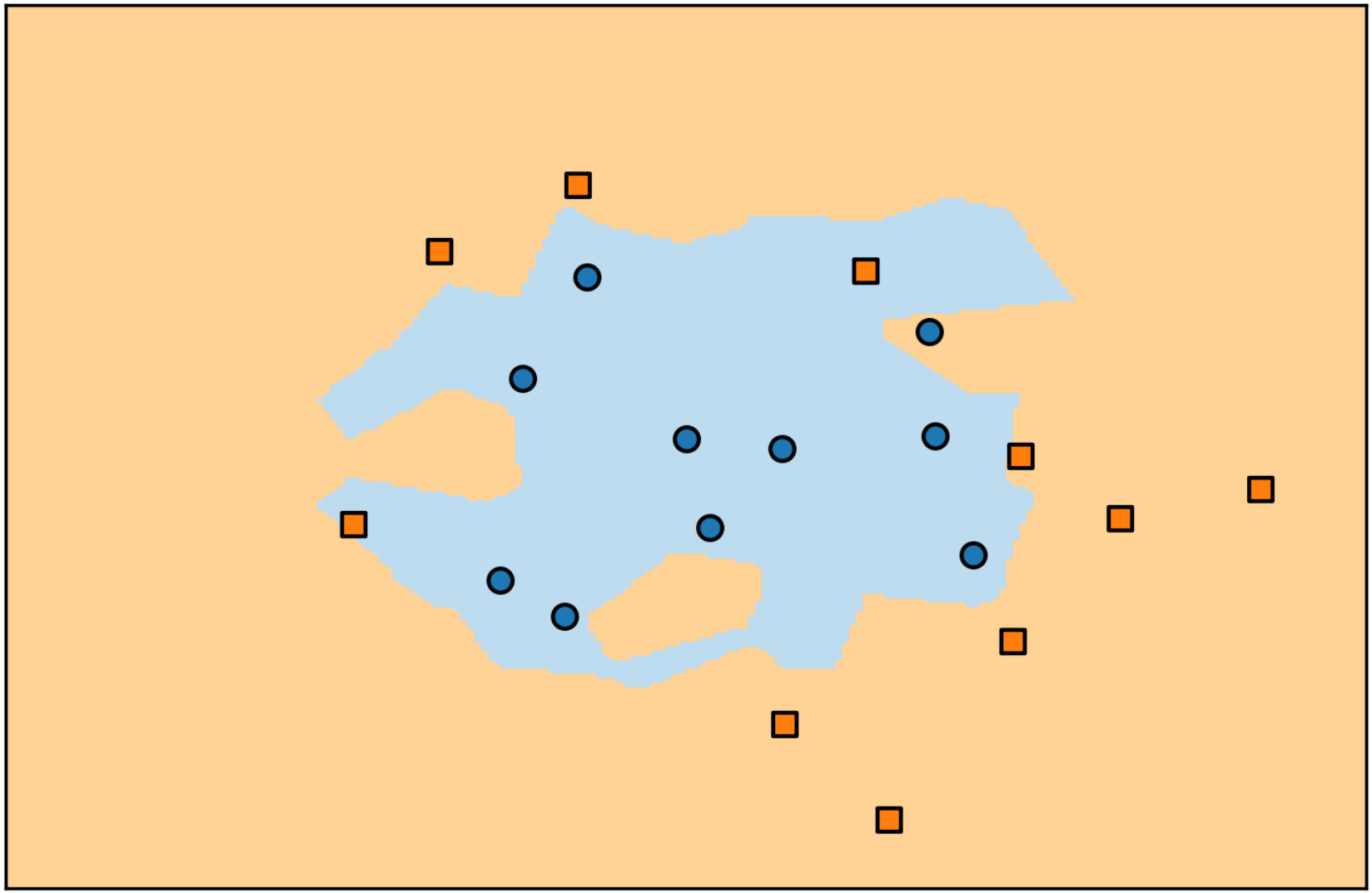
b) Le nombre d'erreurs de prédiction est la somme de tous les coefficients non diagonaux de M.

c) Certains auteurs préfèrent inverser les lignes et les colonnes dans la définition de la matrice de confusion.

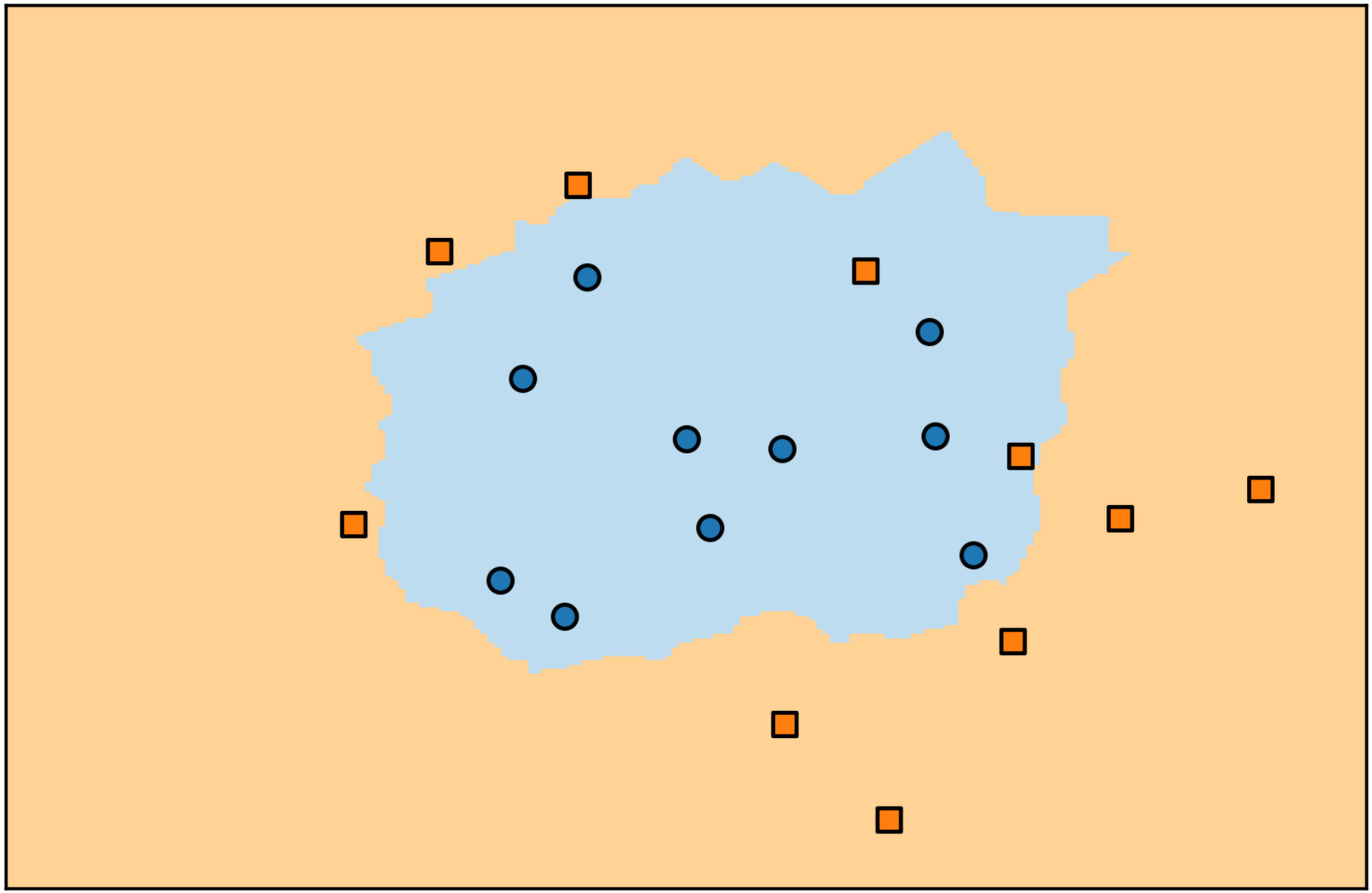
Dans l'exemple de la partie précédente, on avait en fait réservé 20 données (représentée ci-dessous) pour servir d'échantillon de test.



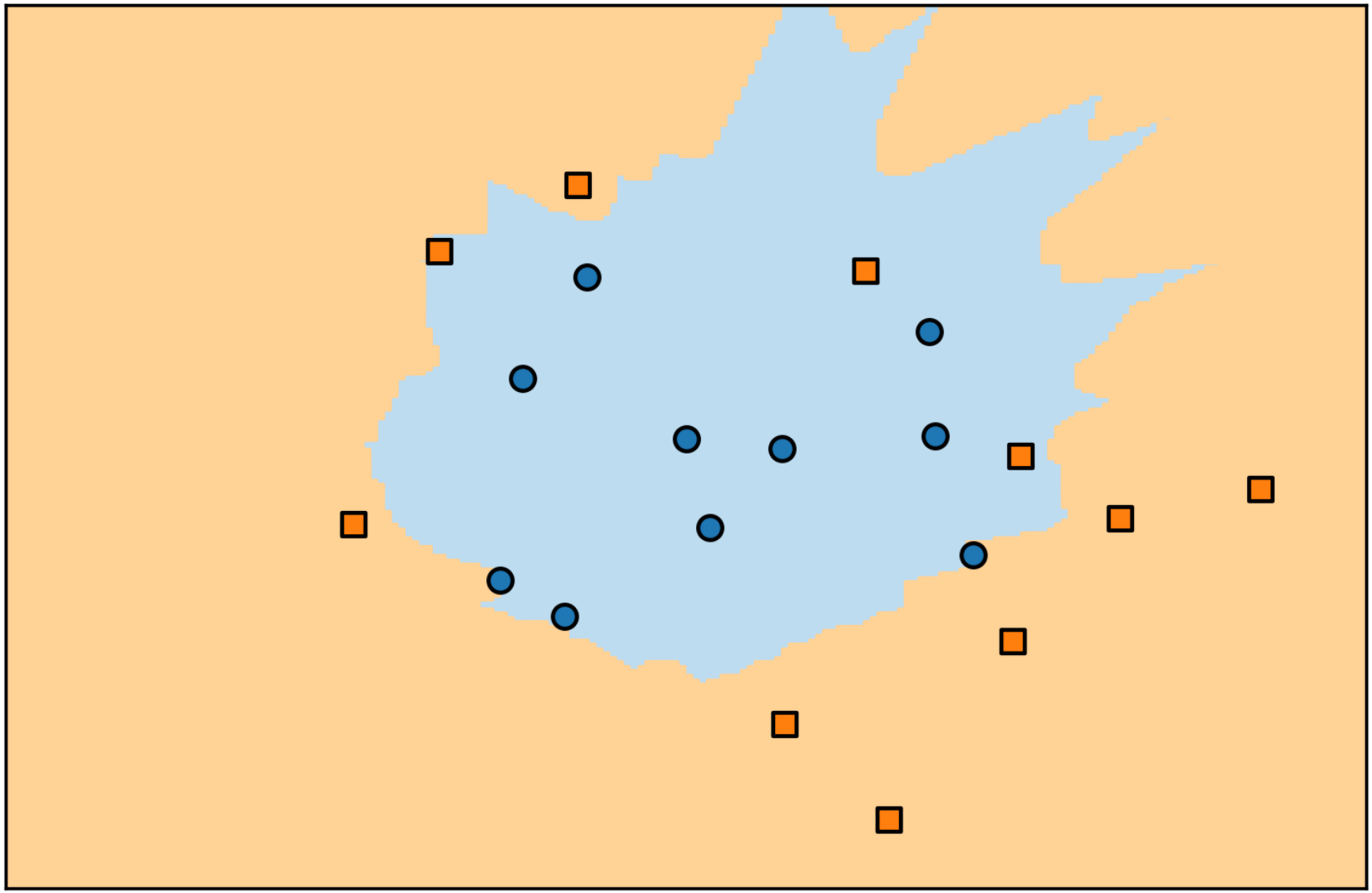
Échantillon de test avec 20 données



Évaluation de la prédiction pour $k = 1$



Évaluation de la prédiction pour $k = 6$

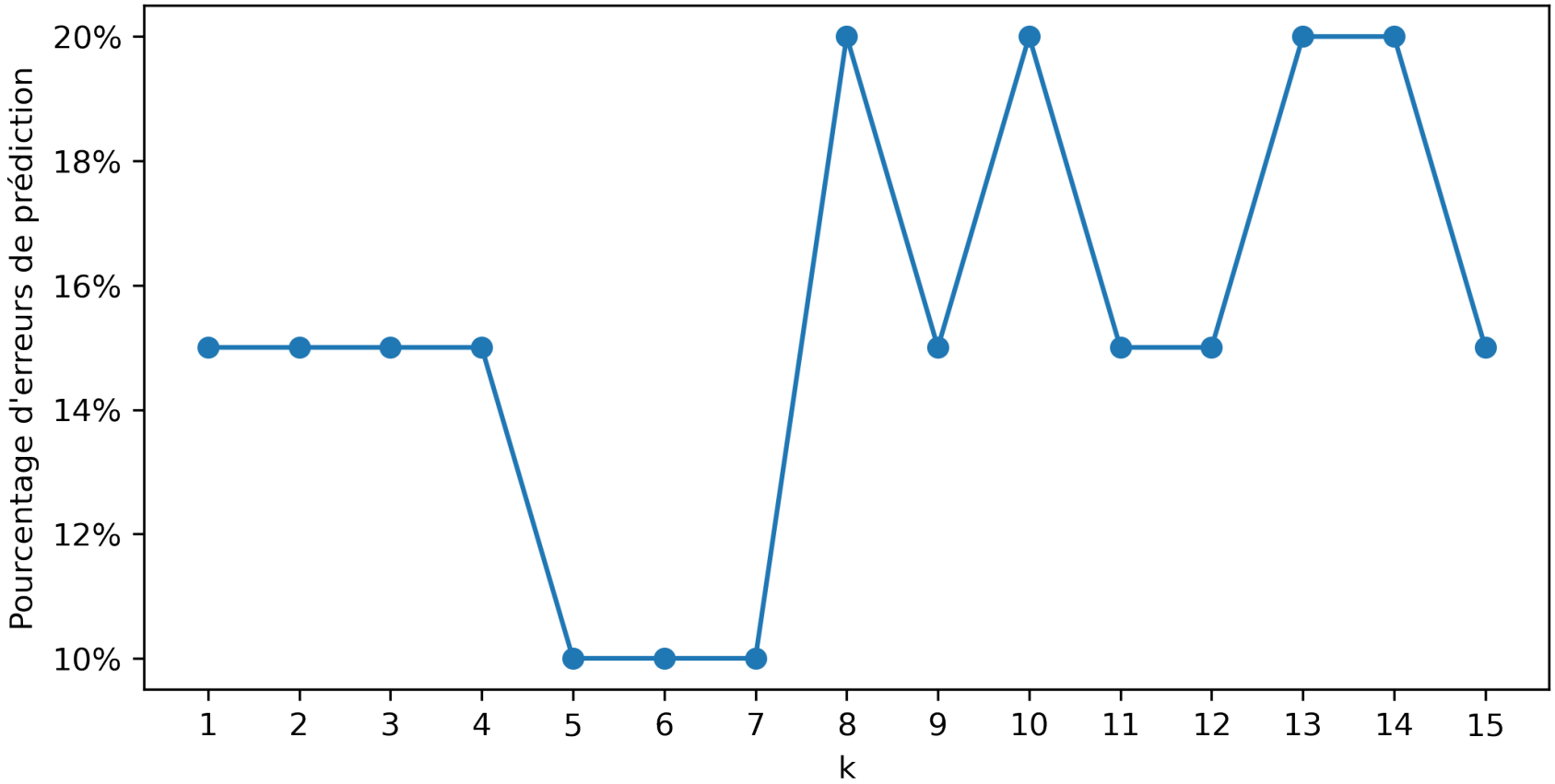


Évaluation de la prédiction pour $k = 13$

En notant M_k la matrice de confusion obtenue pour le paramètre k, on obtient que

$$M_1 = \begin{pmatrix} 9 & 1 \\ 2 & 8 \end{pmatrix}, \qquad M_6 = \begin{pmatrix} 10 & 0 \\ 2 & 8 \end{pmatrix}, \qquad M_{13} = \begin{pmatrix} 9 & 1 \\ 3 & 7 \end{pmatrix}.$$

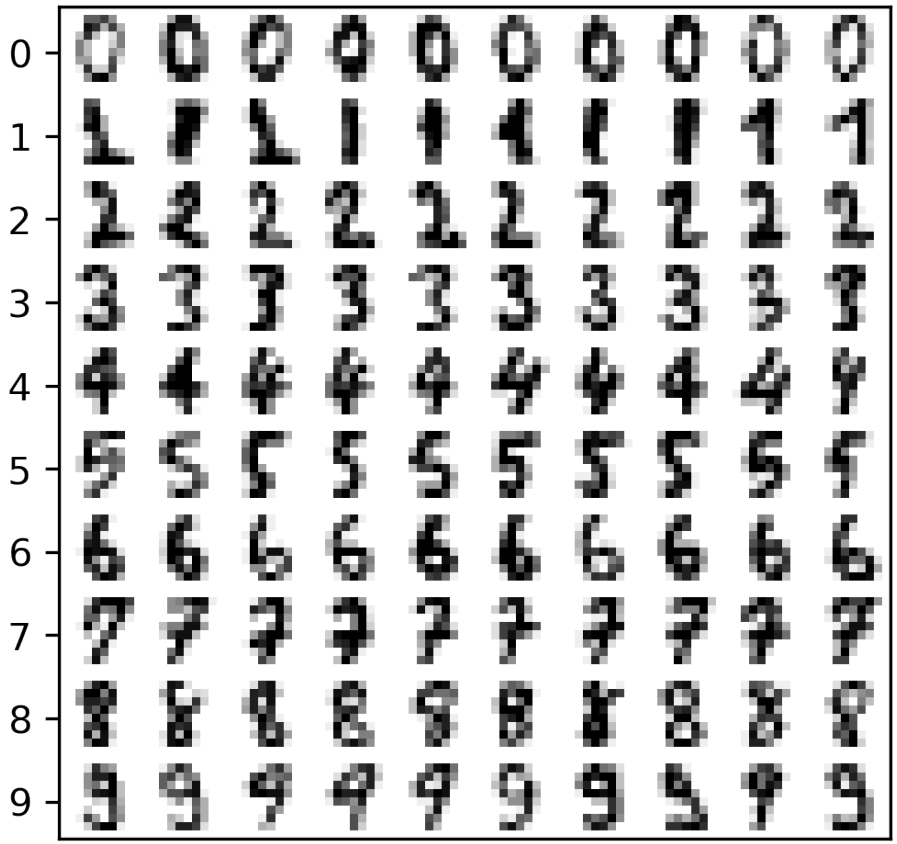
On peut également tracer le nombre d'erreurs en fonction de l'entier k choisi.



Pourcentage d'erreurs en fonction de k

En pratique, on choisit une valeur de k minimisant le nombre d'erreurs de prédiction. Dans l'exemple ci-dessus, on retiendra donc les valeurs $k = 5$, $k = 6$ ou $k = 7$.

On considère un jeu de données composé de 1797 images de 8×8 pixels en niveau de gris représentant des chiffres de 0 à 9 (environ 180 images pour chaque chiffre).



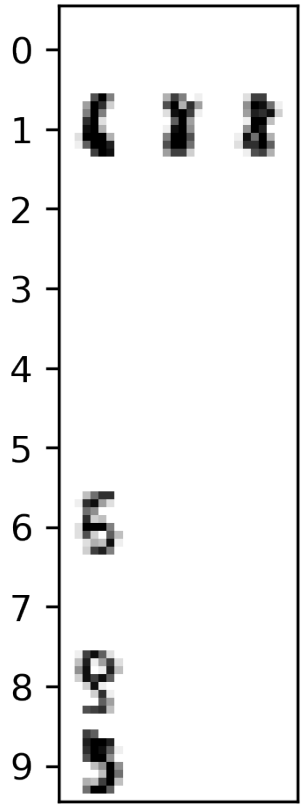
Un échantillon aléatoire de 100 images

Chacune de ces images peut être vue comme un élément de $\mathbb{R}^{8\times 8} = \mathbb{R}^{64}$, ce qui nous permet d'appliquer l'algorithme des k plus proches voisins en utilisant la distance euclidienne de $\mathbb{R}^{64}$.

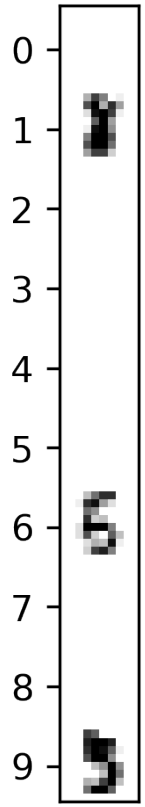
On scinde aléatoirement nos données en deux parties :

- un premier échantillon d'apprentissage composé de 1442 images (environ 80% du jeu de données) de sorte que chaque chiffre apparaisse environ 144 fois ;
- un second échantillon de test composé de 355 images (environ 20% du jeu de données) de sorte que chaque chiffre apparaisse environ 36 fois.

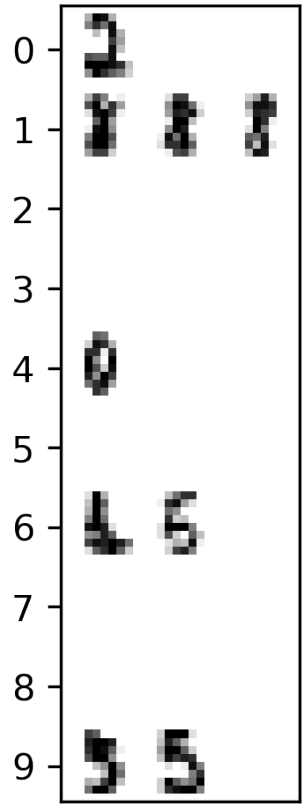
En appliquant l'algorithme des k plus proche voisins avec $k \in \{2, 5, 15\}$ et en testant la qualité de l'apprentissage avec l'échantillon de test, on obtient les erreurs de classification suivantes.



Les erreurs de prédiction pour $k = 2$



Les erreurs de prédiction pour $k = 5$

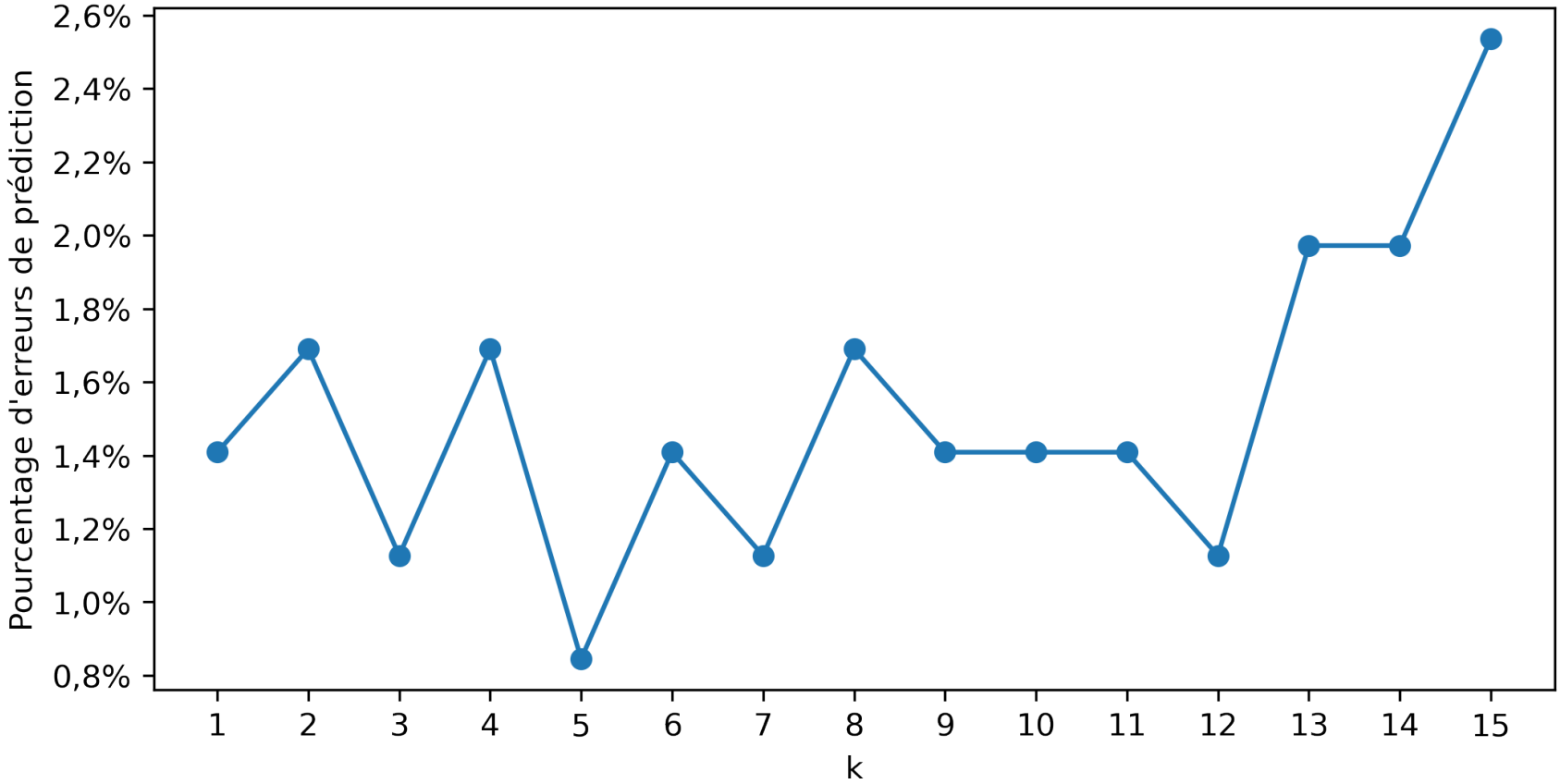


Les erreurs de prédiction pour $k = 15$

On peut déduire la matrice de confusion des graphiques précédents. Par exemple pour $k = 15$, on obtient la matrice suivante.

$$M_{15} = \begin{pmatrix} 34 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 35 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 34 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 36 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 36 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 33 & 1 & 0 & 0 & 2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 36 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 35 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 & 0 & 0 & 0 & 31 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 36 \end{pmatrix}.$$

On peut également tracer le nombre d'erreurs en fonction de l'entier k choisi.



Pourcentage d'erreurs en fonction de k

Dans l'exemple ci-dessus, on retiendra donc la valeur $k = 5$.

On considère une collection d'objets $X = \{x_1, \dots, x_n\}$ appartenant à $\mathscr{O} = \mathbb{R}^d$. Dans ce nouveau contexte, on ne suppose plus que les données sont étiquetées.

L'objectif est d'être capable de répartir ces données en un certain nombre de classes $k \in [\![1, n]\!]$ fixé à l'avance, de sorte que chaque classe contienne des données proches.

Pour formaliser le problème, commençons par rappeler qu'on dit que $(C_1, \dots, C_k)$ est une partition de X si

(i) les ensembles $C_1, \dots, C_k$ sont des parties non vides de X ;

(ii) la réunion des ensembles $C_1, \dots, C_k$ est égale à X ;

(iii) les ensembles $C_1, \dots, C_k$ sont deux à deux disjoints.

Pour toute partition $(C_1, \dots, C_k)$ de X, on définit

- les centres d'inertie $\mu(C_1), \dots, \mu(C_k) \in \mathbb{R}^d$ de $C_1, \dots, C_k$ par

$$\forall i \in [\![1, k]\!], \quad \mu(C_i) = \frac{1}{\mathrm{Card}(C_i)} \sum_{x \in C_i} x \,,$$

- les moments d'inertie $\sigma^2(C_1), \dots, \sigma^2(C_k) \in \mathbb{R}$ de $C_1, \dots, C_k$ par

$$\forall i \in [\![1, k]\!], \quad \sigma^2(C_i) = \frac{1}{\mathrm{Card}(C_i)} \sum_{x \in C_i} \left\| x - \mu(C_i) \right\|^2 .$$

Remarques 3

a) Le centre d'inertie est un indicateur de position : le vecteur $\mu(C_i)$ permet de situer les éléments de C_i dans l'espace $\mathbb{R}^d$.

b) Le moment d'inertie est un indicateur de dispersion : plus les points de la classe C_i sont proches de son centre $\mu(C_i)$, plus le moment d'inertie $\sigma^2(C_i)$ sera faible.

Pour répondre au problème présenté ci-dessus, l'objectif de la machine sera de minimiser la moyenne des moments d'inertie pondérée par la taille de ces classes, i.e. de minimiser la fonction

$$\varphi : (C_1, \dots, C_k) \mapsto \sum_{i=1}^{k} \frac{\mathrm{Card}(C_i)}{\mathrm{Card}(X)} \sigma^2(C_i) = \sum_{i=1}^{k} \sum_{x \in C_i} \left\| x - \mu(C_i) \right\|^2 .$$

En général, pour déterminer le minimum global de la fonction précédente, il faudrait effectuer une recherche exhaustive sur l'ensemble des partitions en k parties de X. Pour des raisons de complexité temporelle, cette approche n'est pas envisageable en pratique. C'est pourquoi nous allons privilégier l'utilisation d'un algorithme glouton fournissant une « bonne » solution (mais pas nécessairement la meilleure).

Le principe de l'algorithme construisant cette « bonne » solution est décrit ci-dessous.

Algorithme des k-moyennes

Entrées : une partie $X = \{x_1, \dots, x_n\}$ de $\mathbb{R}^d$, l'entier k.

1) On choisit aléatoirement k centres d'inertie distincts $\mu_1, \dots, \mu_k$ dans $\mathbb{R}^d$.
2) Chaque donnée $x_j \in X$ est associée au centre μ_j le plus proche. On obtient ainsi k classes $C_1, \dots C_k$.
3) On calcule les centres d'inertie $\mu(C_1), \dots, \mu(C_k)$ respectifs des classes $C_1, \dots C_k$.
4) On a deux cas possibles :
   (i) Si $(\mu(C_1), \dots, \mu(C_k)) = (\mu_1, \dots, \mu_k)$, alors l'algorithme s'arrête et renvoie $(C_1, \dots, C_k)$;
   (ii) sinon on retourne à l'étape 2 en remplaçant $(\mu_1, \dots, \mu_k)$ par $(\mu(C_1), \dots, \mu(C_k))$.

Théorème

L'algorithme des k-moyennes se termine.

Démonstration (Non exigible)

On peut vérifier qu'à chaque fois qu'on passe par l'étape 4.(ii), la quantité $\varphi(C_1, \ldots, C_k)$ décroit strictement. Comme il n'y a qu'un nombre fini de partitions de X en k parties, l'algorithme se termine nécessairement.

Remarques 4

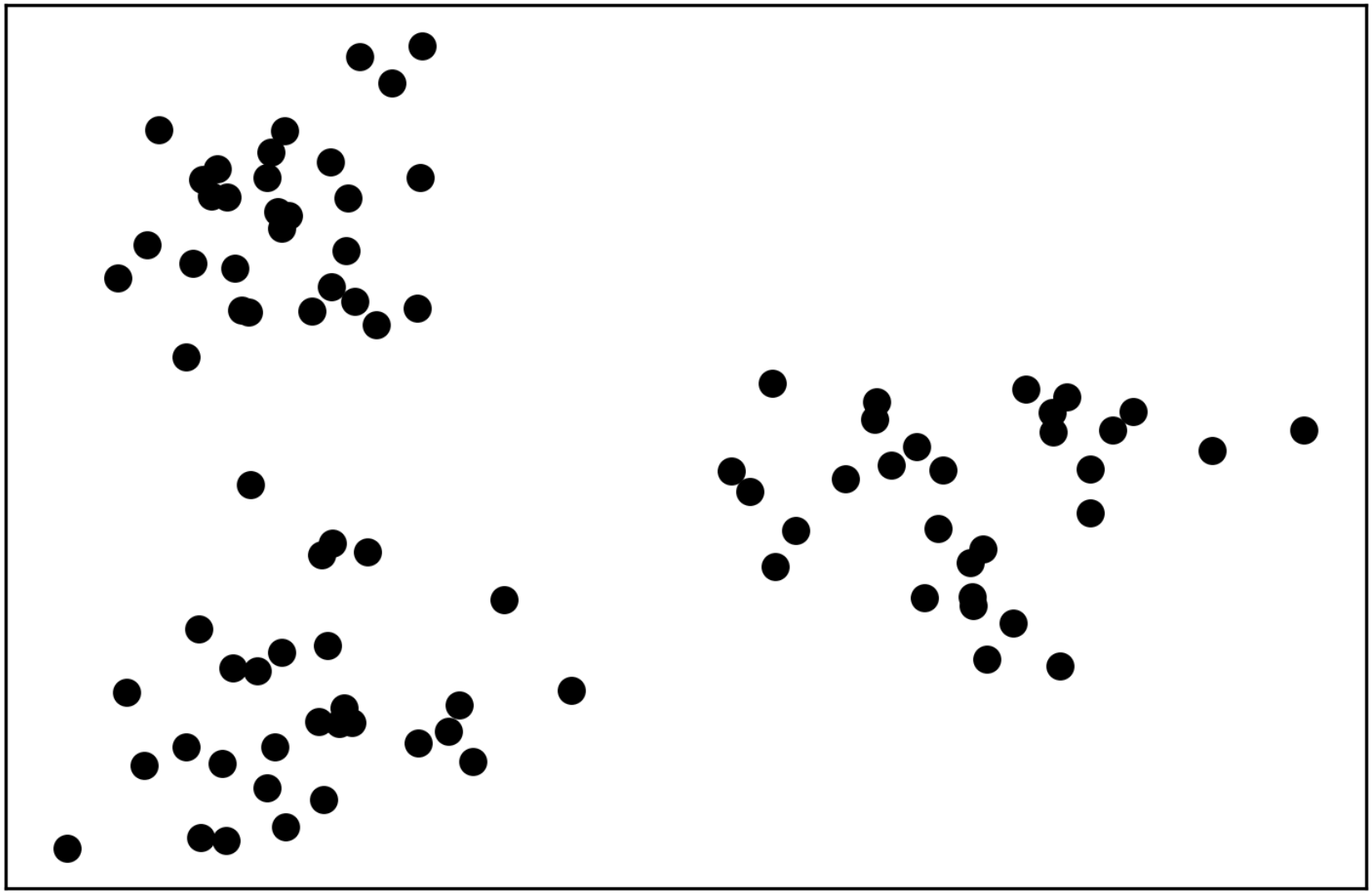
a) Rappelons que la partition $C_1, \ldots, C_k$ de X renvoyée par l'algorithme des k-moyennes n'est pas nécessairement celle réalisant le minimum global de la fonction φ : il s'agit seulement d'un minimum local.

b) L'algorithme est non déterministe : comme les centres d'inertie sont choisis aléatoirement à la première étape, on obtient en général des résultats différents lorsqu'on applique plusieurs fois l'algorithme.

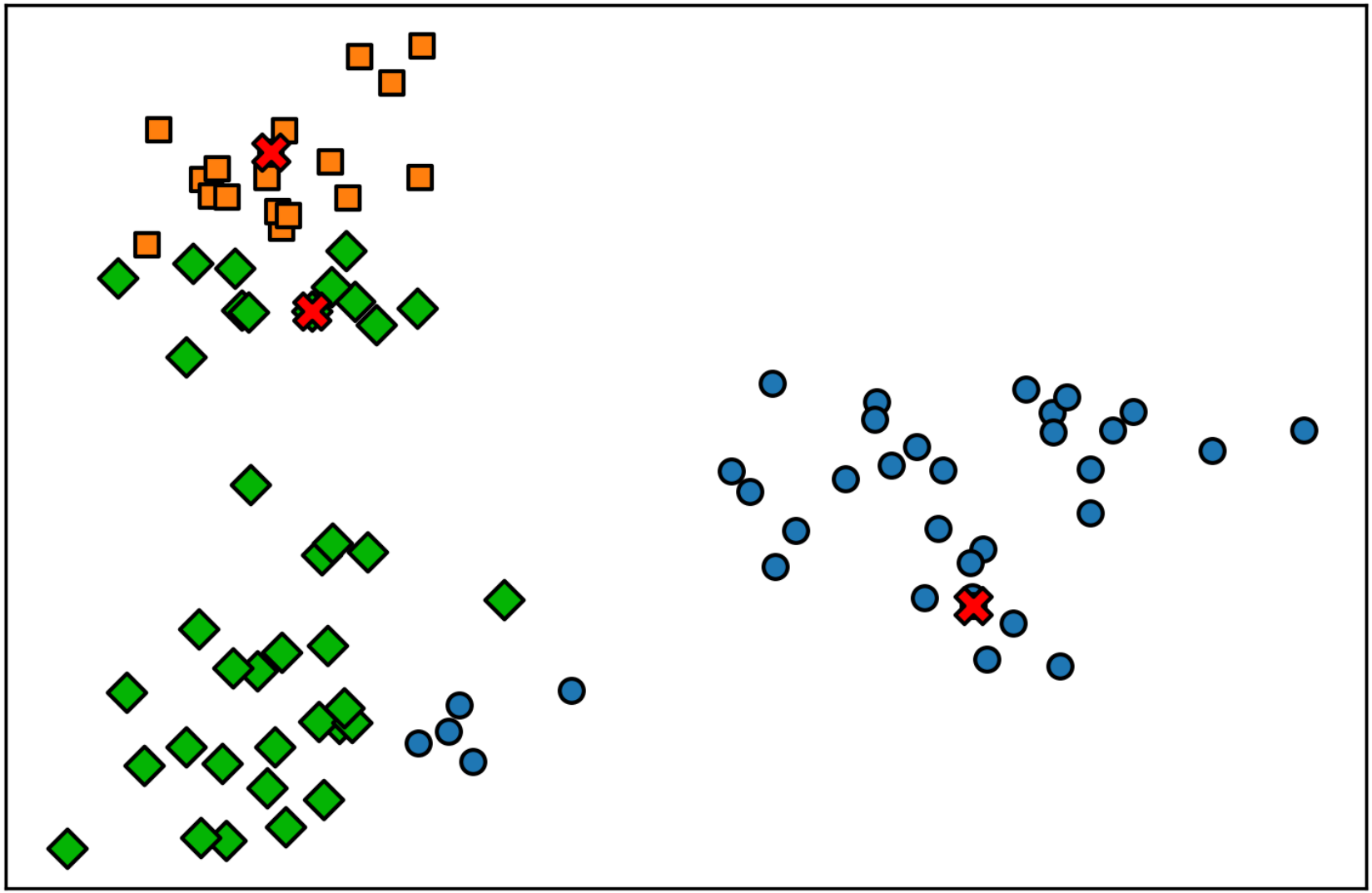
c) Dans certains cas, le résultat obtenu n'est pas satisfaisant (voir un exemple dans la partie suivante).

Dans l'exemple suivant, on considère un jeu de données constitué de 90 points de $\mathbb{R}^2$. On observe naturellement sur le schéma ci-dessus que les points se regroupent selon trois classes.

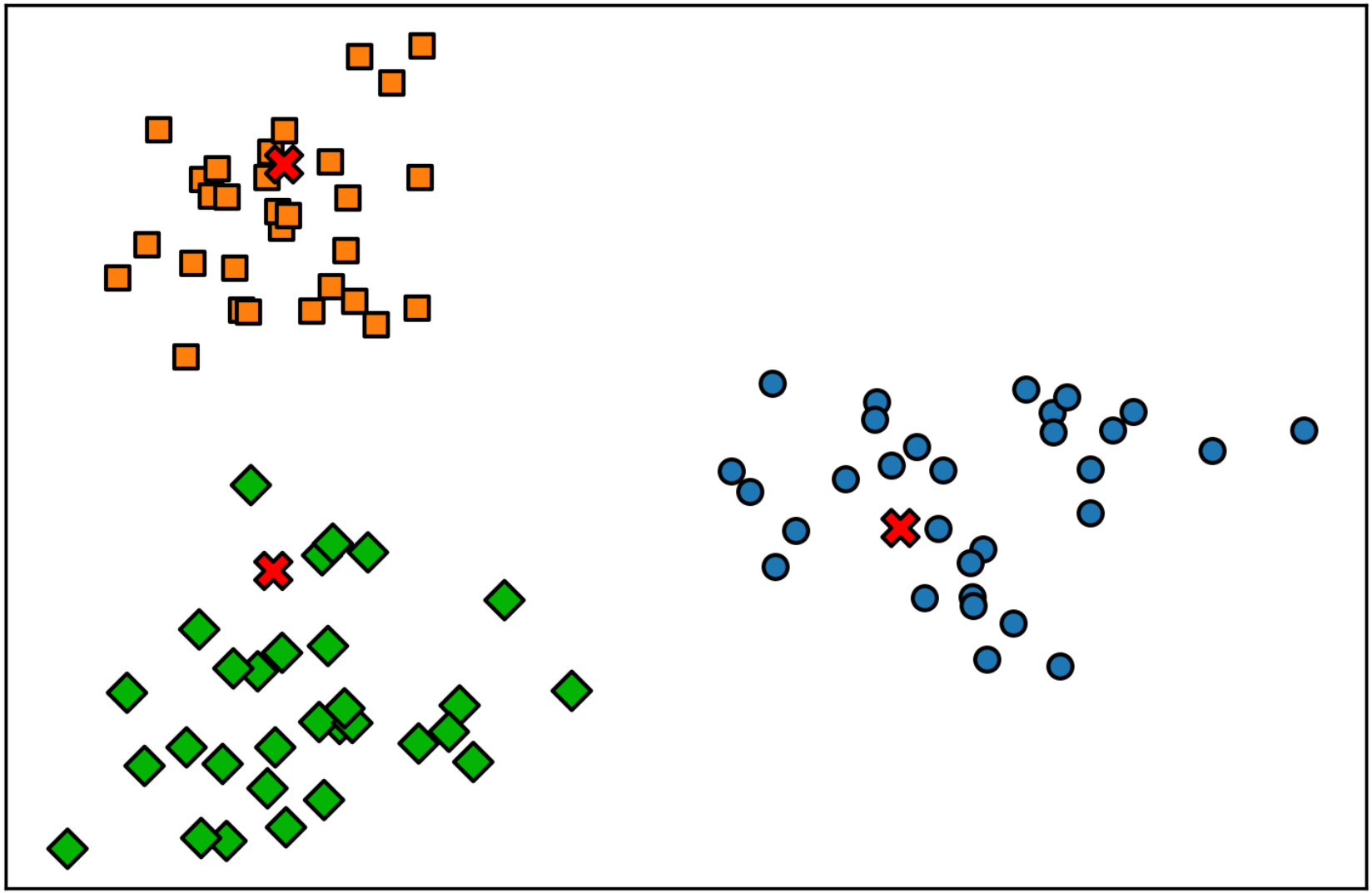
Appliquons l'algorithme des k-moyennes pour $k = 3$ au jeu de données ci-dessus. Dans les illustrations suivantes, on représente les différentes classes par des ronds bleus, des carrés oranges et des losanges verts tandis que les centres d'inertie sont représentés par des croix rouges.



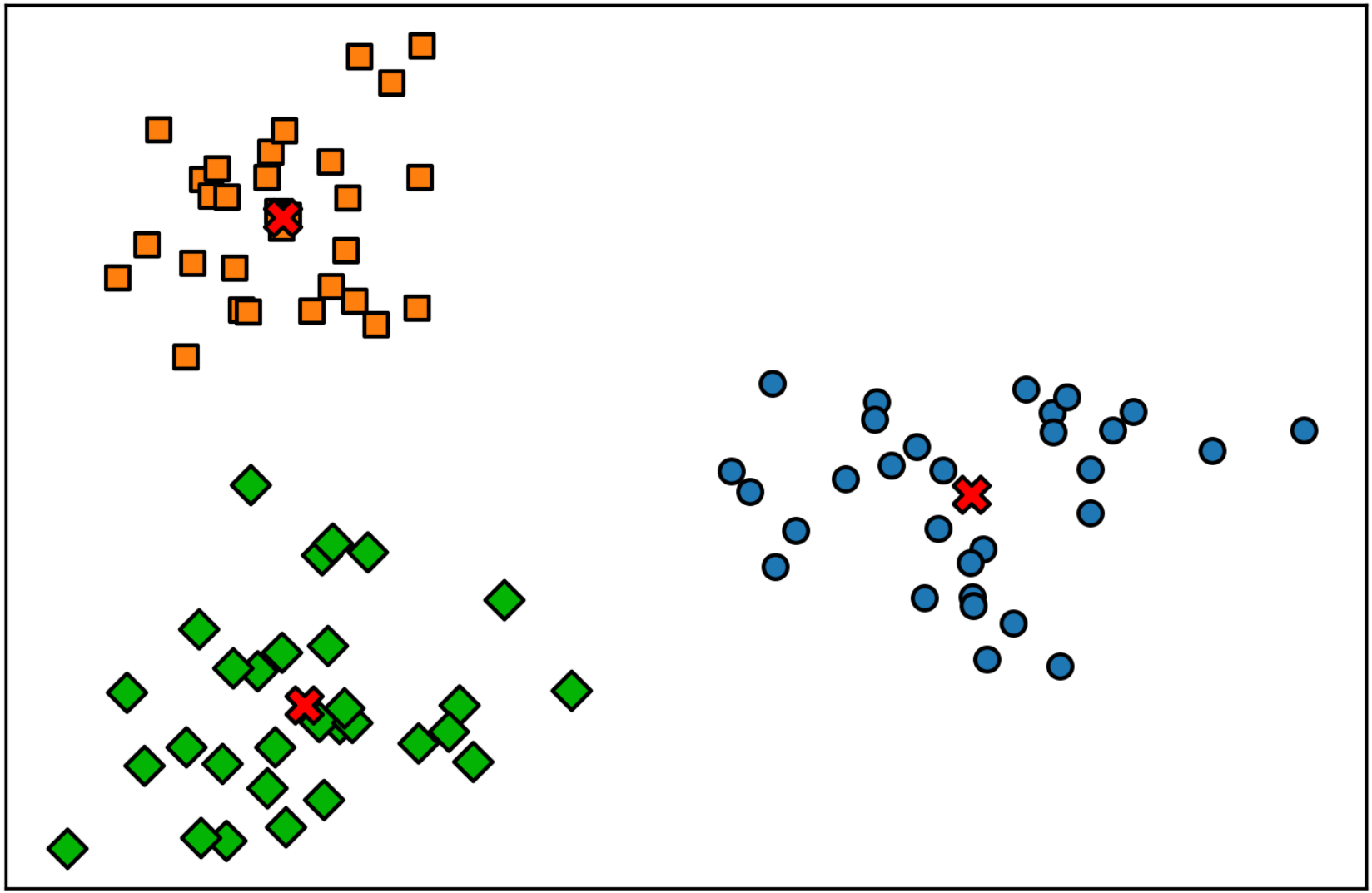
Un jeu de 90 données



Premier tour dans l'algorithme



Deuxième tour dans l'algorithme

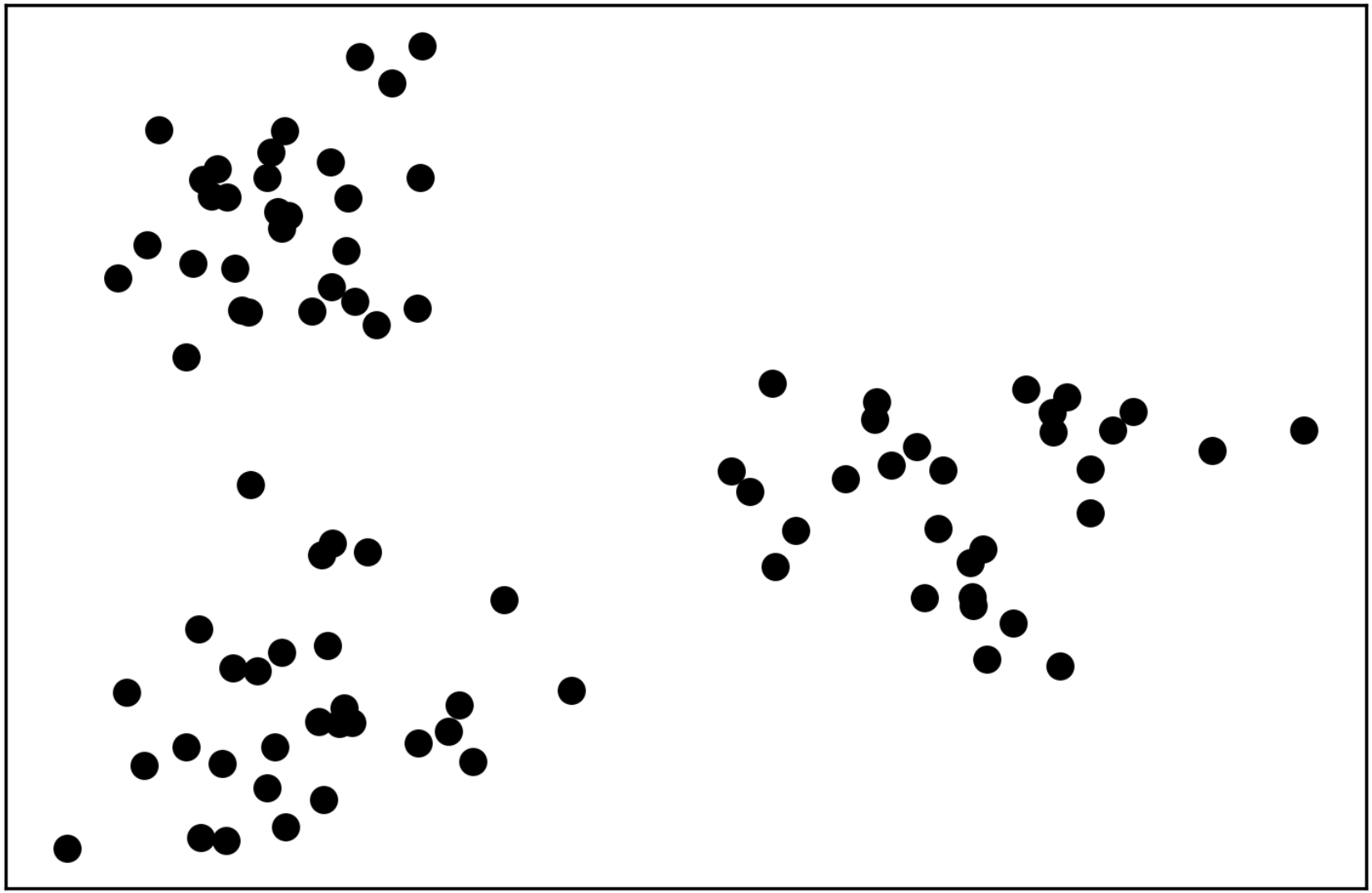


Dernier tour dans l'algorithme

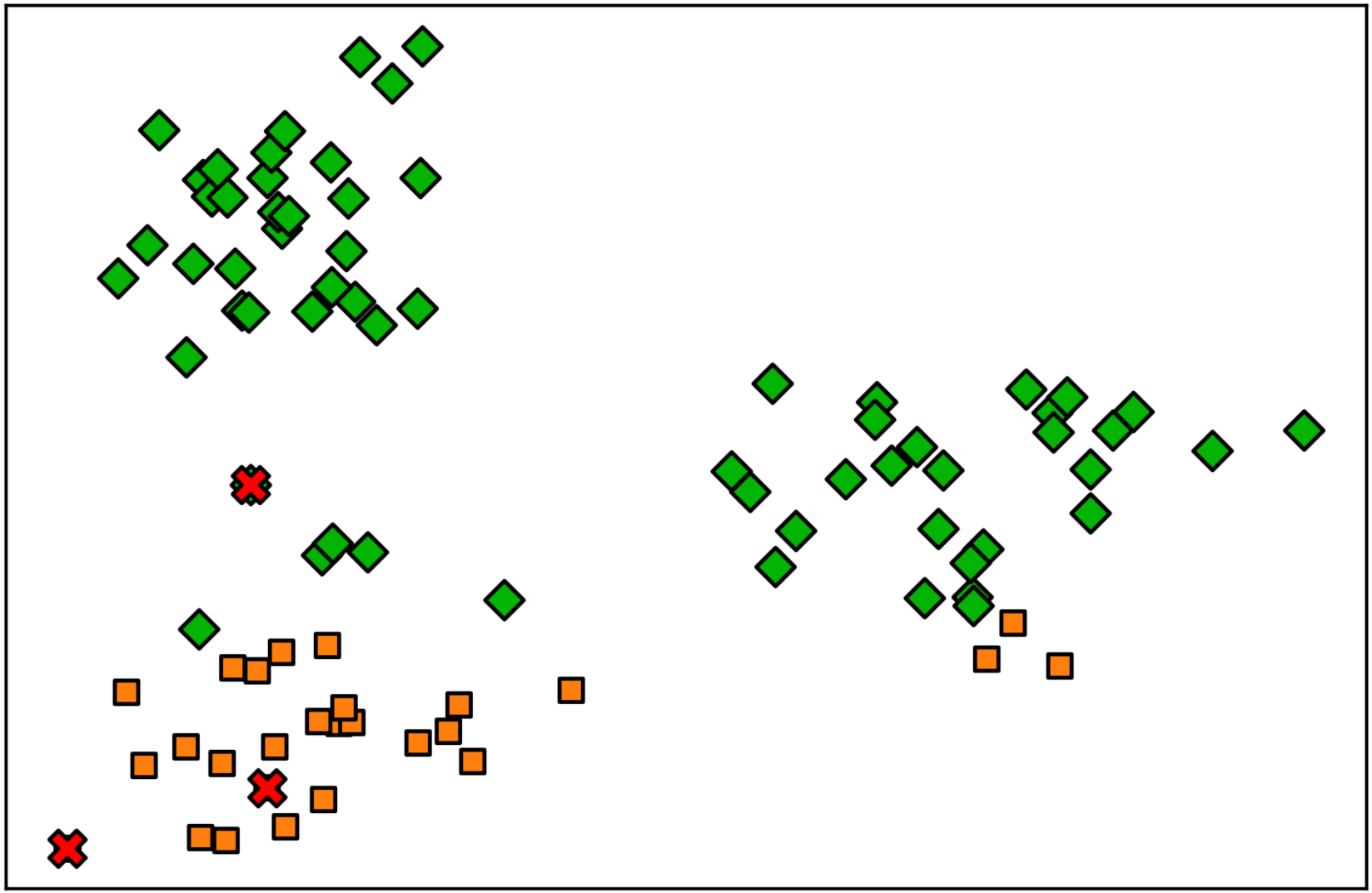
On constate que l'algorithme a bien permis de retrouver les trois classes que nous visualisions à l'œil.

ATTENTION

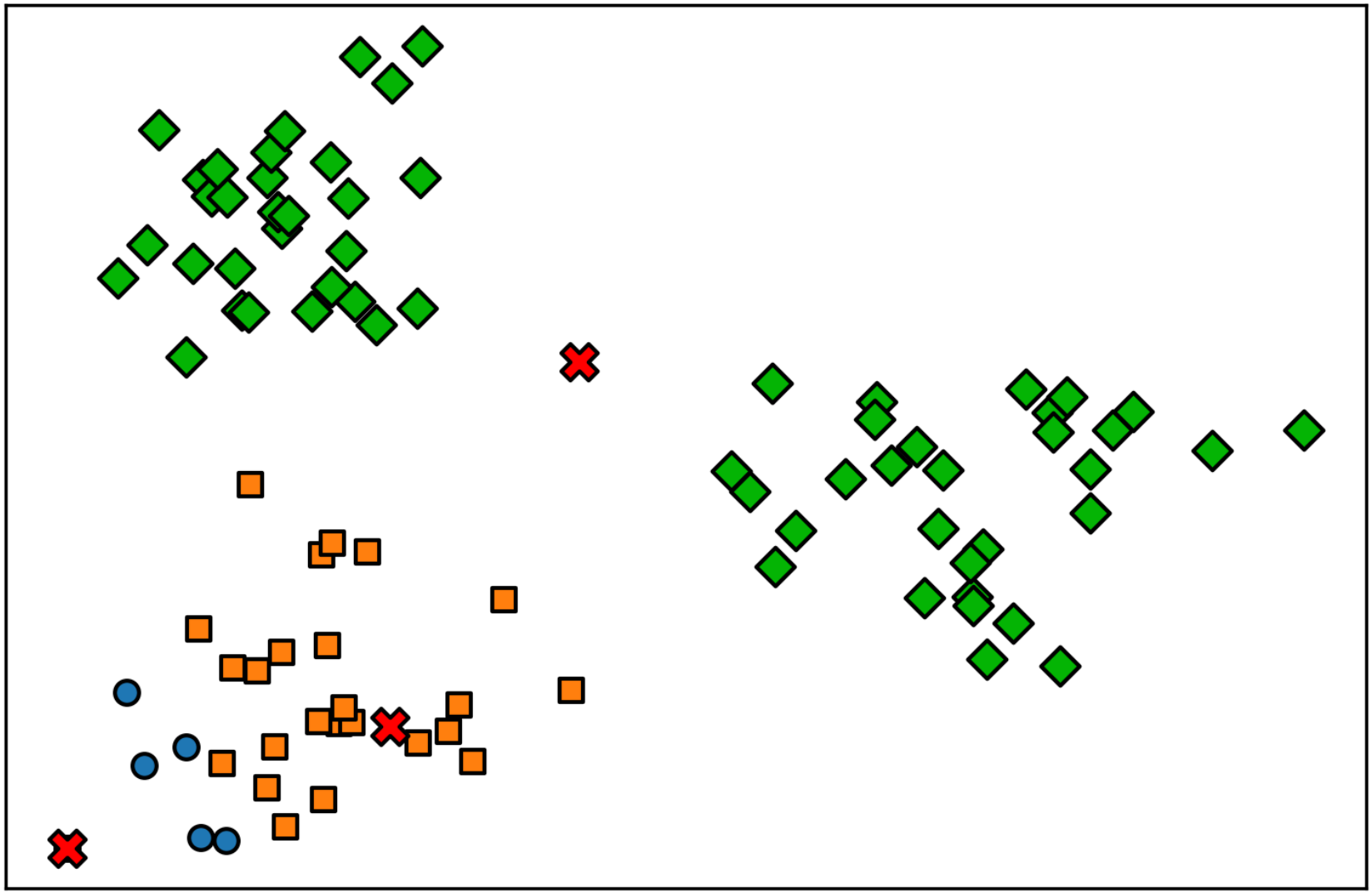
En fonction du choix des centres d'inertie lors de l'initialisation de l'algorithme, il peut arriver, comme sur l'exemple ci-dessous, que le résultat final ne soit pas satisfaisant.



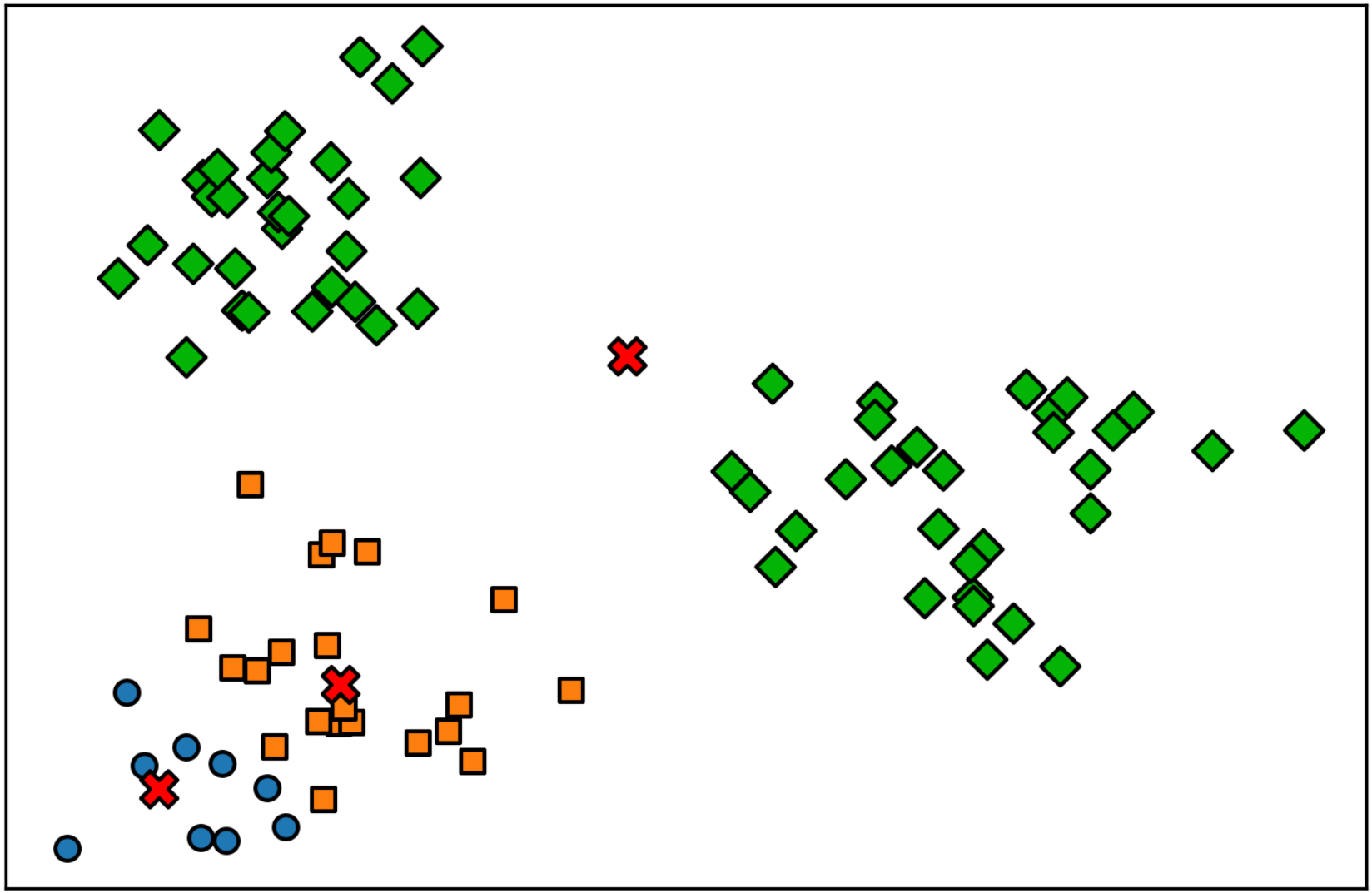
Un jeu de 90 données



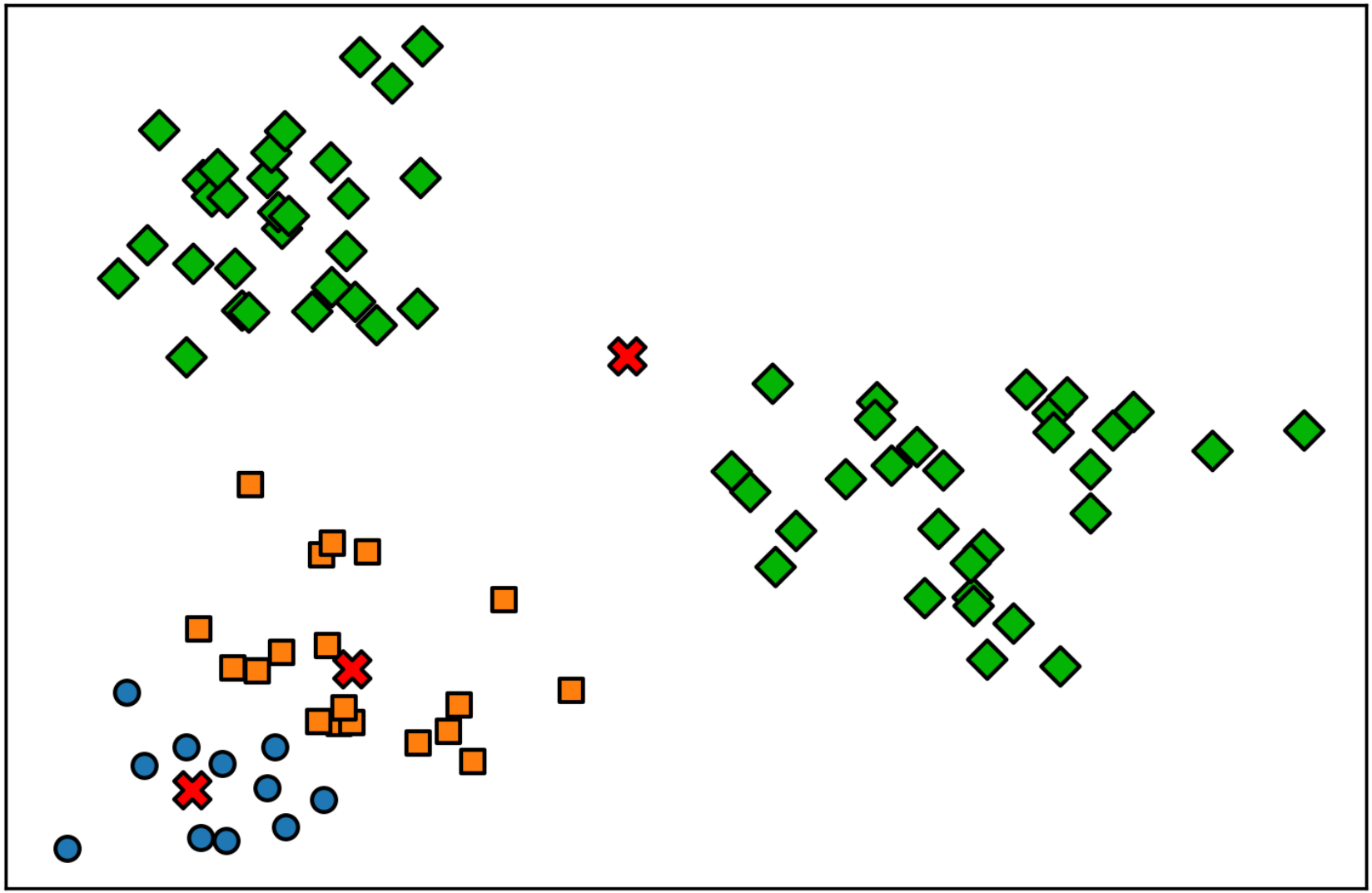
Premier tour dans l'algorithme



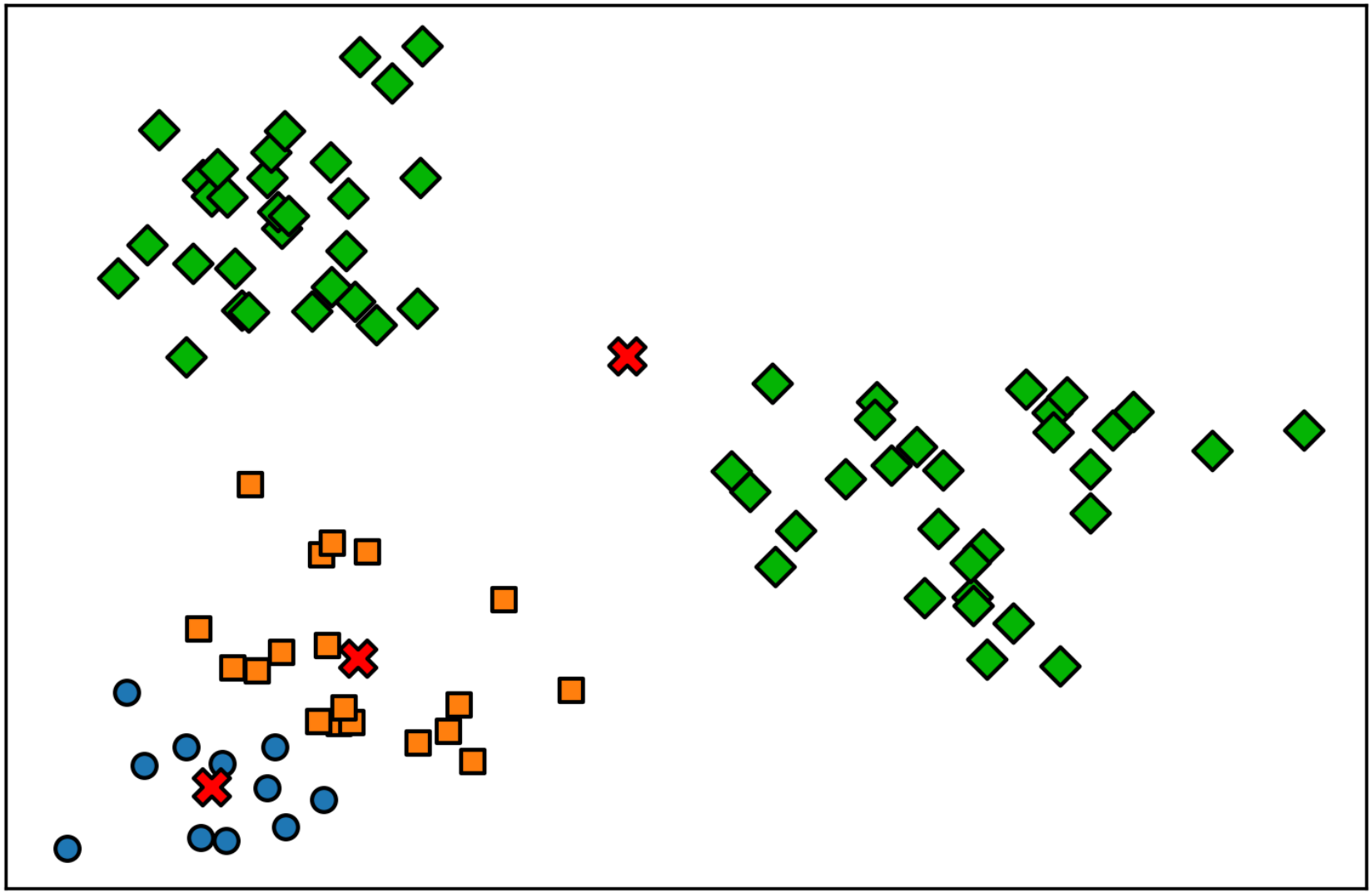
Deuxième tour dans l'algorithme



Troisième tour dans l'algorithme



Quatrième tour dans l'algorithme



Cinquième tour dans l'algorithme