Garanties statistiques et algorithmiques des méthodes de clustering

Raphael Razafindralambo, Jeremy Sarri, Han Di Zhang

January 2023

Table des matières

1 Introduction

L'apprentissage non supervisé et l'apprentissage supervisé sont deux méthodes d'apprentissage automatique (ou *machine learning* en anglais) qui sont utilisées pour apprendre à un système informatique à effectuer une tâche donnée.

— L'apprentissage supervisé consiste à fournir au système des exemples d'entrée (par exemple des images ou des données) ainsi que les sorties attendues correspondantes (par exemple les étiquettes de classe), afin que le système puisse apprendre à associer les entrées aux sorties correctes. Le système est « supervisé »par l'utilisateur qui fournit les exemples d'apprentissage.

— L'apprentissage non supervisé, en revanche, consiste à fournir uniquement des exemples d'entrée sans étiquettes de classe correspondantes. Le système doit alors trouver des structures dans les données sans aucune indication externe sur la bonne réponse. Les deux types d'apprentissage sont importants dans différents domaines de l'IA et sont utilisés pour des applications telles que la reconnaissance de la parole, la traduction automatique, la détection de fraude, la recommandation de produits, etc.

Par exemple, la classification d'images de chats et de chiens est un exemple d'apprentissage supervisé, car les étiquettes de classe (chats ou chiens) sont fournies avec les images. En revanche, la segmentation d'image en régions d'intérêt est un exemple d'apprentissage non supervisé, car il n'y a pas de réponse correcte à fournir à l'avance.

Le clustering est une tâche non supervisé visant à regrouper des données similaires en ensembles homogènes appelés clusters. Cette méthode est très utilisée dans différents domaines tels que la biologie, la finance, la géographie, la recherche d'information, etc. En biologie, le clustering est utilisée pour regrouper les gènes ou les protéines en fonction de leur similarité d'expression ou de séquence, afin d'identifier des groupes fonctionnels ou des régions évolutives conservées. En finance, elle est utilisée pour regrouper les actifs financiers en portefeuilles homogènes en termes de risque et de rendement. En géographie, elle est utilisée pour regrouper les régions en fonction de leurs caractéristiques géographiques et socio-économiques communes. Dans le commerce électronique, le clustering est utilisé pour déterminer l'intention de l'utilisateur et segmenter les clients selon différents critères.

Dans ce mémoire,nous allons étudier la méthode K-means, une méthode de clustering très utilisée, qui consiste à diviser les données en K clusters en minimisant la variance intra-cluster.

Plus précisément, nous allons étudier les garanties statistiques et algorithmiques pour la méthode K-means :

1. **Garanties statistiques**. Nous allons étudier la consistance du modèle des K-means à travers divers outils mathématiques.
2. **Garanties algorithmiques**. Nous allons analyser la convergence de l'algorithme. La méthode converge-t-elle en un nombre fini d'itérations ? Si c'est le cas, à quelle vitesse ? Quelle initialisation choisir ?
3. **Etude de K-means++**. Nous allons voir une technique d'initialisation qui permet d'améliorer la performance du clustering.
4. **Applications**. Nous allons mettre à l'épreuve l'algorithme en Python sur des cas simples comme sur des cas plus complexes. Nous pourrons ainsi également confronter les résultats aux faits établis dans les parties qui précèdent.

2 Garanties statistiques des K-means

2.1 Problème de quantification

Le terme « quantification » provient de la théorie du traitement du signal. Il a été utilisé par les ingénieurs électriciens à partir de la fin des années 40. Dans ce contexte, la quantification signifie un processus de discrétisation des signaux. L'application la plus courante de la quantification est la conversion analogique-numérique mais elle doit le développement de sa théorie aux problèmes de quantification pour la compression de signaux audio ou image. En tant que sujet mathématique, la quantification des distributions de probabilité concerne la meilleure approximation d'une distribution de probabilité d-dimensionnelle μ par une probabilité discrète supportée sur un ensemble de données $y_1, \ldots, y_K$ avec $K \geq 1$. En d'autres termes, on parle de la meilleure approximation d'un vecteur aléatoire X de dimension d avec une distribution μ par un vecteur aléatoire Y avec au plus K valeurs dans son image. En fait, il s'avère qu'il y a toujours une meilleure approximation de la forme $f(X)$, qu'on appelle "version quantifiée de X". Le problème de quantification peut être reformulé comme un problème de partition de l'espace sous-jacent. Nous allons en premier temps formaliser le problème mathématiquement en faisant le lien avec la méthode K-means, puis dans un second temps nous allons présenter les garanties statistiques de cette méthode.

2.1.1 Formulation du problème

Soit $(\Omega, \mathcal{F}, \mathbb{P})$ un espace probabilisé. On note $B(\mathbb{R}^d)$ la tribu des boréliens sur $\mathbb{R}^d$. Soit $X : (\Omega, \mathcal{F}, \mathbb{P}) \to (\mathbb{R}^d, B(\mathbb{R}^d))$ une variable aléatoire de loi μ. Dans la suite, on supposera alors toujours que μ est une mesure de probabilité sur $\mathbb{R}^d$ qui vérifie $\boxed{\int_{\mathbb{R}^d} \|\xi\|^2 \mu(d\xi) < \infty}$. Nous noterons $\boxed{\mathcal{P}_2(\mathbb{R}^d)}$ l'ensemble des mesures de probalités qui satisfont ce critère . On appelle cet ensemble l'*espace de Wasserstein*. De plus, pour toute mesure ν nous désignerons son support par $supp(\nu)$.

Soit $\mathcal{F}_K$ l'ensemble définit par $\mathcal{F}_K = \{f : \mathbb{R}^d \to \mathbb{R}^d \text{ boréliennes}, |f(\mathbb{R}^d)| \leq K\}$. Pour chaque $f \in \mathcal{F}_K$, $f(X)$ donne une version quantifiée de X. L'objectif est alors de trouver la meilleure, c'est-à-dire celle qui minimise l'erreur suivante.

Définition 2.1.1 (Erreur et problème de quantification)**.** *Soit $K \in \mathbb{N}^*$. L'**erreur de quantification de niveau K** est donnée pour X par :*

$$\forall f \in \mathcal{F}_K,\ V_K(\mu, f) = \mathbb{E}_\mu[\|X - f(X)\|^2]$$

*Ainsi, on appelle **problème de quantification de niveau K** le problème d'optimisation sous contrainte suivant :*

$$\inf_{f \in \mathcal{F}_K} \mathbb{E}_\mu[\|X - f(X)\|^2]$$

$V_K(\mu) = \inf_{f \in \mathcal{F}_K} \mathbb{E}_\mu[\|X - f(X)\|^2]$ *est alors appelée **erreur de quantification optimale de niveau K**. On pourra adopter la notation $V_K(f, X)$ à la place de $V_K(f, \mu)$, et $V_K(X)$ à la place de $V_K(\mu)$ selon le contexte. Enfin, on appelle **quantificateur optimal de niveau K** toute solution f pourvu que l'existence soit vérifiée.*

Le problème de quantification consiste à trouver un quantificateur optimal f s'il en existe. Nous montrerons dans la section 2.1.3 que pour tout K dans $\mathbb{N}^*$, le problème de quantification de niveau K admet une solution sous certaines conditions. Nous verrons donc que l'existence est garantie, même si en revanche l'unicité de la solution ne l'est pas en général.

A présent, fixons un niveau de quantification K dans $\mathbb{N}^*$ pour le reste de l'étude. Dans ce cadre, nous allons pouvoir introduire le problème de quantification pour les lois de probabilités sur $\mathbb{R}^d$. En effet, quantifier la variable aléatoire X équivaut à quantifier sa distribution, c'est-à-dire approcher μ par une probabilité discrète $\hat{\mu}^K$ avec K points de support, donc de la forme :

$$\hat{\mu}^K = \sum_{k=1}^{K} p_k \delta_{y_k}$$

où $y_1, \ldots, y_K$ est la discrétisation, $p_1, \ldots, p_K$ les poids qu'on associe à chaque point, et pour tout a de $\mathbb{R}^d$, δ_a est la mesure de Dirac au point a.

Définition 2.1.2 (Grille de quantification)**.** *On appelle tout K-uplet $y = (y_1, \ldots, y_K) \in (\mathbb{R}^d)^K$ une **grille de quantification**.*

*De plus, on appelle **grille sans répétition de y** la grille $\widetilde{y} = (\widetilde{y}_1, \ldots, \widetilde{y}_l)$ avec $l \leq K$ tel que :*

$$\forall i \neq j \ \widetilde{y}_i \neq \widetilde{y}_j \text{ et } \forall i \in \{1, \ldots, l\} \ \exists j \in \{1, \ldots, K\} \ \widetilde{y}_i = y_j \ .$$

Nous verrons dans la suite qu'il suffit, pour la quantification, de trouver une grille optimale. En effet, étant donné une grille sans répétition $y = (y_1, \ldots, y_l)$, les poids optimaux sont déterminés par les cellules $C_1(y), \ldots, C_l(y)$ où $(C_k(y))_{k=1,..,l}$ est ce qu'on appellera partition de Voronoï.

2.1.2 Masses optimales : partitions de Voronoï

Les partitions de Voronoi sur $\mathbb{R}^d$ jouent un rôle central sur le choix des partitions optimales de quantification pour des lois de probabilités sur $\mathbb{R}^d$. Dans cette partie nous allons introduire les partitions de Voronoï par rapport à une grille. Dans la suite, on note $\|.\|$ la norme 2 de $\mathbb{R}^d$, et $B(a, r) = \{x \in \mathbb{R}^d : \|x - a\| \leq r\}$ la boule fermée de centre a et de rayon $r > 0$ pour la norme $\|.\|$.

Définition 2.1.3 (Cellule de Voronoï)**.** *Soit $y = (y_1, \ldots, y_K) \in (\mathbb{R}^d)^K$ une grille de quantification. La **région de Voronoï** générée par $y_k \in \mathbb{R}^d$ est donnée par $R(y_k) = \{x \in \mathbb{R}^d, \|y - y_k\| \leq \min_{1 \leq j \leq K, j \neq k} \|x - y_j\|\}$.*

Proposition 2.1.1. *$\{R(y_k), 1 \leq k \leq K\}$ est alors un recouvrement de $\mathbb{R}^d$.*

Preuve. Soit $x \in \mathbb{R}^d$. Alors il existe y_k t.q $\|x - y_k\| \leq \|x - y_j\|$ pour tout $1 \leq j \leq K$ donc $y \in R(y_k)$. Donc

$$\bigcup_{k=1}^{K} R(y_k) = \mathbb{R}^d$$

□

Définition 2.1.4 (Cellule et partition de Voronoï)**.** *Soit $y = (y_1, \ldots, y_K) \in (\mathbb{R}^d)^K$ une grille de quantification avec $y_i \neq y_j$ pour tout $i \neq j$. On appelle alors **partition de Voronoï** générée par y la partition mesurable $(C_k(y))_{1 \leq k \leq K}$, telle que $\forall k : C_k(y) \subset \{x \in \mathbb{R}^d, \|x - y_k\| \leq \min_{1 \leq j \leq K, j \neq k} \|x - y_j\|\}$. Chaque sous-ensemble $C_k(y)$ de cette partition est appelée **cellule de Voronoï**.*

Exemple 2.1 (Partition de Voronoï)**.**

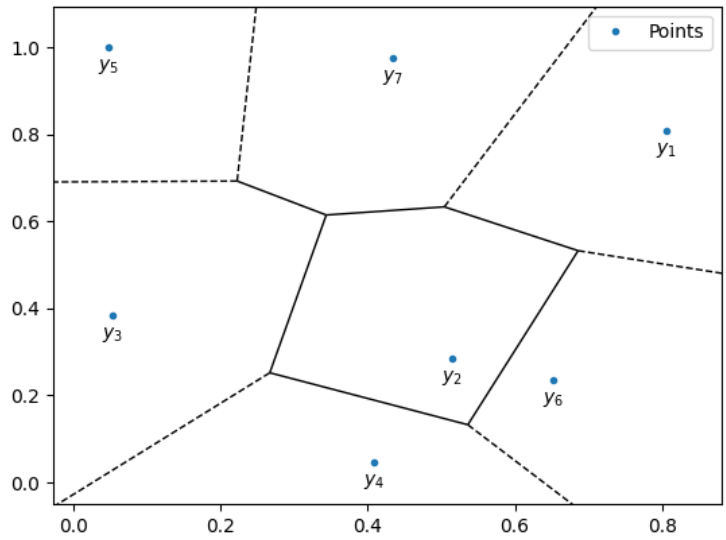

FIGURE 1 – Exemple d'une partition de Voronoï sur $\mathbb{R}^2$

La partition de Voronoï n'est pas unique. Prenons $y = (0, 2, 4) \in \mathbb{R}^3$. Deux partitions générées par y sont :

$C_1(y) =]-\infty, 1]$, $C_2(y) =]1, 3]$, $C_3(y) =]3, +\infty[$

ou

$C'_1(y) =]-\infty, 1[$, $C'_2(y) = [1, 3]$, $C'_3(y) =]3, +\infty[$

Ainsi, l'idée de la quantification de niveau K est, pour une grille y, d'approcher μ par $\hat{\mu}^K$ la mesure discrète donnée par :

$$\hat{\mu}^K = \sum_{k=1}^{K} \mu(C_k(y))\delta_{y_k}$$

où $(C_k(y))$ est une partition de Voronoï et $\mu(C_k(y))$ le poids de y_k (ou de la cellule de Voronoï $C_k(y)$). Formellement, ce choix de poids est optimal pour y. En effet, pour tout $y = (y_1, \dots, y_K)$, l'application :

$$\begin{aligned} f_y \colon \mathbb{R}^d &\longrightarrow \{y_1, \dots, y_K\} \\ \xi &\longmapsto \sum_{k=1}^{K} y_k 1_{C_k(y)}(\xi) \end{aligned}$$

est une fonction de projection sur $y_1, \dots, y_K$. De plus si $X \sim \mu$, alors $\mathbb{P}(f_y(X) = x_i) = \mu(C_i(y)) = \hat{\mu}^K(\{y_i\})$ pour tout $i \in \{1, \dots, K\}$ donc $f_y(X) \overset{\mathcal{L}}{\sim} \hat{\mu}^K$. Autrement dit la loi de la projection de X sur $y_1, \dots, y_K$ est $\hat{\mu}^K$. Ainsi, $\hat{\mu}^K$ telle qu'on l'a définie dans cette partie est une mesure à support $\{y_1, \dots, y_K\}$ qui projette μ sur l'ensemble des mesures à support de taille K.

2.1.3 Quantification optimale

Cette partie s'inspire du livre de Siegfried Graf [GL07] sur le problème de la quantification. Nous avons vu que résoudre le problème de quantification revient à trouver une fonction dans l'espace $\mathcal{F}_K$ qui minimise l'erreur. Or, un problème de minimisation sur un espace de fonctions

n'est en général pas facile à résoudre. La proposition qui suit reformule le problème, et nous amène à la notion de grille optimale $(y_1^*, \dots, y_K^*)$ de $\mathbb{R}^d$.

Proposition 2.1.2. *Soit $\mu \in \mathcal{P}_2(\mathbb{R}^d)$. Alors*

$$V_K(\mu) = \inf_{y=(y_1,\dots,y_K)} \mathbb{E}_\mu[\min_{i=1\dots K} \|X - y_i\|^2]$$

Démonstration. Soient $\mu \in \mathcal{P}_2(\mathbb{R}^d)$, $f \in \mathcal{F}_K$ et $\{y_1, \dots, y_K\} = f(\mathbb{R}^d)$. Alors

$$\begin{aligned}
\mathbb{E}_\mu[\|X - f(X)\|^2] &= \sum_{k=1}^K \int_{\{f=y_k\}} \|x - y_k\|^2 \mu(dx) \\
&\geq \sum_{k=1}^K \int_{\{f=y_k\}} \min_{1\leq i\leq K} \|x - y_i\|^2 \mu(dx) \\
&= \mathbb{E}_\mu[\min_{1\leq i\leq K} \|X - y_i\|^2] \ .
\end{aligned}$$

Réciproquement, soit $y = (y_1, \dots, y_K) \in (\mathbb{R}^d)^K$. On considère $\widetilde{y} = (\widetilde{y}_1, \dots, \widetilde{y}_l)$ avec $1 \leq l \leq K$ la grille sans répétition de y, $(C_k(\widetilde{y}))_{i\leq k\leq l}$ une partition de Voronoï par rapport à $\widetilde{y}$, et $f = \sum_{1\leq k\leq l} \widetilde{y}_k 1_{C_k(\widetilde{y})}$. On a bien $|f(\mathbb{R}^d)| \leq K$ car $f(\mathbb{R}^d) = \{\widetilde{y}_1, \dots, \widetilde{y}_l\}$, ainsi $f \in \mathcal{F}_K$. Ensuite,

$$\mathbb{E}_\mu[\min_{1\leq i\leq K} \|X - y_i\|^2] = \sum_{k=1}^l \int_{C_k(\widetilde{y})} \|x - \widetilde{y}_k\|^2 \mu(dx) = \mathbb{E}_\mu[\|X - f(X)\|^2] \ .$$

On en déduit que

$$\inf_{f\in\mathcal{F}_K} \mathbb{E}_\mu[\|X - f(X)\|^2] = \inf_{y=(y_1,\dots,y_K)} \mathbb{E}_\mu[\min_{i=1\dots K} \|X - y_i\|^2] \ .$$

□

Le problème ne nous impose plus de minimiser sur un espace fonctionnelle, mais de trouver un K-uplet optimal, autrement dit une grille de quantification optimale. De cette manière il nous permettra de faire le lien avec l'algorithme des K-means qui est en faite une façon de résoudre le problème de quantification. De plus, cette écriture nous permettra de démontrer l'existence d'un K-quantificateur optimal et donc le théorème d'existence. Mais avant, quelques lemmes sont nécessaires.

Lemme 2.1.1. *La fonction*

$$\begin{aligned}
\psi_K : (\mathbb{R}^d)^K &\longrightarrow \mathbb{R}_+ \\
(y_1, \dots, y_K) &\longmapsto \mathbb{E}_\mu[\min_{i=1\dots K} \|X - y_i\|^2]
\end{aligned}$$

est continue.

Démonstration. La fonction $(y_1, \dots, y_K) \longmapsto \min_{i=1\dots K} \|x - y_i\|^2$ est continue pour tout $x \in \mathbb{R}^d$. Soit $(y^n)_{n\in\mathbb{N}}$ une suite de $(\mathbb{R}^d)^K$ qui converge vers $y \in (\mathbb{R}^d)^K$. Montrons que $\psi_K(y^n) \xrightarrow[n\to+\infty]{} \psi_K(y)$.

Pour tout y dans $(\mathbb{R}^d)^K$ posons $\phi_K(y) = \min_{i=1\dots K} \|X - y_i\|^2$. D'après ce qui précède on a $\phi_K(y^n) \xrightarrow[n\to+\infty]{} \phi_K(y)$ p.s. Or $\phi_K(y) \leq Y$ presque sûrement où $Y = \|X - y_1\|^2$ est intégrable

par rapport à μ. Donc d'après le théorème de convergence dominée de Lebesgue on a bien le résultat.

□

Lemme 2.1.2. *Supposons que μ soit à support compact. Alors pour tout $R > 0$ tel que $supp(\mu) \subset B(0,R)$ on a :*

$$\inf_{y=(y_1,\dots,y_K)} \psi_K(y) = \inf_{\substack{y=(y_1,\dots,y_K) \\ \|y_i\| \leq R \, \forall i \in [\![1,K]\!]}} \psi_K(y) \ .$$

Démonstration. Soient $y = (y_1, \dots, y_K) \in (\mathbb{R}^d)^K$, et $R > 0$ tel que $supp(\mu) \subset B(0,R)$. Montrons l'assertion suivante :

$$\|y_1\| > R \implies \psi_K(y) > \inf_{\substack{z=(z_1,\dots,z_K) \\ \|z_1\| \leq R}} \psi_K(z) \ .$$

Supposons que $\|y_1\| > R$. Comme $X \in B(0,R)$ par hypothèse sur μ et $y_1 \notin B(0,R)$, d'après le théorème de projection sur un convexe fermé on peut trouver $x_1 \in B(0,R)$ tel que

$$\|X - y_1\| \geq \|X - x_1\| \ .$$

Donc $\min(\|X-y_1\|^2, \|X-y_2\|^2, \dots, \|X-y_K\|^2) \geq \min(\|X-x_1\|^2, \|X-y_2\|^2, \dots, \|X-y_K\|^2)$ et par croissance de l'espérence,

$$\psi_K(y_1, y_2, \dots, y_K) \geq \psi_K(x_1, y_2, \dots, y_K) \ .$$

Donc l'assertion est vraie.

On raisonne de la même manière pour chacun des y_i où $i > 1$, et on déduit qu'on peut restreindre le problème à la contrainte $\{y = (y_1, \dots, y_K) \in (\mathbb{R}^d)^K, \ \|y_i\| \leq R \quad \forall 1 \leq i \leq K\}$.

□

On adopte à présent la notation $A_R = \{y = (y_1, \dots, y_K) \in (\mathbb{R}^d)^K, \ \|y_i\| \leq R \quad \forall 1 \leq i \leq K\}$.

Théorème 2.1. *Supposons que μ soit à support compact. Alors le problème suivant :*

$$\inf_{f \in \mathcal{F}_K} E_\mu[\|X - f(X)\|^2]$$

admet une solution.

Démonstration. Soit $R > 0$ tel que $supp(\mu) \subset B(0,R)$.

1. ψ_K est continue d'après le Lemme 2.1.1.
2. A_R est compact non vide en tant que produit de compacts non vides.

D'après le Lemme 2.1.2, la Proposition 2.1.2, et le théorème de Weierstrass des bornes atteintes l'existence d'une solution est vérifiée. □

Le théorème nous donne une garantie sur l'existence d'un quantificateur pourvu que μ soit à support compact. Cela dit, l'existence d'une solution est en faite vérifiée sans cette contrainte sur μ (voir [Pol82] [GL07]). En revanche, l'unicité n'est pas toujours vraie. Pour mettre en évidence cela, nous introduisons la notion de similitude. En géométrie euclidienne, une similitude est une transformation qui multiplie toutes les distances par une constante fixe, appelée son scalaire ou son rapport. L'image de toute figure par une telle application est une figure semblable, c'est-à-dire intuitivement « de même forme ». [Wik23b].

Définition 2.1.5 (Similitude). *Soit $T : \mathbb{R}^d \to \mathbb{R}^d$ une application bijective. On dit que la transformation T est une similitude de scalaire ou rapport $\alpha > 0$ si pour tout $x, y \in \mathbb{R}^d$, $\|T(x) - T(y)\| = \alpha\|x - y\|$. De plus, on dit que deux éléments (resp. ensembles) x et y (resp. A et B) de $\mathbb{R}^d$ sont similaires si il existe une similitude T telle que $x = T(y)$ (resp. $A = T(B)$).*

Exemple 2.2 (Similitudes). *Il existe des exemples triviaux de similitudes.*

- *Toute translation $T : \mathbb{R}^d \to \mathbb{R}^d, x \mapsto x + b$ où $b \in \mathbb{R}^d$ est une similitude de scalaire 1. En particulier, l'identité est une similitude.*
- *Toute homothétie $T : \mathbb{R}^d \to \mathbb{R}^d, x \mapsto ax$ où $a \in \mathbb{R}^*$ est une similitude de scalaire $|a|$.*
- *Toute rotation d'angle non nul est une similitude de scalaire 1.*

Plus précisément, si deux objets sont similaires, l'un peut être obtenu à partir de l'autre par mise à l'échelle uniforme (agrandissement ou réduction), éventuellement avec translation, rotation et réflexion supplémentaires. Cela signifie que chaque objet peut être redimensionné, repositionné et réfléchi, de manière à coïncider précisément avec l'autre objet.

Proposition 2.1.3. *Soit T une similitude de scalaire $\alpha > 0$. Alors si $y^* = (y_1^*, \dots, y_K^*)$ est un quantificateur optimal de X de niveau K, $(T(y_1^*), \dots, T(y_K^*))$ est un quantificateur optimal de $T(X)$ de niveau K. De plus, $V_K(T(X)) = \alpha^2 V_K(X)$.*

Démonstration. Soient T une similitude de scalaire $\alpha > 0$ et y^* une grille optimale du problème de quantification de X. Alors pour tout $z = (z_1, \dots, z_K) \in (\mathbb{R}^d)^K$,

$$\begin{aligned} E_\mu[\min_{i=1\dots K} \|T(X) - T(y_i^*)\|^2] &= \alpha^2 E_\mu[\min_{i=1\dots K} \|X - y_i^*\|^2] \\ &\leq \alpha^2 E_\mu[\min_{i=1\dots K} \|X - z_i\|^2] \\ &= E_\mu[\min_{i=1\dots K} \|T(X) - T(z_i)\|^2] \ . \end{aligned}$$

Or T est bijective, donc comme l'inégalité est vrai pour tout z on a

$$E_\mu[\min_{i=1\dots K} \|T(X) - T(y_i^*)\|^2] \leq E_\mu[\min_{i=1\dots K} \|T(X) - y_i\|^2]$$

pour tout $y \in (\mathbb{R}^d)^K$. On en déduit que $(T(y_1^*), \dots, T(y_K^*))$ est une grille de quantification optimale de $T(X)$ de niveau K. Enfin, l'égalité $V_K(T(X)) = \alpha^2 V_K(X)$ est trivial d'après la proposition 2.1.2 et le fait que $T(\mathbb{R}^d) = \mathbb{R}^d$ par bijectivité. □

Voyons maintenant des exemples de non unité du minimiseur. On considère alors $K \geq 2$.

Exemple 2.3 (Non unicité du minimiseur par permutation). *Si on se fixe un niveau $K = 2$, alors la grille (y_1, y_2) est optimale si et seulement si (y_2, y_1) est optimale. Plus généralement, pour un niveau $K \geq 2$ toute permutation d'une grille optimale est une grille optimale. Cela signifie que dès que au moins deux composantes d'une grille optimale sont disjointes, l'unicité n'est pas vérifiée.*

Exemple 2.4 (Non unicité du minimiseur sans permutation). *On se base sur le résultat suivant. Pour toutes variables aléatoires X et Y dans $\mathbb{R}^d$, l'égalité en loi implique immédiatement que la quantification de X est équivalente à celle de Y, au sens où les solutions sont égales et donnent la même valeur pour l'erreur. En effet, pour toute fonction $f \in \mathcal{F}_K$, $V_K(X, f) = V_K(Y, f)$. Ainsi, les variables aléatoires qui possède certaines propriétés d'invariance en loi sont des exemples adéquats pour illustrer la non unicité. Prenons par exemple X qui suit une loi μ uniforme sur*

$[-2,-1]\cup[1,2]$. *On se donne comme niveau de quantification* $K=3$. *Alors* $y=(-\frac{3}{2},\frac{5}{4},\frac{7}{4})$ *est une grille optimale. Ainsi, si on considère la similitude* $T:x\mapsto -x$, *on a que* $T(X)\overset{\mathcal{L}}{=}X$ *et donc que* $-y$ *est aussi optimale pour la quantification de* X *(car elle l'est pour* $T(X)$*). Par conséquent, l'ensemble des solutions contient* $\left\{(-\frac{7}{4},-\frac{5}{4},\frac{3}{2}),(-\frac{3}{2},\frac{5}{4},\frac{7}{4})\right\}$. *On vérifie numériquemen ce résultat dans la section 5.3.*

Dans la même idée, pour une loi normale standard $\mathcal{N}(0,I_d)$ *toute rotation d'une grille optimale est une grille optimale. D'une manière générale, pour une loi de probabilité de support invariant par une isométrie* T *(similitude de scalaire* 1*), il est facile de trouver plusieurs quantificateurs optimaux distincts une fois qu'une solution est exhibée. Ce fait est d'autant plus vrai lorsque* $d\geq 2$.

Cependant, pour $d=1$, il a été prouvé que dès que μ est absolument continue avec une densité log-concave, il existe exactement une grille de quantification optimale au niveau K. Cette grille est de taille K, c'est-à-dire sans répétition (voir [Kie83]).

2.1.4 Quantification vers K-Means

Dans cette partie, nous voyons le lien entre la méthode K-means et le problème de quantification. Pour établir ce lien, introduisons d'abord l'objet suivant.

Définition 2.1.6 (Mesure empirique)**.** *Soit* $X=(X_1,\dots,X_n)$ *un* n*-échantillon iid de loi* μ. *On appelle mesure empirique la mesure aléatoire* μ_n *définie par*

$$\mu_n=\frac{1}{n}\sum_{i=1}^{n}\delta_{X_i}\ .$$

La motivation de l'étude des mesures empiriques est qu'étant donné un échantillon de variables aléatoires *iid*, la loi sous-jacente est souvent inconnue. Par conséquent, on collecte les données et on calcule leur fréquence. Un autre moyen d'estimer une loi est de passer par la fonction de répartition empirique. Regardons rapidement des propriétés de consistance de l'estimateur $\mu_n(A)$ pour tout $n\in\mathbb{N}^*$ où A est un ensemble $B(\mathbb{R}^d)$-mesurable fixé.

Proposition 2.1.4. *Pour tout ensemble mesurable* A, $\mu_n(A)$ *est un estimateur sans biais et fortement consistant de* $\mu(A)$.

Démonstration. Soit $A\in B(\mathbb{R}^d)$ fixé et $n\in\mathbb{N}^*$. Alors $\mu_n(A)=\frac{1}{n}\sum\limits_{i=1}^{n}\delta_{X_i}(A)=\frac{1}{n}\sum\limits_{i=1}^{n}1_{\{X_i\in A\}}$. La suite $(1_{\{X_i\in A\}})_{1\leq i\leq n}$ est une suite de variables *iid* et intégrables. Par conséquent, $\mathbb{E}[\mu_n(A)]=\mathbb{E}[\delta_{X_1}(A)]=\mu(A)$. De plus, d'après la loi forte des grands nombres on a bien $\mu_n(A)\to\mu(A)$ presque sûrement lorsque $n\to+\infty$. □

Remarque 2.1. *En particulier, si* $(C_k(y))_{1\leq i\leq K}$ *est une partition de Voronoï générée par une grille* y, *alors pour tout* $k\in\{1,\dots,K\}$, $\mu_n(C_k(y))=\frac{|X_i\in C_k(y)|}{n}$ *est un estimateur consistant de* $\mu(C_k(y))$

Ces propriétés nous rassure sur la pertinence d'un tel estimateur pour la loi μ. Néanmoins le fait que nous ayons fixé un ensemble mesurable rend ces propriétés relativement « faibles ». Nous verrons en fait un résultat plus fort dans la section 2.2.3.

Soit $X=(X_1,\dots,X_n)$ un n-échantillon *iid* de loi μ et $\mu_n=\frac{1}{n}\sum\limits_{i=1}^{n}\delta_{X_i}$ la mesure empirique de μ. Alors le K-Means clustering consiste à se donner un entier naturel non-nul K

qu'on appelle *nombre de clusters*, et à quantifier la mesure μ_n i.e chercher une K-grille optimale $y^* = (y_1^*, \dots, y_K^*)$, où pour $i \in \{1, \dots, K\}$ les y_i^* sont appelés centroïdes, qui vérifient

$$V_K(\mu_n) = \mathbb{E}_{\mu_n}[\min_{i=1\dots K} \|X - y_i^*\|^2].$$

Le problème consiste donc à minimiser l'erreur de quantification pour la mesure empirique μ_n dont la formule est donnée par :

$$\begin{aligned}
\mathbb{E}_{\mu_n}[\min_{i=1\dots K} \|X - y_i\|^2] &= \int \min_{i=1\dots K} \|\xi - y_i\|^2 \mu_n(d\xi) \\
&= \sum_{k=1}^{K} \int_{C_k(y)} \min_{i=1\dots K} \|\xi - y_i\|^2 \mu_n(d\xi) \\
&= \sum_{k=1}^{K} \int_{C_k(y)} \|\xi - y_k\|^2 \mu_n(d\xi) \\
&= \frac{1}{n} \sum_{k=1}^{K} \sum_{i=1}^{n} \int_{C_k(y)} \|\xi - y_k\|^2 \delta_{X_i}(d\xi) \\
&= \sum_{k=1}^{K} \frac{1}{n} \sum_{i=1}^{n} 1_{C_k(y)}(X_i) \|X_i - y_k\|^2 \\
&= \sum_{k=1}^{K} \frac{1}{n} \sum_{X_i \in C_k(z)}^{n} \|X_i - y_k\|^2 \ .
\end{aligned}$$

On se ramène ainsi à la minimisation de la somme de carrés d'erreurs aux centroïdes, ce qui est le principe de la méthode K-means. L'erreur global, dans le cadre des K-means, est appelée la *variance* (intra-classe). Multipliée par n, on l'appelle l'inertie. À la fin, les cellules de Voronoï associés à la grille optimale forment une partition de l'espace.

2.2 Consistance de K-Means : convergence de l'erreur et des centroïdes optimaux

Fixons un niveau de quantification $K \geq 1$. Le but de cette partie sera d'exhiber des propriétés de consistance de type

$$\mu_n \xrightarrow[n\to+\infty]{} \mu \implies V_K(\mu_n) \xrightarrow[n\to+\infty]{} V_K(\mu)$$

où (μ_n) est une suite de mesures et μ est la loi de X. De plus, notons $Q_K(\mu)$ une mesure de quantification optimale que l'on supposera unique pour μ, et $Q_K(\mu_n)$ une mesure optimale pour μ_n. Après avoir donné une caractérisation claire de cette notion, nous allons montrer :

$$\mu_n \xrightarrow[n\to+\infty]{} \mu \implies Q_K(\mu_n) \xrightarrow[n\to+\infty]{} Q_K(\mu)$$

Une première étape sera de décrire la métrique que nous allons utiliser pour la convergence. Ensuite, nous allons démontrer les propriétés de convergence.

2.2.1 Distance de Wasserstein

La distance de Wasserstein est une mesure de distance entre deux distributions de probabilité, également appelée distance de Kantorovich-Rubinstein ou distance de transport optimal. Elle est définie comme suit :

Définition 2.2.1. *Soient μ et ν deux mesures de probabilité sur $\mathcal{P}_2(\mathbb{R}^d)$. La distance de Wasserstein (quadratique) entre μ et ν est donnée par :*

$$W_2(\mu,\nu) = \left(\inf_{\gamma\in\Gamma(\mu,\nu)} \int_{X\times Y} \|x-y\|^2 d\gamma(x,y)\right)^{1/2}$$

où $\Gamma(\mu,\nu) \subset \mathcal{P}_2(\mathbb{R}^d \times \mathbb{R}^d)$ est l'ensemble des couplages de μ et ν, c'est-à-dire l'ensemble des mesures de probabilité γ sur $\mathbb{R}^d \times \mathbb{R}^d$ dont les mesures marginales sont respectivement μ et ν.

D'une manière équivalente, on peut définir cette distance de la manière suivante :

$$W_2(\mu,\nu) = \left(\inf_{X\sim\mu, Y\sim\nu} \mathbb{E}[\|X-Y\|^2]\right)^{1/2}.$$

Ainsi, cette distance définit une métrique sur $\mathcal{P}_2(\mathbb{R}^d)$. C'est donc celle-ci qui sera adoptée pour étudier les propriétés de convergence de mesures.

2.2.2 Mesure optimale de quantification

Pour K fixé, nous noterons D_K l'ensemble des probabilités discrètes $\nu \in \mathcal{P}_2(\mathbb{R}^d)$ supportées sur au plus K points.

Lemme 2.2.1. *Soit $\mu \in \mathcal{P}_2(\mathbb{R}^d)$. Alors*

$$V_K(\mu) = \inf_{\nu\in D_K} W_2^2(\mu,\nu)$$

Démonstration. Soit $f \in \mathcal{F}_K$. Soient μ_f la mesure image de μ par f et μ_ϕ la mesure image de μ par $\phi : \mathbb{R}^d \to \mathbb{R}^d \times \mathbb{R}^d$, $x \mapsto (x, f(x))$. On pose $d^2 : (x,y) \mapsto \|x-y\|^2$. Ainsi, comme

$$\mathbb{E}[\|X - f(X)\|^2] = \int d^2(x, f(x)) d\mu(x)$$

et que par définition de μ_ϕ on a

$$\int d^2 \circ \phi \; d\mu = \int d^2 \; d\mu_\phi$$

alors

$$\mathbb{E}[\|X - f(X)\|^2] = \int \|x-y\|^2 d\mu_\phi(x,y) \; .$$

De plus, $\mu_\phi \in \Gamma(\mu, \mu_f)$. En effet, soient $A \in \mathcal{B}(\mathbb{R}^d)$ et $B \in \mathcal{B}(\mathbb{R}^d)$. Alors :

- $\mu_\phi(A \times \mathbb{R}^d) = \mu(\{x \in \mathbb{R}^d, (x, f(x)) \in A \times \mathbb{R}^d\}) = \mu(\{x \in \mathbb{R}^d, x \in A\}) = \mu(A)$.
- $\mu_\phi(\mathbb{R}^d \times B) = \mu(\{x \in \mathbb{R}^d, (x, f(x)) \in \mathbb{R}^d \times B\}) = \mu(\{x \in \mathbb{R}^d, f(x) \in B\}) = \mu_f(\{x \in \mathbb{R}^d, x \in B\}) = \mu_f(B)$.

Donc μ_f et μ sont bien deux marginales de μ_ϕ, et finalement

$$\mathbb{E}[\|X - f(X)\|^2] = \int \|x-y\| d\mu_\phi(x,y) \geq W_2^2(\mu, \mu_f)$$

ce qui implique

$$V_K(\mu) \geq \inf_{f\in\mathcal{F}_K} W_2^2(\mu,\mu_f) \geq \inf_{\nu\in D_K} W_2^2(\mu,\nu) \; .$$

Soit $\nu \in D_K$ avec $\nu(\mathcal{X}) = 1$ où $\mathcal{X} = \{y_1, \ldots, y_K\}$ (donc $|\mathcal{X}| \leq K$). Alors pour toute mesure de probabilité γ sur $\mathbb{R}^d \times \mathbb{R}^d$ de marginales μ et ν :

$$\begin{aligned}\int \|x-y\|^2 d\gamma(x,y) &= \int_{\mathbb{R}^d \times \mathcal{X}} \|x-y\|^2 d\gamma(x,y) \\ &\geq \int_{\mathbb{R}^d \times \mathcal{X}} \min_{1\leq i\leq K} \|x-y_i\|^2 d\gamma(x,y) \\ &\geq \int_{\mathbb{R}^d} \min_{1\leq i\leq K} \|x-y_i\|^2 d\mu(x) \ .\end{aligned}$$

Par conséquent,

$$W_2^2(\mu,\nu) \geq \mathbb{E}[\min_{1\leq i\leq K} \|X-y_i\|^2] \ .$$

Et d'après la Proposition 2.1.2

$$\inf_{\nu\in D_K} W_2^2(\mu,\nu) \geq V_K(\mu) \ .$$

□

Définition 2.2.2. *Une mesure ν est appelée* ***mesure optimale de quantification*** *de μ si $W_2^2(\mu,\nu) = V_K(\mu)$.*

Dans la preuve, nous avons montré en particulier que :

$$V_K(\mu) = \inf_{f\in\mathcal{F}_K} W_2^2(\mu,\mu_f) = \inf_{\nu\in D_K} W_2^2(\mu,\nu).$$

Donc on a que f est une fonction de quantification optimale pour K fixé , alors μ_f est une mesure de quantification optimale. Réciproquement, si $\nu \in D_K$ est une mesure optimale et $(C_k(y))_{k=1\ldots K}$ est une partition de Voronoï par rapport à y où $\{y_1, \ldots, y_K\} = supp(\nu)$, alors la fonction de projection $f = \sum y_k 1_{C_k(y)}$ est une fonction de quantification optimale.

2.2.3 Propriétés de consistance

Nous étudierons dans cette partie la consistance en supposant l'**unicité du minimiseur** pour le problème de quantification de niveau K. D'après ce qui précède, on peut définir une mesure de probabilité optimale pour le problème. Ainsi, dans la suite on adoptera la notation $Q_K(\mu)$ pour la mesure dans D_K qui vérifie l'optimalité $V_K(\mu) = W_2^2(\mu, Q_K(\mu))$.

Lemme 2.2.2. *Soit (M,d) un espace métrique, et $N \subset M$ un ensemble non vide. Soit $f : N \to \mathbb{R}_+$ une fonction continue telle que les ensembles de niveau $L(c) = \{x \in N, f(x) \leq c\}$ pour $c > \inf_{z\in N} f(z)$ sont compacts. Supposons que f admette un unique minimiseur y^* sur N. Alors si $(y_k)_{k\geq 1}$ est une suite minimisante dans N pour f, c'est-à-dire $f(y_k) \to \inf_{z\in N} f(z)$, alors*

$$d(y_k, y^*) \to 0, \quad k \to +\infty$$

Démonstration. La suite $(y_k)_k$ est minimisante donc comme $\inf_{z\in N} f(z) < c$, il existe un rang à partir duquel $f(y_k) \leq c$ donc $y_k \in L(c)$. Par compacité de $L(c)$, on peut extraire une sous-suite convergente $(y_{k_n})_{n\geq 1}$. On en prend une et on note y sa limite. Alors par continuité de f et par unicité de la limite, $f(y) = \inf_{z\in N} f(z)$ donc $y = y^*$ par unicité du minimiseur. On peut à présent montrer que :

$$d(y_k, y^*) \to 0, \quad k \to +\infty$$

Raisonnons par l'absurde : supposons que $\limsup d(y_k, y^*) > 0$. Alors il existe $\epsilon > 0$ et une sous suite convergente $(y_{k_n})_{n\geq 1}$ de $(y_k)_{k\geq 1}$ tels que $d(y_{k_n}, y^*) > \epsilon$ pour tout $n \geq 1$. Or $y = y^*$ donc on a une contradiction.

□

Théorème 2.2 (Consistance)**.** *Soient $\mu \in \mathcal{P}_2(\mathbb{R}^d)$ et (μ_n) une suite de mesures. Supposons que $W_2(\mu_n, \mu) \xrightarrow[n\to+\infty]{} 0$. Alors*

$$V_K(\mu_n) \xrightarrow[n\to+\infty]{} V_K(\mu) \text{ et } Q_K(\mu_n) \xrightarrow[n\to+\infty]{W_2} Q_K(\mu) \ .$$

Démonstration. On se place dans l'espace métrique $(\mathcal{P}_2(\mathbb{R}^d), W_2)$. Supposons que $\mu_n \to \mu$. Considérons alors f la fonction définie par :

$$\begin{aligned} f\colon D_K &\longrightarrow \mathbb{R}_+ \\ \nu &\longmapsto W_2(\mu, \nu) \end{aligned}$$

La fonction f est continue sur D_K car W_2 est une distance. On admet que les sous-ensembles de niveau de f sont compacts. Montrons que

$$f(Q_K(\mu_n)) \xrightarrow[n\to+\infty]{} f(Q_K(\mu))$$

où

$$|f(Q_K(\mu_n)) - f(Q_K(\mu))| = |W_2(Q_K(\mu_n), \mu) - W_2(Q_K(\mu), \mu)| \ .$$

D'après l'inégalité triangulaire et par définition de $Q_K(\mu_n)$:

$$W_2(Q_K(\mu_n), \mu) \leq W_2(Q_K(\mu_n), \mu_n) + W_2(\mu_n, \mu) = V_K^{\frac{1}{2}}(\mu_n) + W_2(\mu_n, \mu) \ .$$

Or, d'après le Lemme 2.2.1 :

$$V_K^{\frac{1}{2}}(\mu_n) = \inf_{\nu\in D_K} W_2(\nu, \mu_n) \leq W_2(Q_K(\mu), \mu_n) \leq W_2(Q_K(\mu), \mu) + W_2(\mu_n, \mu) \ .$$

Donc finalement on a

$$0 \leq W_2(Q_K(\mu_n), \mu) - W_2(Q_K(\mu), \mu) \leq 2W_2(\mu_n, \mu) \ .$$

On en déduit que $f(Q_K(\mu_n)) \xrightarrow[n\to+\infty]{} f(Q_K(\mu))$ car μ_n tend vers μ. Et ainsi comme $(Q_K(\mu_n))_n$ est une suite minimisante pour f, en utilisant le Lemme 2.2.2 on peut conclure que :

$$Q_K(\mu_n) \xrightarrow[n\to+\infty]{W_2} Q_K(\mu)$$

Avec les mêmes arguments on obtient que $V_K(\mu_n) - V_K(\mu) \leq W_2(\mu_n, \mu)$ et par symétrie $|V_K(\mu_n) - V_K(\mu)| \leq W_2(\mu_n, \mu)$, donc

$$V_K(\mu_n) \xrightarrow[n\to+\infty]{} V_K(\mu)$$

□

Enfin, nous pouvons compléter ce théorème par le suivant.

Théorème 2.3 ("Loi forte des grands nombres" appliqué à la mesure empirique pour W_2)**.** *Soit* $(X_1, \ldots, X_n)$ *un vecteur de variables aléatoires iid de loi* μ*. Alors*

$$\mu_n = \frac{1}{n} \sum_{i=1}^{n} \delta_{X_i} \xrightarrow[n \to +\infty]{W_2} \mu$$

Démonstration. Théorème admis. □

Ce théorème nous indique que le théorème de consistance s'applique pour la quantification de la mesure empirique d'un échantillon de taille n i.i.d de loi μ. Jusqu'ici, nous avons établi trois résultats importants.

1. Il existe au moins une solution au problème de quantification.
2. Le problème K-means est consistant : Par le Théorème 2.3 l'erreur optimale sous la mesure empirique μ_n converge vers l'erreur optimale sous μ. On a également convergence pour les centroïdes ! Si l'ensemble des centroïdes optimaux pour μ est unique, les centroïdes optimaux de la mesure empirique convergent vers cet ensemble.
3. La méthode des K-means est un moyen de résoudre le problème de quantification.

Supposons maintenant qu'on ait n réalisations $x_1, \ldots, x_n$ tirées selon la loi μ, et qu'on veuille accéder numériquement au minimiseur $(y_1^*, ..., y_K^*)$. L'algorithme de Lloyd, également connu sous le nom d'algorithme de k-means, est une méthode numérique qui permet de résoudre ce problème.

3 Garanties de convergence de l'algorithme de Lloyd

La méthode de Lloyd a été conçue pour la première fois en 1957 par S.P. Lloyd, mais n'a été publiée qu'en 1982 dans un de ses ouvrages. À notre connaissance, c'était la première méthode consacrée au calcul numérique de quantificateurs. Elle a été redécouverte indépendamment par Max au début des années 1960. Aujourd'hui, cet algorithme est la méthode la plus connue pour résoudre le problème des K-means.

3.1 Description de l'algorithme et stationnarité

Dans cette section, nous allons étudier la convergence de cet algorithme. Soit $\mathcal{X} = \{x_1, \ldots, x_n\}$ un ensemble de données de n points dans $\mathbb{R}^d$ où $d \geq 1$. On cherche à les partitionner en K clusters $C_1, \ldots, C_K$ représentés par les centres optimaux $y^* = (y_1^*, y_2^*, ..., y_K^*)$. L'algorithme 1 montre comment fonctionne la méthode K-means de Lloyd.

A chaque itération, si y est un K-uplet, on cherche à réduire le coût

$$\phi_K(y) = \phi_K(y_1, \ldots, y_K) = \sum_{k=1}^{K} \frac{1}{n} \sum_{x_i \in C_k}^{n} \|x_i - y_k\|^2$$

où $C_1, \ldots, C_K$ sont les clusters associés aux centroïdes, en trouvant un nouveau K-uplet y_{new}. Dans cet algorithme, la distance entre deux points est souvent (voir toujours) mesurée à l'aide de la distance euclidienne $\|.\|$. En résumé, l'algorithme fonctionne en initialisant aléatoirement les centres des clusters, puis répète jusqu'à convergence l'affectation des données aux clusters les plus proches et la mise à jour des centres de clusters en calculant la moyenne de toutes les données assignées à chaque cluster.

Lemme 3.1.1. *L'algorithme de Lloyd converge en un nombre fini d'iterations.*

Algorithm 1 K-means clustering

Require: X : set of n data points, K : number of clusters
Ensure: C : set of K clusters
1: Randomly initialize K cluster centers $y_1, \ldots, y_K$
2: **repeat**
3: **for** $i = 1$ to n **do**
4: $j \leftarrow \arg\min_k ||x_i - y_k||^2$
5: Assign x_i to cluster $j : C_j$
6: **end for**
7: **for** $j = 1$ to K **do**
8: Update cluster center : $y_j = \frac{1}{|C_j|} \sum_{x_i \in C_j} x_i$ where C_j data set of jth cluster
9: **end for**
10: **until** Until $y_1, \ldots, y_K$ no longer change
11: Return clusters : C=(C_1^*,...,C_K^*) and cluster centers : y^*= (y_1^*,...,y_K^*)

Démonstration. Pour prouver la convergence de l'algorithme, nous allons montrer que la suite de la fonction de coût ϕ_K stationnaire. En effet, elle est décroissante et positive à chaque itération. De plus, la fonction coût ne peut prendre qu'un nombre fini de valeurs. On pourra alors déduire que la suite est constante à partir d'un certain rang.

Supposons que $\phi_K(y^{(t)})$ soit la valeur de la fonction de coût ϕ_K à l'itération $t \geq 1$ de l'algorithme. On cherche à montrer qu'à l'itération $t+1$, $\phi_K(y^{(t+1)})$ est plus petit.

Soit $j : \{1, \ldots, n\} \to \{1, \ldots, K\}$ la fonction qui à l'indice i associe l'indice du centroïde le plus proche. En d'autre termes, on a $j : i \mapsto \arg\min_{k \in 1,\ldots,K} \|x_i - y_k\|$. Alors on a

$$\begin{aligned}
\phi_K(y^{(t)}) &= \sum_{i=1}^{n} ||x_i - y_{j(i)}^{(t)}||^2 \\
&= \sum_{i=1}^{n} ||x_i - y_{j(i)}^{(t)} + y_{j(i)}^{(t+1)} - \frac{1}{|C_j|} \sum_{x_i \in C_j} x_i||^2 \\
&= \sum_{i=1}^{n} ||x_i - \frac{1}{|C_j|} \sum_{x_i \in C_j} x_i||^2 + \sum_{i=1}^{n} ||y_{j(i)}^{(t+1)} - y_{j(i)}^{(t)}||^2 \\
&+ 2 \sum_{i=1}^{n} ||x_i - \frac{1}{|C_j|} \sum_{x_i \in C_j} x_i|| \times ||y_{j(i)}^{(t+1)} - y_{j(i)}^{(t)}|| \\
&\geq \sum_{i=1}^{n} ||x_i - \frac{1}{|C_j|} \sum_{x_i \in C_j} x_i||^2 \\
&= \sum_{i=1}^{n} ||x_i - y_{j(i)}^{(t+1)}||^2 \\
&= \phi_K(y^{(t+1)}).
\end{aligned}$$

Dans la $t+1$ème itération de l'algorithme K-means, la distance de la somme des carrés des erreurs entre x_i et le centre de son cluster $y_{j(i)}^{(t+1)}$ est plus petite ou égale à celle de l'itération qui précède. Cette propriété est due au fait que lors de chaque itération, l'algorithme réaffecte chaque point de données au centre de cluster le plus proche, ce qui fait que le nouveau centre de cluster est plus proche des points de données, entraînant ainsi une diminution de la valeur de la

fonction objectif.

□

3.2 Algorithme de Lloyd comme méthode de Newton

Nous allons montrer que l'algorithme de Lloyd est en fait la méthode de Newton appliquée à une certaine fonction.

La méthode de Newton-Raphson est une méthode numérique d'optimisation pour trouver les racines (ou zéros) d'une fonction. Lorsqu'elle est appliquée à la dérivée d'une fonction convexe, la racine est la position de la valeur minimale. Elle est donc également très utile en optimisation. Cette méthode est basée sur l'approximation d'une fonction par une série de Taylor à deuxième ordre, qui est ensuite résolue pour trouver le minimum.

La descente de gradient repose sur une approximation du premier ordre de f : localement, $f(x_0 + h)$ ressemble à $f(x_0) + \langle \nabla f(x_0), h \rangle$, donc pour diminuer f, nous devons faire un pas dans la direction minimisant la quantité $\langle \nabla f(x_0), h \rangle$ (et cette direction est donnée par $-\nabla f(x_0)$). Si nous essayions d'écrire une approximation du second ordre de f autour de x_0, nous obtenous la méthode de Newton.

Nous avons

$$f(x_0 + h) \approx P_{f,x_0} = f(x_0) + \langle \nabla f(x_0), h \rangle + \frac{1}{2} h^T \nabla^2 f(x_0) h \tag{1}$$

où $\nabla^2 f$ est la Hessienne.

Trouvons le minimum de la fonction quadratique à droite P_{f,x_0}. Son gradient est égal à $\nabla f(x_0) + \nabla^2 f(x_0) h$. Par conséquent, si la Hessienne est inversible en x_0 (par exemple, si f est fortement convexe), alors le minimum de P_{f,x_0} est atteint pour $h = (\nabla^2 f(x_0))^{-1} \nabla f(x_0)$.

Définition 3.2.1. *Soit $f : \mathbb{R}^d \to \mathbb{R}$ une fonction convexe et deux fois differentiable, telle que la hessienne $\nabla^2 f$ de f est toujours inversible. Supposons que $x^{(0)}$ soit initialisé. Un pas de la méthode de Newton est donné par :*

$$x^{t+1} = x^t - \nabla^2 f(x^t)^{-1} \nabla f(x^t) \tag{2}$$

pour tout $t \in \mathbb{N}$.

Pour obtenir des garanties de convergence sur la méthode de Newton, nous aurons besoin de supposer que la matrice hessienne $\nabla^2 f$ soit lipschitz.

Théorème 3.1. *L'algorithme de Lloyd est équivalent à la méthode de Newton appliqué à la fonction $f_K(y_1, \ldots, y_K) = \frac{1}{n} \sum_{i=1}^{n} \min_{l \in 1,\ldots,K} ||y_l - x_i||^2 = \phi_K(y_1, \ldots, y_K)$.*

Démonstration. Le preuve de ce théorème est une réécriture de ce qu'on trouve ici : [Diva].

$$x^{t+1} = x^t - \nabla^2 f(x^t)^{-1} \nabla f(x^t)$$

où $\nabla^2 f(x^t)$ est la matrice hessienne de f en x^t et $\nabla f(x^t)$ est le gradient de f en x^t. La méthode de Newton nécessite que la matrice hessienne de f soit inversible en chaque point de l'itération.

Soit f_K la fonction :

$$f_K(y_1, \ldots, y_K) = \frac{1}{n} \sum_{i=1}^{n} \min_{l \in 1,\ldots,K} ||y_l - x_i||^2 \quad = \frac{1}{n} \sum_{l=1}^{K} \sum_{i \in C_l} ||y_l - x_i||^2$$

où C_l est l'ensemble des points du l-ème cluster.

Une itération de la méthode de Newton est donnée par :

$$y_{new} = y - \nabla^2 f_K(y_1, \ldots, y_K)^{-1} \nabla f_K(y_1, \ldots, y_K)$$

où $\nabla^2 f$ est la matrice hessienne et $y = (y_1, \ldots, y_K)$

Nous avons :

$$\begin{aligned}
\frac{\partial f_K(y_1, \ldots, y_K)}{\partial y_1} &= \sum_{i \in C_1} 2(y_1 - x_i) \\
&= \frac{2}{n} \times (n_1 \times y_1 - \sum_{i \in C_1} x_i) \\
&= \frac{2n1}{n}(y_1 - \frac{\sum_{i \in C_1} x_i}{n_1}) \\
&= \frac{2n1}{n}(y_1 - \tilde{y_1}).
\end{aligned}$$

où $\tilde{y_1}=\frac{\sum_{i \in C_1} xi}{n_1}$ et n_1 est le nombre de points du cluster 1.

Ainsi :

$$\nabla f_K(y_1, \ldots, y_K) = \begin{bmatrix} \frac{\partial f_K(y_1, \ldots, y_K)}{\partial y_1} \\ \frac{\partial f_K(y_1, \ldots, y_K)}{\partial y_2} \\ \vdots \\ \frac{\partial f_K(y_1, \ldots, y_K)}{\partial y_n} \end{bmatrix} = \frac{2}{n} \begin{bmatrix} n1(y_1 - \tilde{y_1}) \\ n2(y_2 - \tilde{y_2}) \\ \vdots \\ nk(y_K - \tilde{y_K}) \end{bmatrix}.$$

où n_1,n_2,...,n_K est le nombre d'échantillion dans les K clusters.

La matrice hessienne est égale à :

$$\nabla^2 f_K(y_1, \ldots, y_K) = \begin{bmatrix} \frac{\partial f_K}{\partial \mathbf{y}_1 \partial \mathbf{y}_1} & \frac{\partial f_K}{\partial \mathbf{y}_i \partial \mathbf{y}_2} & \cdots & \frac{\partial f_K}{\partial \mathbf{y}_i \partial \mathbf{y}_n} \\ \vdots & \vdots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \frac{\partial f_K}{\partial \mathbf{y}_K \partial \mathbf{y}_K} \end{bmatrix} = \frac{2}{n} \begin{bmatrix} n_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & n_K \end{bmatrix},$$

Donc

$$\nabla^2 f_K(y_1, \ldots, y_K)^{-1} = \frac{n}{2} \times \begin{bmatrix} \frac{1}{n1} & \cdots & & \\ \cdots & \frac{1}{n2} \cdots & & \\ \vdots & \vdots & \ddots & \frac{1}{nK} \end{bmatrix}.$$

Donc

$$\nabla^2 f_K(y_1, \ldots, y_K)^{-1} \nabla f_K(y_1, \ldots, y_K) = \begin{bmatrix} y_1 - \tilde{y_1} \\ \vdots \\ y_K - \tilde{y_K} \end{bmatrix}.$$

Ainsi la méthode de Newton donne les nouveaux centroïdes :

$$\begin{bmatrix} y_{new,1} \\ \vdots \\ y_{new,K} \end{bmatrix} = \begin{bmatrix} y_1 - (y_1 - \tilde{y_1}) \\ \vdots \\ y_K - (y_K - \tilde{y_K}) \end{bmatrix} = \begin{bmatrix} \tilde{y_1} \\ \vdots \\ \tilde{y_K} \end{bmatrix}.$$

□

Proposition 3.2.1 (Convergence vers le minimum [Divb]). *Soit $f : \mathbb{R}^d \to \mathbb{R}$ une fonction fortement convexe et deux fois différentiable avec la propriété que $\|\nabla^2 f(x)u - \nabla^2 f(y)u\| \leq \gamma\|x-y\|\|u\|$ pour tout $x, y, u \in \mathbb{R}^d$, par un certain $\gamma, \beta \in \mathbb{R}$, β représente la régularité dans x^*. Soit x^* est le minimum de f. Alors la méthode de Newton converge pour toute initialisation $x_0 \in \mathbb{R}^d$ assez proche de x^*. Plus précisément, on a :*

$$||x^{t+1} - x^*|| \leq \frac{\gamma}{2\alpha}(||x^t - x^*||)^2 \tag{3}$$

$$\tag{4}$$

En particulier, si $||x_0 - x^*|| \leq \frac{\alpha}{\gamma}$ et T est un nombre de pas, on a

$$f(x^T) - f(x^*) \leq \frac{\beta\alpha}{\gamma} 2^{-2^{T+1}}$$

Il est important de prendre garde car le théorème en question ne peut être appliqué à la fonction f si l'on ne peut pas garantir que f est convexe et trois fois différentiable. Or ce n'est pas le cas pour la fonction $f_K(y_1, \ldots, y_K) = \frac{1}{n}\sum_{i=1}^{n} \min_{l \in 1,\ldots,K} ||y_l - x_i||^2$. Ainsi, ce théorème ne peut pas être directment appliqué. Toutefois, il est possible de démontrer que l'algorithme de Lloyd converge pour une bonne initialisation.

De plus, on sait que la méthode de Newton ne converge pas en cas de mauvaise initialisation. La méthode d'initialisation k-means++, que nous allons étudier par la suite, fait partie des bonnes initialisations.

4 Recherche de clustering optimal : Étude de K-means++

L'initialisation de l'agorithme K-means est un problème à part entière. Classiquement, on choisit uniformément les centroïdes initiaux dans les données, mais en pratique ce n'est pas efficace. Si l'initialisation est mal choisie, le minimum local atteint par l'algorithme pourra être très loin de la solution optimale. On cherche donc à améliorer cette initialisation sans ajouter trop de complexité à l'algorithme.

K-means++ désigne un algorithme qui répond à ce problème de façon simple et efficace. L'initialisation est toujours aléatoire, mais augmente la probabilité que les centroïdes initiaux choisis soient espacés. K-means++ a été présenté dans le papier de David Arthur et Sergei Vassilvitskii [AV06]. Le résultat principal nous donne une garantie statistique supplémentaire à l'algorithme K-means, une borne à l'erreur moyenne de la solution optimale.

Nous allons d'abord introduire quelques notations. Jusqu'ici K était constante, mais comme K-means++ ajoute itérativement les c_j^K, on indicera en exposant par K pour bien observer quand K changera. Soit $\mathcal{X}$, un ensemble de points de $\mathbb{R}^n$ que l'on cherche à grouper dans K clusters. On désignera par c_j^K le j-ème centroïde d'un clustering de K clusters. On désignera par C_j^K les points les plus proches de c_j^K dans $\mathcal{X}$.

Soit A un ensemble de points, $c(A)$ est la moyenne de ses points :

$$c(A) = \frac{\sum_{x_i \in A} x_i}{|A|}, \text{ de sorte que } c(C_j^K) = c_j^K$$

On cherche à minimiser l'inertie totale ϕ_{c^K} associée à un K-clustering c^K.

$$\phi_{c^K} = \sum_{k=0}^{K} \sum_{x_i \in C_j^K} \|x_i - c(C_j^K)\|^2$$

Définition 4.0.1. *K-means++ choisit les centroïdes initiaux de cette façon :*

Le premier centroïde est tiré uniformément parmi tous les points. On obtient donc c^1, un ensemble avec un unique centroïde. À la k-ième étape, soit c^{K_0} le vecteur des centroïdes obtenus jusqu'à présent.

Soit $D(x)$, la distance euclidienne de x au centroïde le plus proche déjà tiré.

$$D(x) = \min_{c_i^{K_0} \in c^{K_0}} \|x - c_i^{K_0}\|$$

On ajoute un nouveau centroïde parmis les x, il est tiré aléatoirement avec cette probabilité :

$$p(x) = \frac{D^2(x)}{\sum_{x_i \in \mathcal{X}} D^2(x_i)}$$

On répète ce tirage pour $c_{K_0+1}^{K_0+1}$ jusqu'à avoir K centroïdes. c^K est donc un vecteur aléatoire. Lorsque que l'on complétera un vecteur c^{K_0} de cette façon pour qu'il soit de taille K, on appelera cela un un tirage aléatoire de poids D^2.

Théorème 4.1 (Inertie de Kmeans++)**.** *Après une étape de K-means appliquée aux centroïdes obtenus avec kmeans++, l'inertie ϕ_{c^K} est telle que :*

$$\mathbb{E}[\phi_{c^K}] \leq 8(\ln(k) + 2)\phi_O,$$

Où ϕ_O est l'inertie d'un K-clustering optimal.

Comme l'inertie décroit à chaque étape de K-means, l'espérance de l'inertie finale après convergence de K-means respecte aussi cette inégalité. Voir Lemme 3.1.1.

Définition 4.0.2. *Soit $\mathcal{A} \subset \mathcal{X}$, la contribution des points de A à l'inertie est :*

$$\phi_{c^K}(\mathcal{A}) = \sum_{k=0}^{K} \sum_{x_i \in C_j^K \cap \mathcal{A}} \|x_i - c(C_j^K)\|^2.$$

Lemme 4.0.1. *La somme du carré des distances à un point respecte cette égalité :*

$$\sum_{x \in A} \|x - z\|^2 = |A| \cdot \|z - c(A)\|^2 + \sum_{x \in A} \|x - c(A)\|^2.$$

Démonstration. On a,

$$\begin{aligned}
\sum_{x\in A}\|x-z\|^2 &= \sum_{x\in A}\|x-c(A)+c(A)-z\|^2 \\
&= \sum_{x\in A}\left(\|x-c(A)\|^2+\|z-c(A)\|^2+2\langle c(A)-x, z-c(A)\rangle\right) \\
&= \sum_{x\in A}\|x-c(A)\|^2+\sum_{x\in A}\|z-c(A)\|^2+\sum_{x\in A}\langle c(A)-x, z-c(A)\rangle \\
&= \sum_{x\in A}\|x-c(A)\|^2+\sum_{x\in A}\|z-c(A)\|^2+\langle\sum_{x\in A}c(A)-x, z-c(A)\rangle \\
&= \sum_{x\in A}\|x-c(A)\|^2+\sum_{x\in A}\|z-c(A)\|^2 \\
&= \sum_{x\in A}\|x-c(A)\|^2+|A|\cdot\|z-c(A)\|^2.
\end{aligned}$$

□

Le lemme suivant nous donne l'erreur moyenne intra-cluster quand on se restreint aux clusters optimaux.

Lemme 4.0.2. *Soit $A \in C_O^K$, où C_O^K est un K-clustering optimal*
Soit C^1 un clustering d'un seul cluster dont le centroïde a_0 est tiré uniformément dans A. Alors,

$$\mathbb{E}[\phi_{C^1}(A)] = 2\phi_O(A).$$

Démonstration. Par le théorème de transfert,

$$\begin{aligned}
\mathbb{E}[\phi_{C^1}(A)] &= \sum_{a_0\in A}\frac{1}{|A|}\cdot\left(\sum_{a\in A}\|a-a_0\|^2\right) = \frac{1}{|A|}\cdot\sum_{a_0\in A}\sum_{a\in A}\|a-a_0\|^2 \\
&= \frac{1}{|A|}\sum_{a_0\in A}\left(\sum_{a\in A}\|a-c(A)\|^2+|A|\cdot\|a_0-c(A)\|^2\right), \text{ d'après le lemme 4.0.1.} \\
&= \frac{1}{|A|}\sum_{a_0\in A}\sum_{a\in A}\|a-c(A)\|^2+\sum_{a_0\in A}\|a_0-c(A)\|^2 \\
&= 2\sum_{a_0\in A}\|a_0-c(A)\|^2.
\end{aligned}$$

□

Lemme 4.0.3. *Soit A un cluster de C_O^K, et c^{K_0} un clustering quelconque de K_0 centroïdes. Si on ajoute un centroïde de A à c^{K_0} avec un tirage aléatoire de poids D^2, alors l'inertie obtenue sur A avec le clustering c^{K_0+1} est telle que :*

$$\mathbb{E}[\phi_{c^{K_0+1}}(A)] \leq 8\phi_O(A).$$

Démonstration. La probabilité de tirer a_0 de A est égale à $\frac{D^2(a_0)}{\sum_{a\in\mathcal{X}}D^2(a)}$.

Après avoir tiré a_0, la contribution d'un point $a \in A$ à l'inertie totale est $\min(D(a), \|a-a_0\|)^2$ donc

$$\mathbb{E}[\phi_{c^{K_0+1}}(A)] = \sum_{a_0 \in A} \frac{D^2(a_0)}{\sum_{a \in A} D^2(a)} \sum_{a \in A} \min(D(a), \|a - a_0\|)^2$$

Par inégalité triangulaire, $D(a_0) \leq D(a) + \|a - a_0\|$, donc pour tout a et tout a_0, $D^2(a_0) \leq 2D^2(a) + 2\|a - a_0\|^2$,

Ainsi, on a en sommant pour tout a,

$$|A| D^2(a_0) \leq 2 \sum_{a \in A} D^2(a) + 2 \sum_{a \in A} \|a - a_0\|^2$$

Et donc,

$$\begin{aligned}
\mathbb{E}[\phi_{C^{K_0}}(A)] &\leq \sum_{a_0 \in A} \frac{2\sum_{a \in A} D^2(a) + 2\sum_{a \in A} \|a - a_0\|^2}{|A| \sum_{a \in A} D^2(a)} \sum_{a \in A} \min(D(a), \|a - a_0\|)^2 \\
&\leq \sum_{a_0 \in A} \left(\frac{2}{|A|} + \frac{2\sum_{a \in A} \|a - a_0\|^2}{|A| \sum_{a \in A} D^2(a)} \right) \sum_{a \in A} \min(D(a), \|a - a_0\|)^2 \\
&\leq \sum_{a_0 \in A} \left(\frac{2\sum_{a \in A} \min(D(a), \|a - a_0\|)^2}{|A|} + \frac{2\sum_{a \in A} \min(D(a), \|a - a_0\|)^2 \sum_{a \in A} \|a - a_0\|^2}{|A| \sum_{a \in A} D^2(a)} \right) \\
&\leq \sum_{a_0 \in A} \left(\frac{2\sum_{a \in A} \|a - a_0\|^2}{|A|} + \frac{2\sum_{a \in A} D^2(a) \sum_{a \in A} \|a - a_0\|^2}{|A| \sum_{a \in A} D^2(a)} \right) \\
&\leq \sum_{a_0 \in A} \left(\frac{2\sum_{a \in A} \|a - a_0\|^2}{|A|} + \frac{2\sum_{a \in A} \|a - a_0\|^2}{|A|} \right) \\
&\leq \frac{4}{|A|} \cdot \sum_{a_0 \in A} \sum_{a \in A} \|a - a_0\|^2 \\
&\leq 8\phi_O(A), \text{ d'après le lemme 4.0.2.}
\end{aligned}$$

□

Le lemme suivant est nécéssaire à la démonstration du théorème principal et sera admis. Il nous donne une borne sur l'erreur moyenne quand on ajoute des centroïdes avec tirage aléatoire de poids D^2, en fonction du nombre de cluster optimaux dont on a découvert au moins un point.

Lemme 4.0.4. *Soit c^{k_0} un clustering de $\mathcal{X}$. On choisit $u > 0$ clusters de C_o^K tels qu'aucun des centroïdes de c^{K_0} ne soit dans ces clusters. On désignera par $\mathcal{X}_u$ les points des clusters correspondants dans C_o^K. On pose $\mathcal{X}_c = \mathcal{X} \setminus \mathcal{X}_u$. On ajoute $t \leq u$ centroïdes à c^{K_0} avec un tirage aléatoire de poids D^2. On appelle c^{K_0+t} le clustering obtenu. On regarde $\phi_{c^{K_0+t}}$ l'inertie correspondante. Alors,*

$$\mathbb{E}[\phi_{c^{K_0+t}}] \leq \left(\phi_{c^{K_0}}(\mathcal{X}_c) + 8\phi_O^K(\mathcal{X}_u)\right) \cdot (1 + H_t) + \frac{u-t}{u} \cdot \phi_{c^{K_0}}(\mathcal{X}_u),$$

où H_t désigne la somme harmonique $\sum_{i=1}^{t} \frac{1}{i}$.

Nous allons désormais démontrer le théorème principal sur K-means++.

Démonstration. Soit A, le cluster dans lequel on choisit le premier centroïde. On applique le lemme précédent avec $t = u = K - 1$.

On obtient

$$\begin{aligned}
\mathbb{E}[\phi_{c^K}] &\leq (\phi_{c^K}(A) + 8\phi_O(\mathcal{X} \setminus A)\,(1 + H_{K-1}) \\
&\leq (\phi_{c^K}(A) + 8\phi_O(\mathcal{X}) - 8\phi_O(A))\,(1 + H_{K-1}) \\
&\leq (2\phi_O(A) + 8\phi_O(\mathcal{X}) - 8\phi_O(A))\,(1 + H_{K-1}), \text{ d'après le lemme 4.0.2} \\
&\leq 8\phi_O(\mathcal{X})(1 + H_{K-1}) \\
&\leq 8\phi_O(\mathcal{X})(2 + \ln(K)).
\end{aligned}$$

□

5 Application du K-Means en Python : résultats et interprétation

Dans cette partie, nous nous intéresserons à l'application de l'algorithme sur des jeux de données simples. De cette manière, nous pourrons vérifier si l'algorithme du K-Means fonctionne bien, et en particulier s'il est capable de détecter des clusters plus ou moins évidents dans un premier temps. Pour ce faire nous avons développé notre propre implémentation de l'algorithme en Python (2) avant d'opter pour la méthode issue de la librairie `scikit-learn`.

5.1 Exemples sur des cas "simples" de mélanges gaussiens

L'algorithme K-Means est utilisé pour la partition des données en plusieurs groupes distincts. Pour tester l'efficacité de cet algorithme, une bonne idée est de l'appliquer sur des données générées par un modèle de mélange gaussien. Cette approche est particulièrement intéressante car, en utilisant les paramètres adéquats de moyennes et de variances, on peut obtenir un échantillon de points divisé en plusieurs clusters bien distincts. De cette manière, elle permet de valider la capacité de l'algorithme K-Means à identifier et séparer les différents groupes de données.

Définition 5.1.1 (Mélange gaussien)**.** *Soit $X = (X_1, ..., X_n) \in (\mathbb{R}^d)^n$ un échantillon de variables aléatoires iid de loi $\sum_{i=1}^{K} p_i \mathcal{N}(\mu_i, \Sigma_i)dx$ où $(p_1,...,p_K)$ est un vecteur de probabilités. On dit alors que X suit un mélange gaussien.*

Le mélange gaussien est une distribution de probabilité qui peut être représentée par la somme pondérée de plusieurs distributions gaussiennes. Chaque distribution gaussienne est caractérisée par sa moyenne et son écart-type. En particulier, c'est un modèle pour modéliser des données en tant que combinaison de plusieurs distributions gaussiennes, chacune pondérée par un coefficient de mélange. Les mélanges gaussiens ont de nombreuses applications dans la modélisation de données complexes, notamment en classification non supervisée. L'exemple ci-dessous reprénte la densité d'un mélange gaussien.

Exemple 5.1 (Densité d'un mélange gaussien)**.** *On considère $\mu_1 = -40$, $\mu_2 = 40$, $\sigma_1 = 20$, $\sigma_2 = 15$, $p_1 = p_2 = \frac{1}{2}$.*

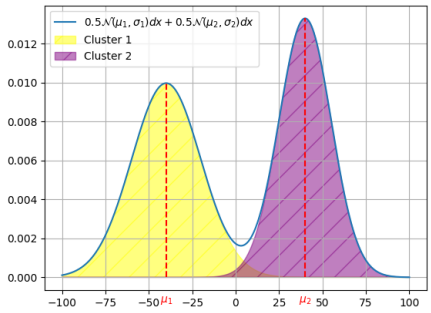

FIGURE 2 – Exemple de mélange gaussien

Générer une observation de mélange gaussien consiste à tirer une observation gaussienne $\mathcal{N}(\mu_i, \Sigma_i)$ avec probabilité p_i. Dans l'exemple ci-dessus, on associe à chacune des deux gaussiennes un cluster, correspondant à la plage de valeur sous la zone coloriée de chacune des

distributions. A présent, regardons l'efficacité de l'algorithme K-Means sur des données distribués selon un mélange gaussien. Pour une bonne visualisation, nous nous placerons dans cette partie en dimension $d = 2$.

Exemple 5.2. *On considère ici un mélange de deux gaussiennes multi-dimensionelles* $\mathcal{N}_1 = \mathcal{N}(\mu_1, \Sigma_1)$ *et* $\mathcal{N}_2 = \mathcal{N}(\mu_2, \Sigma_2)$*, où* $\mu_1 = (0,0)^T, \mu_2 = (12,12)^T$*,* $\Sigma_1 = \begin{pmatrix} 3 & 1 \\ 1 & 3 \end{pmatrix}$*,* $\Sigma_2 = \begin{pmatrix} 7 & 1 \\ 1 & 7 \end{pmatrix}$*;* $p_1 = p_2 = 0.5$*. On applique sur un ensemble de données de taille* $n = 300$ *une classification de niveau* $K = 2$*.*

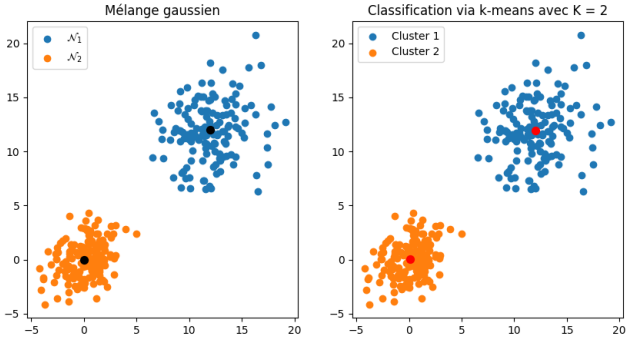

FIGURE 3 – Classification par l'algorithme des K-Means

Résultats :

— L'algorithme converge. Pour une initialisation aléatoire de centroïdes, il suffit généralement d'une itération.

— L'algorithme génère de "bonnes" partitions (séparées ici par leur couleur). Ces K = 2 partitions correspondent exactement aux deux gaussiennes choisis pour la distribution de l'échantillon.

— Les centroïdes, représentés en rouge dans le tracé de droite, se rapprochent des vrais moyennes de chacune des gaussiennes.

```
Centroids: [12.02894821 11.90519541] [0.11562845 0.07204774]
```

Cette performance de classification est due principalement à deux choses. D'abord, la distance des sous-ensembles des données est assez grande. En d'autre termes, le nuage est décomposé en plusieurs sous-nuages qui ne se recoupent pas. De plus, les données sont séparées en ensembles convexes, donc chaque point de données est généralement plus proche du centre de son cluster que des centres des autres clusters. Ainsi, comme K-Means trouve les centroïdes qui minimisent la somme des carrés des distances des points à ceux-ci, le deuxième résultat était attendu. Ajoutons quelques remarques sur les deux derniers résultats.

D'abord, il est évident que ceux-ci seraient différents si on avait choisi un autre K que $K = 2$. Le regroupement ne serait pas aussi pertinent selon les clusters qu'on cherche à priori à dévoiler.

Ensuite, le troisième résultat est d'autant plus vrai que la taille de l'échantillon n devient très grand. En effet, on peut constater que l'erreur définit par $\|\mu_i - c_i\|_2^2$ où c_i est le centroïde le plus proche de μ_i se rapproche de zéro quand n est grand (sous $K = 2$). Ici, les vrais moyennes μ_1 et μ_2 des gaussiennes sont les centres qui minimisent la somme des distances entre les points et les centres les plus proches.

Itérations considérés : `[10,20,30,...,980,990,1000]`

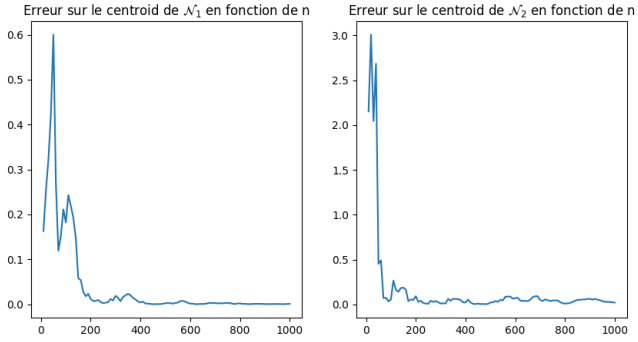

FIGURE 4 – Graphes des erreurs sur chacun des centroïdes

Ces remarques sont valables uniquement dans le contexte spécifique dans lequel nous les avons formulées. En modifiant les paramètres (moyennes,variances,K,...), celles-ci ne sont en générale plus vraies. Les exemples suivant montrent ce fait et de surcroît que l'algorithme du K-Means peut devenir un choix moins judicieux dans certaines situations.

Exemple 5.3. *On considère toujours un mélange de deux gaussiennes sur lequel on veut appliquer la classification avec $K = 2$. Regardons son effet en choisissant des moyennes "proches" en terme de distance, donc $\mu_1 = (0,0)^T, \mu_2 = (2,2)^T$. De plus, prenons des matrices de covariance qui rendent la dispertion de chacune des gaussiennes assez grande, soit comme l'exemple précédent $\Sigma_1 = \begin{pmatrix} 3 & 1 \\ 1 & 3 \end{pmatrix}$, $\Sigma_2 = \begin{pmatrix} 7 & 1 \\ 1 & 7 \end{pmatrix}$. On prend encore $p_1 = p_2 = 0.5$.*

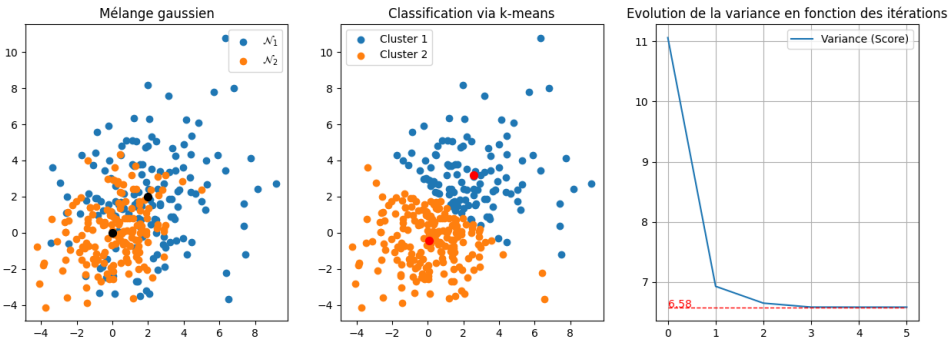

FIGURE 5 – Classification par l'algorithme des K-Means

Résultats :

— L'algorithme converge en cinq itérations. On a toujours une réduction progressive de la variance, avec comme optimum trouvé `6.58`.

— Mais les clusters trouvés par l'algorithme ne coincident pas avec les clusters initiaux.

— Les centroïdes ne se rapprochent pas des moyennes des gaussiennes.
`Centroids: Centroids: [2.57109249 3.17025919] [ 0.07450893 -0.41792057]`

Exemple 5.4. *Similairement, considérons un mélange gaussien de taille* $n = 10000$ *munie de poids deséquilibrés, comme* $p_1 = 0.1$, $p_2 = 0.9$

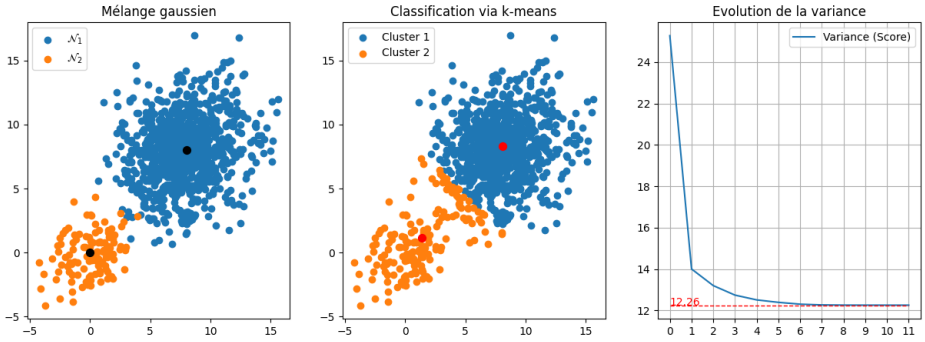

FIGURE 6 – Classification par l'algorithme des K-Means

`Centroids: [8.10485873 8.28354534] [1.4017334 1.13676944]`

En plus de la classification nous avons vu que l'algorithme de Lloyd est un aussi un moyen de résoudre le problème de quantification en approchant la loi sous-jacente d'un échantillon de variables aléatoires indépendantes par une mesure supporté sur K points. Le support en question serait alors les K centroïdes trouvés à la convergence de l'algorithme.

Exemple 5.5. *On considère* $n = 10000$ *réalisations d'un mélange gaussien de 4 composantes, et on supposoe qu'on souhaite réaliser* $K = 4$ *partitions. En représentant les clusters obtenus ainsi que par les cellules de Voronoï associées aux centroïdes par dessus, on obtient la Figure 7.*

On observe dans cette figure que les cellules de Voronoï associées aux centroïdes délimitent exactement les clusters. On note $\hat{y}_1, \dots, \hat{y}_K$ *les centroïdes et* $C_1, \dots, C_K$ *les clusters numériquement obtenus. Supposons que* y *soit la grille de quantification optimale pour la loi sous-jacente* μ*, et* $(C_k(y))_{1 \leq k \leq K}$ *une partition de Voronoï générée par* y*. Alors pour tout* $k \in \{1, \dots, 4\}$*, une approximation du poids* $\mu(C_k(y))$ *est* $|C_k|/n$*, autrement dit la fréquence de points observés dans le cluster* k*. Cette fréquence est l'estimateur de la Remarque 2.1 appliqué aux observations. Ainsi, la mesure* $\hat{\mu}^4 = \sum_{k=1}^{4} \frac{|C_k|}{n} \delta_{\hat{y}_k}$ *est une mesure qui approxime* μ*. Numériquement, on trouve le support et le vecteur des poids suivants :*

`Centroids: [ 2.01516369, -2.05730937] [-1.97535827, -4.01225507] [2.06333189, 1.08438386] [-1.06038243, 1.04388805]`

`Probabilities: [0.2584, 0.2456, 0.246, 0.25].`

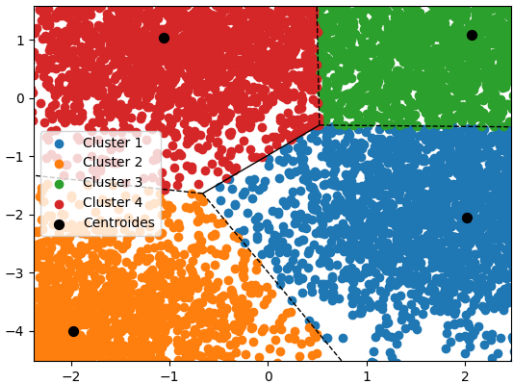

FIGURE 7 – Clustering et cellules de Voronoï

5.2 Exploration de la performance sur des données de formes plus complexes

Comme nous l'avons vu précédemment, l'algorithme du K-Means est en général bien adapté pour des données divisés en clusters gaussiens de distance assez grande entre eux. Néanmoins, on peut se demander si l'algorithme est adapté pour des jeux de données plus complexes.

Définition 5.2.1 (Données linéairement séparables)**.** *Des données sont* ***linéairement séparables*** *si elles peuvent être séparées par une ligne droite ou un hyperplan. Autrement dit, il est possible de tracer une frontière de décision linéaire (une ligne droite ou un hyperplan) qui permet de séparer les exemples dits positifs et négatifs de manière exacte.*

Exemple 5.6. *Cercles emboités. On tire uniformément n couples (U, V) sur l'intervalle $[0, 1]$, et ensuite nous appliquons la transformation $(X, Y) = (rcos(U), rsin(V))$ où $r > 0$. On choisit un cercle de rayon 1 admettant 50 points, et un cercle de rayon 4 avec 60 points.*

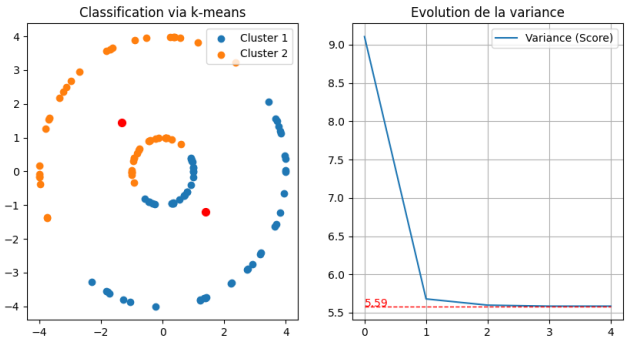

FIGURE 8 – Classification par l'algorithme des K-Means

Exemple 5.7. *Demi-cercles. La librairie* `scikit-learn` *fournit un moyen de tracer des demi-cercles sur une base aléatoire.*

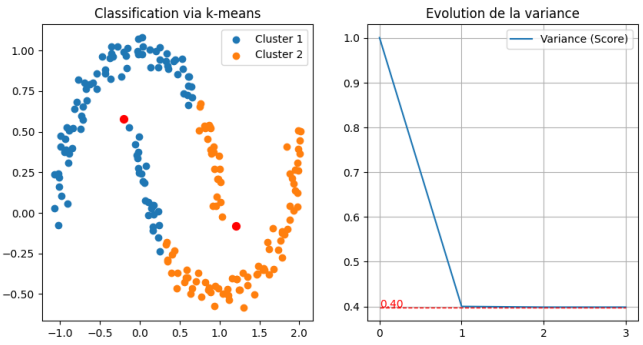

FIGURE 9 – Classification par l'algorithme des K-Means

Résultats :

— L'algorithme converge dans les deux cas. On trouve bien un minimiseur au problème de l'inertie totale intra-classe.

— Les clusters mis en évidence ne sont pas ceux qui sont à priori intuités. En effet, si dans le premier exemple des points situés sur le petit cercle et d'autres sur le grand cercle partagent le même cluster. Même constat dans l'exemple des demi-cercles.

L'échec vient du fait que les deux groupes de données représentés dans ces derniers exemples ne sont pas linéairement séparables. En effet, le critère minimisé par l'algorithme est la somme des variances intra-classes. Par conséquent, les clusters trouvés, même si leur pertinence peut être remis en cause, restent optimaux au sens du problème. Cela veut dire que le clustering correspondant à associer à chaque (demi)-cercle son propre cluster, n'est pas forcément une solution (même locale) au problème. Cette idée est valable dans beaucoup de cas, même parfois lorsque les données sont linéairement séparables, comme on peut le voir dans l'exemple suivant.

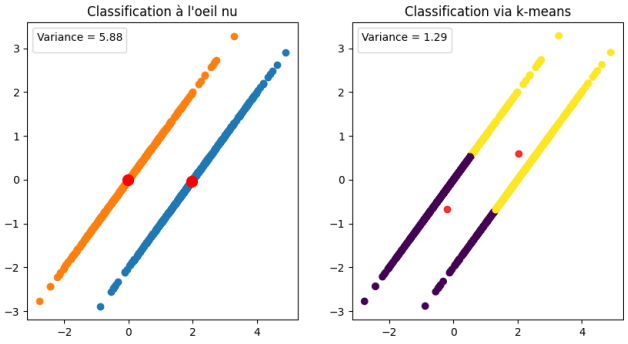

FIGURE 10 – Classification par l'algorithme des K-Means

Même si les clusters de gauche sont plus intuitifs, la variance associée est nettement supérieur à celle associée aux clusters de droite : `5.88` > `1.29`.

Une idée pour résoudre ces problèmes de clustering est l'application d'une fonction sur le nuage de points. L'idée est de projeter les données dans un espace où les caractéristiques pertinentes sont mises en évidence et les caractéristiques non pertinentes sont éliminées ou réduites. En utilisant

une fonction de projection, l'algorithme K-means peut identifier les groupes de données plus facilement et avec une plus grande précision. Cela est dû au fait que les données sont maintenant représentées dans un espace où la distance entre les points est plus significative et cohérente avec la structure des données. L'exemple suivant illustre cette technique.

Exemple 5.8 (Feature mapping)**.** *On reprend l'exemple 8 des cercles emboités, en appliquant sur les données d'entrée la transformation f suivante :*

$$\begin{aligned} f: \quad \mathbb{R}^2 &\longrightarrow \mathbb{R}^3 \\ (x, y) &\longmapsto (x, y, 2\|(x, y)\|) \end{aligned}$$

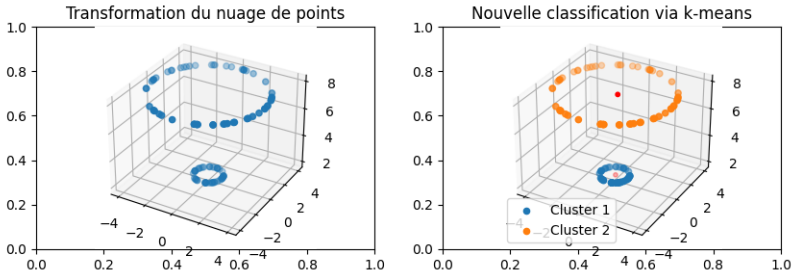

FIGURE 11 – Classification par l'algorithme des K-Means après transformation

On est passé d'un espace de dimension 2 *à un espace de dimension* 3*. L'algorithme converge toujours et donne une nouvelle partition des points.*

On appelle cette méthode l'*astuce du noyau*. En apprentissage automatique, l'astuce du noyau, ou kernel trick en anglais, est une méthode qui permet d'utiliser un classifieur linéaire pour résoudre un problème non linéaire. L'idée est de transformer l'espace de représentation des données d'entrées en un espace de plus grande dimension, où un classifieur linéaire peut être utilisé et obtenir de bonnes performances [Wik23a]. Or, pour K=2, l'algorithme K-means cherche à séparer les données en deux clusters en traçant une ligne droite (ou un hyperplan) dans l'espace. Cette ligne droite peut alors être considérée comme un classificateur linéaire qui sépare les deux clusters.

5.3 Sensibilité à l'initialisation

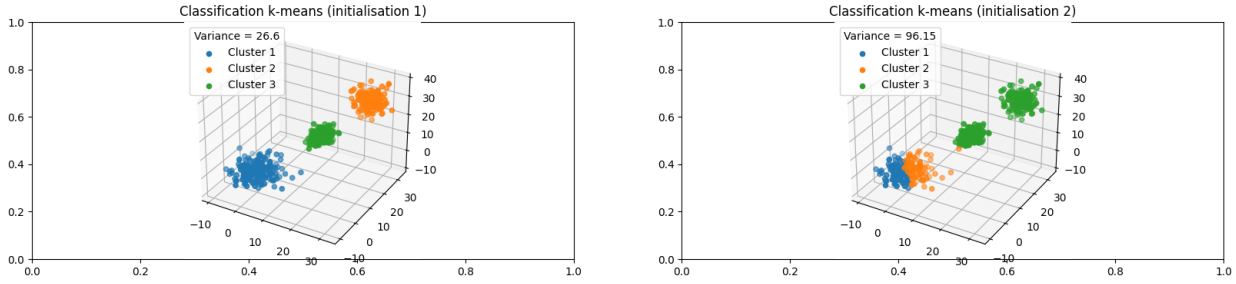

FIGURE 12 – Clusterings en fonction de l'initialisation

Les résultats obtenus dépendent fortement des centres initiaux choisis pour chaque cluster. Si les centres initiaux sont mal choisis, l'algorithme risque de converger vers un minimum local plutôt que le minimum global. En pratique, pour réduire les effets de la sensibilité à l'initialisation, il est courant d'effectuer plusieurs fois l'algorithme et de sélectionner la meilleure solution parmi ces initialisations en utilisant une métrique de qualité comme l'inertie intra-cluster. C'est ce que fait par défaut la fonction proposée par *scikit-learn*. Même si une manière courante d'initialiser les centres est de les choisir de manière aléatoire, il existe des choix d'initialisation qui augmentent la probabilité d'avoir un minimum global. On peut citer par exemple K-means++ qui est une technique abordée en détail dans la section 4. En effet, cette initialisation permet de borner l'erreur moyenne par l'erreur optimale à une constante multiplicative près, donc l'erreur ne peut pas être « trop grande ».

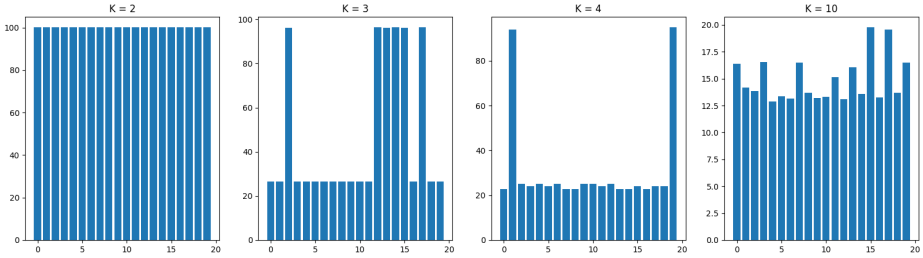

FIGURE 13 – Volatilité de la variance en fonction du niveau K

Nous avons vu dans l'Exemple 2.4 que le problème de quantification peut admettre plusieurs solutions. Nous savons aussi que l'algorithme K-means permet d'accéder à maximum un seul minimiseur. Toutefois, la sensibilité de l'algorithme à l'initialisation permet dans certains cas de trouver numériquement chacun des minimiseurs.

Exemple 5.9 (Retour sur l'Exemple 2.4)**.** *On considère un échantillon de* $n = 1000$ *points tirés selon une loi uniforme* $\mathcal{U}([-2,-1] \cup [1,2])$*. On prend* $K = 3$ *et on considère les initialisations suivantes :*

1. $(-1.7, -1.5, 1.5)$

2. $(-1.5, 1.5, 1.7)$.

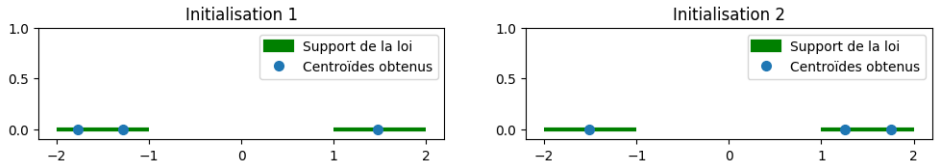

FIGURE 14 – Minima trouvés par l'algorithme K-means en fonction de l'initialisation

```
Centroids = [-1.76516819 -1.27739078 1.47771997]
Centroids = [-1.50818804 1.25887535 1.75102817]
```

On a bien approché les solutions théoriques de l'ensemble $\{(-\frac{7}{4}, -\frac{5}{4}, \frac{3}{2}), (-\frac{3}{2}, \frac{5}{4}, \frac{7}{4})\}$.

5.4 Application au jeu de données Mnist

MNIST est une base de données de chiffres manuscrits très utilisée dans le domaine de l'apprentissage automatique et de la vision par ordinateur. Elle contient un ensemble de 70 000 images de chiffres manuscrits, chacune étant normalisée pour avoir une taille de 28x28 pixels et une orientation uniforme. Les images sont divisées en deux ensembles : un ensemble d'entraînement de 60 000 images et un ensemble de test de 10 000 images. Chaque image est associée à une étiquette qui indique le chiffre qu'elle représente, de 0 à 9. La base de données MNIST est souvent utilisée comme référence pour évaluer et comparer les performances des algorithmes de reconnaissance de caractères manuscrits et de classification d'images. Par conséquent, se pose la problématique suivante : Peut-on efficacement regrouper le jeu de données en $K = 10$ partitions où chacune correspondrait à un chiffre ? Il existe plusieurs bibliothèques Python qui offrent des fonctions pour importer, manipuler et visualiser les données de MNIST. Dans le code donné en annexe C, nous utilisons `Tensorflow`.

Feature engineering (annexe C) : Pour commencer, il faut redimensionner les données que nous possédons car l'algorithme prend en entrée une structure de données unidimensionnelle. De plus, on fait une normalisation des données en les divisant par `x_train.max() = 255`, de sorte à les ramener toutes les valeurs entre 0 et 1.

Entraînement (annexe D) : Pour appliquer K-Means, on utilise la fonction prête à l'emploi proposée par la librairie `scikit-learn`. L'initialisation choisie est k-means++, elle permet d'améliorer la probabilité d'obtenir la solution optimale (minimum global). Nous verrons cette technique plus en détail dans une autre partie. Nous observons bien une décroissance de l'inertie au fil des itérations.

```
Initialization complete
Iteration 0, inertia 4006291.001307185.
Iteration 1, inertia 2478254.6282684174.
Iteration 2, inertia 2422025.241099925.
...
Iteration 103, inertia 2369222.484131025.
Iteration 104, inertia 2369222.4283553385.
Converged at iteration 104: center shift 5.682393798457611e-06 within tolerance
6.725132078460147e-06.
```

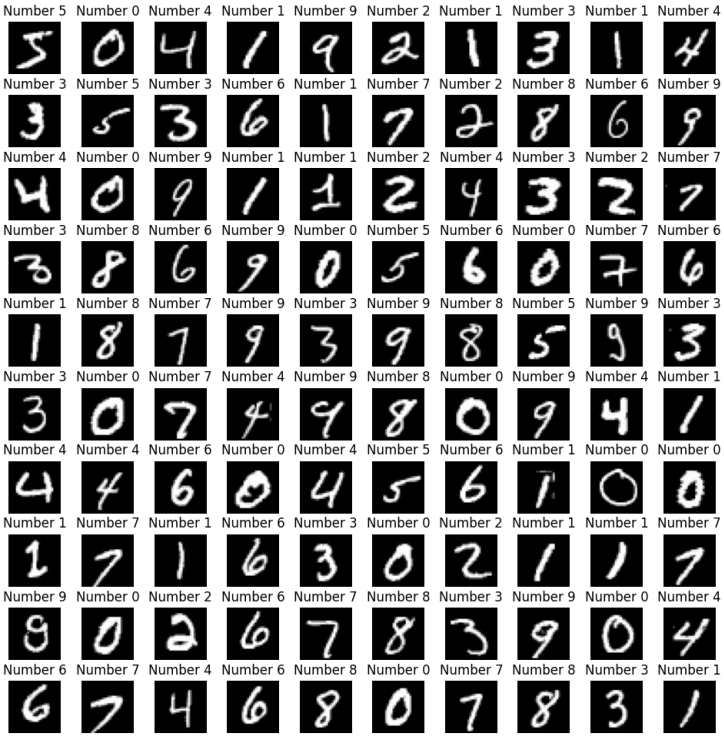

FIGURE 15 – Visualisation de données Mnist

On trouve les centroïdes suivants :

FIGURE 16 – Centroïdes pour K = 10

Résultats sur l'algorithme :

— L'algorithme converge.

— Les centroïdes font clairement apparaître certains chiffres de manière unique comme 8 et 0. D'autres chiffres ne se dévoilent pas d'une manière aussi évidente comme le chiffre 3, qui semble apparaître deux fois. Le chiffre 4 n'est cependant pas identifiable.

Résultats métriques :

```
Inertie: 2369222.3625731682
Variance: 39.48703937621947
Score (opposé de l'inertie): -2369222.3625731682
```

Il se trouve que l'inertie (et de manière équivalente la variance) ne nous renseigne pas réellement sur la pertinence de l'algorithme dans ce cas-ci. En effet, avec cette information seule on

ne peut pas juger de la pertinence du partitionnement sur le jeu de données Mnist. A la place de cette quantité, on peut évaluer l'*accuracy* du modèle. Pour ce faire, on peut utiliser les étiquettes associés à chaque image qui sont donnés par le jeu de donnée mnist que l'on a importé **??**, et les comparer avec les étiquettes qu'on leurs prédits. Cependant, K-Means est un modèle d'apprentissage non supervisé. Par conséquent, les étiquettes que l'on attribura par notre algorithme K-means feront référence au cluster $k \in \{1, \dots, 10\}$ auquel chaque donnée a été attribué, et non à l'entier cible réel.

Soit $\mathcal{D}^{x,y} = \{(x_i, y_i), 1 \leq i \leq n\}$ l'ensemble des données issus de Mnist. Pour chaque i, x_i représente une matrice et $y_i \in \{0, \dots, 9\}$ le **vrai** label donc le chiffre représenté. On définit ainsi l'*accuracy* par la quantité $\frac{1}{n} \sum_{i=1}^{n} 1_{\{y_i = \hat{y}_i\}}$ où $\hat{y}_i \in \{0, \dots, 9\}$ représente l'étiquette assignée au cluster de la donnée X_i. Comment déterminer l'étiquette de chaque cluster ? En regardant quelle chiffre est le plus représentée dans le cluster (voir E inspiré de [Sha]). Ainsi, on peut regarder quelles sont les étiquettes attribuées aux centroïdes.

```
Predicted labels: [0 1 2 3 4 6 7 8]
```

On constate que certains chiffres n'ont été attribués, comme le 9 et le 5. Cela signifie que d'autres l'ont été plusieurs fois. En d'autres termes, les images représentant 9 et 5 ne sont surreprésentés dans aucun cluster. Cela s'explique que la « forme » de chacun de ces chiffres ne constitue pas un centroïde optimal pour le problème K-Means.

On obtient l'accuracy suivante :

```
Accuracy: 0.5972166666666666
```

Ce qui signifie que 60 pourcent des images ont une étiquette prédite qui correspond à leur véritable étiquette. Il s'agit d'un taux plutôt satisfaisant.

6 Conclusion

Ce mémoire a examiné les garanties statistiques et algorithmiques de l'algorithme de clustering K-means. Nous avons exploré les avantages et les inconvénients de cette méthode de clustering largement utilisée dans les applications pratiques, et nous avons examiné les preuves théoriques de la convergence de l'algorithme. Nous avons également étudié les conditions requises pour que les résultats du clustering soient statistiquement significatifs.

Lors de notre étude, nous nous somme donnés K comme étant un hyperparamètre fixe. Cependant, il aurait été intéressant d'examiner de manière plus approfondie le processus de sélection de la valeur de K dans le cadre de notre étude. Il est facile de vérifier que si K augmente, la variance intra-classe diminue. De ce fait, il est possible de trouver un K optimal pour le clustering : ce serait un K à partir duquel la variance ne décroît plus significativement. Pour déterminer cette valeur, on peut exploiter ce qu'on appelle le graphe Elbow.

Enfin, l'idée de K-means est de minimiser une somme d'erreurs au carré. Le choix du carré de la distance euclidienne est donc cruciale pour garantir la convergence vers un bon minimum si on utilise l'algorithme de Lloyd. Typiquement, si on choisit une autre distance comme celle de Manhattan, K-means n'est plus adapté. Il faut soit le modifier, soit transformer les données, soit choisir un autre algorithme. En fait, il s'avère que l'algorithme de Lloyd n'est pas le seul qui permet de résoudre le problème de partitionnement. Par exemple, il existe la méthode des K-medoïdes. Celle-ci a l'avantage de pouvoir être utilisée avec des distances arbitraires, tandis que l'algorithme traditionnelle nécessite généralement la distance euclidienne qu'on élève au carré pour des solutions efficaces. Elle est plus robuste au bruit et aux valeurs aberrantes que les K-means de Lloyd. Un autre point est que l'algorithme de Lloyd a des grandes chances de rester bloqué dans un optimum local, en fonction de son initialisation. Nous avons proposé la

modification k-means++ pour pallier ce problème, mais il existe d'autres modifications possibles de l'algorithme comme celui de Hartigan-Wong.

En définitive, l'algorithme K-means demeure une méthode de clustering simple mais puissante qui peut globalement être utilisée avec succès dans une variété de problèmes de classification non-supervisé et quantification d'une loi de probabilité.

Appendices

A Librairies utilisés

```
import sys
import sklearn
import numpy as np
import math
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow import keras
import sklearn.cluster
import random as rd
import time
from scipy.stats import norm
from scipy.spatial import Voronoi, voronoi_plot_2d
from sklearn.metrics import accuracy_score

```

Code source 1 – Librairies utilisés pour le mémoire

B Fonctions principales utilisées en Python

```
# Fixer la graine.
seed = 5

# Fonction qui genere un echantillon de taille n issu d'un melange gaussien.
def gaussian_mixture(n,probs,means,covs):
    """
    Genere un echantillon a partir d'un melange de gaussiennes.

    Args:
        n (int): Nombre total d'observations a generer.
        probs (list): Liste des probabilites de chaque composante de la
    distribution.
        means (list): Liste des moyennes de chaque composante de la distribution
    .
        covs (list): Liste des matrices de covariance de chaque composante de la
     distribution.

    Returns:
        tuple : Un tuple contenant :
            - np.array : Un tableau de taille (n, d), ou d est la dimension de l
    'espace des observations. Chaque ligne est une observation generee a partir
    du melange de gaussiennes.
            - list : Une liste contenant le nombre d'observations generees pour
    chaque composante de la distribution.

    Raises:
        ValueError: Si les listes probs, means et covs n'ont pas la meme
    longueur ou si elles sont vides.

    """
    # Initialisation du generateur de nombres aleatoires
    random_gen=np.random.default_rng(seed)
    # Generation du nombre d'observations pour chaque composante de la
    distribution
```

```
27      nb=random_gen.multinomial(n,probs)
28      sizes = []
29      X=[]
30      d = means[0].shape[0]
31
32      # Boucle sur chaque composante de la distribution
33      for i in range(len(probs)):
34          if (d==1):
35            # Cas unidimensionnel : generation de nombres aleatoires a partir d'
        une loi normale unidimensionnelle
36            u=random_gen.normal(means[i],covs[i],size=nb[i])
37          else:
38            # Cas multidimensionnel : generation de nombres aleatoires a partir d'
        une loi normale multidimensionnelle
39            u=random_gen.multivariate_normal(means[i],covs[i],size=nb[i])
40          X.append(u)
41          sizes.append(nb[i])
42      return (np.concatenate(X),sizes)
43
44
45  def circle_random_points(n,r):
46
47      """
48      Genere n points aleatoires uniformement repartis sur un cercle de rayon r
        centre en (0, 0).
49
50      """
51      # Cree un generateur de nombres aleatoires
52      random_gen=np.random.default_rng(seed)
53
54      v=2*np.pi*random_gen.uniform(0,1,size=n)
55      x=r*np.cos(v)
56      y=r*np.sin(v)
57      points=np.transpose([x,y])
58      return points
59
60
61  def scatter_points(points):
62      """
63      Affiche un nuage de points a partir d'une liste de points.
64      """
65      (x,y)=np.array(points).transpose()
66      plt.scatter(x,y)
67
68  def ax_scatter_points(points,ax):
69      """
70      Affiche un nuage de points sur un axe spacifie a partir d'une liste de
        points.
71      """
72      (x,y)=np.array(points).transpose()
73      ax.scatter(x,y)
74
75
76  def arg_closest(point,centroids):
77      """
78      Renvoie l'indice (ou les indices si plusieurs sont a egale distance)
79      du centroide le plus proche d'un point donne.
80
81      Args:
82      - point (array): un point dans l'espace.
83      - centroids (list): liste des centroides.
84
```

```
    Returns:
    - arg (list): indice (ou indices) du centroide le plus proche.
    """
    arg=[0]
    for i in range(1,len(centroids)):
        d1=np.linalg.norm(point-centroids[i])
        d2=np.linalg.norm(point-centroids[arg[0]])
        if d1==d2:
            arg.append(i)

        if d1<d2:
            arg=[i]
    if len(arg)!=1:
        print("nombre de centroides a distance egale:",len(arg))
    return arg

def find_centroids(cluster):
    """
    Calcule le centroide d'un cluster.

    Args:
    - cluster (list): liste des points du cluster.

    Returns:
    - centroids (list): liste des centroides.
    """
    centroids=[]
    for i in cluster:
            centroids.append(np.mean(i,axis=0))
    return centroids


def find_centroids_replaceifempty(cluster,centroids):
    """
    Calcule le centroide d'un cluster et remplace les centroides vides par les
    centroides precedents.

    Args:
    - cluster (list): liste des points du cluster.
    - centroids (list): liste des centroides.

    Returns:
    - new_centroids (list): liste des nouveaux centroides.
    """
    new_centroids=[]
    for i in range(len(cluster)):
            if len(cluster[i])!=0:
                new_centroids.append(np.mean(cluster[i],axis=0))
            else:
                new_centroids.append(centroids[i])
    return new_centroids

def kmeans_step (points,cluster,centroids):
    """
    Etape du k-means: attribue a chaque point le cluster dont le centroide est
    le plus proche.

    Args:
    - points (array): tableau contenant les points.
    - cluster (list): liste des clusters.
    - centroids (list): liste des centroides.

```

```
    Returns:
    - new_cluster (list): nouvelle liste des clusters.
    """
    new_cluster=[]
    for i in range(len(centroids)):new_cluster.append([])
    for i in points:
        arg=arg_closest(i,centroids)
        arg_default=arg[0]
        new_cluster[arg_default].append(i)
    return new_cluster


#K-Means
def kmeans(points,cluster,calculate_var=True):
    """
    Implemente l'algorithme K-Means pour une liste de points et de clusters.

    Args:
    - points (array): tableau contenant les points.
    - cluster (list): liste des clusters initiaux.
    - calculate_var (bool): si True, calcule la variance a chaque etape de l'
    algorithme.

    Returns:
    - cluster (list): liste des clusters finale.
    - var_tab (array): tableau de la variance a chaque etape de l'algo (ou
    tableau vide si calculate_var=False).
    """

    # Calcul des centroides initiaux
    centroids=find_centroids(cluster)
    # Initialisation du tableau de variance si demande
    if calculate_var:
      var_tab = np.array([variance(points,cluster,centroids)])
    else:
      var_tab = np.array([])

    # Etape initiale
    new_cluster=kmeans_step(points,cluster,centroids)
    new_centroids=find_centroids_replaceifempty(new_cluster,centroids)
    # Boucle principale
    while not(np.array_equal(centroids,new_centroids)):
        # Mise a jour des clusters et des centroides
        (cluster,centroids)=(new_cluster,new_centroids)
        new_cluster=kmeans_step(points,cluster,centroids)
        new_centroids=find_centroids_replaceifempty(new_cluster,centroids)
        # Calcul de la variance si demande
        if calculate_var:
          var_tab = np.append(var_tab,variance(points,new_cluster,new_centroids)
    )

    return cluster,var_tab


def random_init_same_size(k,points):
    """
    Repartit de maniere aleatoire les points en k clusters de meme taille.

    Args:
    - k (int): nombre de clusters
    - points (array-like): liste ou tableau numpy des points a repartir

```

```
    Returns:
    - new_cluster (list): liste de k clusters ou chaque cluster est une liste de
     points
    """
    random_gen=np.random.default_rng(seed)
    a=np.array(points)
    random_gen.shuffle(a)
    new_cluster=[]
    for i in range(k): new_cluster.append([])
    for i in range(len(a)):new_cluster[i%k].append(a[i])
    return new_cluster

def variance(points,clusters,centroids):
    """
    Calcule la variance intra-classe a partir de l'ensemble des points, des
    clusters, et des centroides.

    Args:
    - points (array-like): liste ou tableau numpy des points a repartir
    - clusters (list): liste de k clusters ou chaque cluster est une liste de
    points
    - centroids (list): liste de k centroides

    Returns:
    - var (float): variance intra-classe
    """
    n = len(points)
    K = len(clusters)
    dist = 0
    for k in range(K):
      for i in range(n):
        for j in clusters[k]:
          if ((np.array(points[i]) == j).all()): # si le point appartient au
    cluster k
            dist += np.linalg.norm(points[i]-centroids[k],ord=2)**2
    return (1/n)*dist # variance intra-classe
```

Code source 2 – Mélange gaussien et algorithme du K-Means à la main

C Manipulation et affichage des données mnist

```
# Extraction de donnees mnist et division en donnees train et test
mnist = tf.keras.datasets.mnist
(x_train,y_train),(x_test,y_test) = mnist.load_data()
# 60000 datas pour train, 10000 pour test

# Affichage de 100 donnees mnist
fig, axs = plt.subplots(10, 10, figsize = (12, 12))
plt.gray()

# Boucle sur les sous-graphes pour ajouter les images mnist
for i, ax in enumerate(axs.flat):
    ax.imshow(x_train[i])
    ax.axis('off')
    ax.set_title('Number {}'.format(y_train[i]))
```

```

plt.subplots_adjust(hspace=0.4)

plt.show()

# Normalisation des donnees
X = x_train.reshape(len(x_train),-1)/255
Xtest = x_test.reshape(len(x_test),-1)/255
Y = y_train
Ytest = y_test


```

D Application de K-means sur mnist

```
# Application de K-means pour K=10 avec sklearn
K = 10
n = len(x_train)
km = sklearn.cluster.KMeans(n_clusters=K,init='k-means++',verbose=2,random_state
    =2,n_init = 1).fit(X) #verbose = 2 pour afficher la variance

# Affichage des resultats metriques du K-means sur les donnees train
print("Resultats sur les donnees d'entrainement:")
print("Labels:", km.labels_)
print("Inertie: ",km.inertia_)
print("Variance: ", km.inertia_/n)
print("Score (oppose de l'inertie):", km.score(X))

# Representation graphique des centroides trouves

clusterCenters = km.cluster_centers_  # Liste des centroides finaux

print("Centroids: ")
centroids = [np.reshape(a,(28,28)) for a in clusterCenters]
centroids *= 255
fig, axs = plt.subplots(1, 10, figsize = (12, 12))
plt.gray()

# Boucle sur les sous-graphes pour ajouter les images mnist
for i, ax in enumerate(axs.flat):
    ax.imshow(centroids[i])
    ax.axis('off')

# Affiche la figure
plt.show()

```

E Attribution d'étiquettes pour mnist et calcul de l'accuracy score

```
def infer_cluster_labels(km, actual_labels):
    """
    Infere les etiquettes des clusters a partir des etiquettes reelles des
    points.

    Args:
```

```
        km (sklearn.cluster.KMeans): Objet KMeans contenant les resultats de l'algorithme.
        actual_labels (array-like): Tableau des etiquettes reelles des points.

    Returns:
        dict: Dictionnaire des etiquettes predites pour chaque cluster.

    """
    inferred_labels = {}

    # Boucle sur les clusters
    for i in range(km.n_clusters):

        # Trouve les indices des points du cluster
        labels = []
        index = np.where(km.labels_ == i)

        # Ajoute les vrais labels pour chaque point du cluster
        labels.append(actual_labels[index])

        # Determine le label le plus recurrent
        if len(labels[0]) == 1:
            counts = np.bincount(labels[0])
        else:
            counts = np.bincount(np.squeeze(labels))

        # Assigne le cluster a une valeur dans le dictionnaire inferred_labels
        if np.argmax(counts) in inferred_labels:
            # Ajoute le nouveau chiffre au tableau existant a cet emplacement
            inferred_labels[np.argmax(counts)].append(i)
        else:
            # Cree le nouveau tableau a cet emplacement
            inferred_labels[np.argmax(counts)] = [i]

    return inferred_labels

def infer_data_labels(X_labels, cluster_labels):
        """
        Infere les etiquettes de chaque tableau en fonction du cluster auquel il a ete attribue.

        Args:
            X_labels (array-like): Tableau des etiquettes reelles de chaque point.
            cluster_labels (dict): Dictionnaire des etiquettes predites pour chaque cluster.

        Returns:
            array-like: Tableau des etiquettes predites pour chaque point.

        """

        # tableau vide de taille len(X_labels)
        predicted_labels = np.zeros(len(X_labels)).astype(np.uint8)

        for i, cluster in enumerate(X_labels):
            for key, value in cluster_labels.items():
                if cluster in value:
                    predicted_labels[i] = key

        return predicted_labels

```

```
64
65 kmeansLabels = km.predict(X)
66 cluster_labels = infer_cluster_labels(km, y_train)
67 predicted_labels = infer_data_labels(kmeansLabels, cluster_labels)
68 print(predicted_labels[:20])
69 print(y_train[:20])
70 print("Predicted labels:", np.unique(predicted_labels))
71 print("Accuracy: ", accuracy_score(predicted_labels,y_train))
72
```

Références

[AV06] David Arthur and Sergei Vassilvitskii. k-means++ : The advantages of careful seeding. Technical report, Stanford, 2006.

[Diva] Vincent Divol. Clustering methods. https://vincentdivol.github.io/math_tools_22/ch5.pdf.

[Divb] Vincent Divol. Convex optimization. https://vincentdivol.github.io/math_tools_22/ch2.pdf.

[GL07] Siegfried Graf and Harald Luschgy. *Foundations of quantization for probability distributions*. Springer, 2007.

[Kie83] J. Kieffer. Uniqueness of locally optimal quantizer for log-concave density and convex error weighting function. *IEEE Transactions on Information Theory*, 29(1) :42–47, 1983.

[Pol82] David Pollard. Quantization and the method of k-means. *IEEE Transactions on Information theory*, 28(2) :199–205, 1982.

[Sha] Roshan Sharma. Mnist-using-k-means. https://github.com/sharmaroshan/MNIST-Using-K-means/.

[Wik23a] Wikipedia. Astuce du noyau — Wikipedia, the free encyclopedia. http://fr.wikipedia.org/w/index.php?title=Astuce%20du%20noyau&oldid=199309107, 2023. [Online ; accessed 08-May-2023].

[Wik23b] Wikipedia. Similarity (geometry) — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Similarity%20(geometry)&oldid=1146908235, 2023. [Online ; accessed 06-May-2023].