UNIVERSITÉ DU QUÉBEC

MÉMOIRE PRÉSENTÉ À
L'UNIVERSITÉ DU QUÉBEC À TROIS-RIVIÈRES

COMME EXIGENCE PARTIELLE
DE LA MAÎTRISE EN MATHÉMATIQUES ET INFORMATIQUE APPLIQUÉES

PAR
NORBERT BERTRAND SANKA

ÉTUDE COMPARATIVE ET CHOIX OPTIMAL DU NOMBRE DE CLASSES EN CLASSIFICATION ET RÉSEAUX DE NEURONES : APPLICATION EN SCIENCE DES DONNÉES

Mars 2021

Résumé

Ce mémoire s'intéresse spécifiquement à l'apprentissage statistique non supervisé (clustering) et aux relations qui existent entre les différentes approches de clustering à savoir les méthodes classiques, l'aspect probabiliste de la classification et les méthodes neuronales. La description de chacune de ces méthodes permettra de bien distinguer les bases de chacune d'elles. Des applications sur des jeux de données variés sont développées à cet effet pour les mettre en pratique et mieux évaluer leurs performances.

Nous avons exploré différentes méthodes de classification automatique pour mieux analyser nos différents jeux de données à savoir Mnist et Mybasket. Les résultats obtenus pour l'ensemble des méthodes de classification non supervisée sont assez bons dans l'ensemble. Mais les méthodes neuronales donnent de meilleures performances pour les données volumineuses et présentent beaucoup plus de similarité avec les méthodes classiques non hiérarchiques, telles que les k-means et les k-médoides, qu'avec les modèles de mélange de densités.

Norbert Bertrand Sanka

Nadia Ghazzali

Abstract

This thesis focuses specifically on unsupervised learning (clustering) and the relationships that exist between the different clustering approaches, namely classical methods, the probabilistic aspect of classification and neural methods. The description of each of these methods will make it possible to clearly distinguish the bases of each of them. Applications on various datasets are developed for this purpose to put them into practice and better assess their performance.

We have explored different automatic classification methods to better analyze our different datasets namely Mnist and the Mybasket. The results obtained for all unsupervised classification methods are quite good overall. But neural methods give better performance for large datasets and show much more similarity to classical non hierarchical methods (such as k-means and k-medoids) than to density mixing models.

Avant-propos

Au terme de la rédaction de ce présent mémoire, je souhaiterais adresser mes remerciements à ceux qui de près ou de loin ont participé à sa réalisation. A cet effet, je tiens à adresser mes plus vifs remerciements à ma directrice de recherche, Madame Nadia Ghazzali, pour ses recommandations et le soutien indéfectible qu'elle a manifesté à mon égard. Toute ma reconnaissance au personnel administratif et au corps professoral du département de mathématiques et d'informatique pour la qualité des enseignements et la patience dont ils ont fait preuve. Je tiens à remercier ma famille, pour leur soutien moral et leurs encouragements. Je ne saurais terminer sans adresser mes remerciements à tous les étudiants avec qui j'ai eu à collaborer et plus précisément à ceux du laboratoire de statistique pour l'esprit de fraternité et de solidarité qu'on a eu à partager.

Table des matières

Résumé **ii**

Abstract **iii**

Avant-propos **iv**

Table des matières **v**

Introduction **1**

1 Méthodes de classification classiques **3**

1.1 Méthodes hiérarchiques 3

1.1.1 Classification ascendante hiérarchique (CAH) 4

1.1.2 Classification descendante hiérarchique 6

1.1.3 Mise en application 6

1.2 Méthodes non hiérarchiques 8

1.2.1 Méthode des k-means 9

1.2.2 Méthode des k-médoïdes 10

1.2.3 Nuées dynamiques 11

1.3 Choix optimal du nombre de classes 13

1.3.1 Présentation du package NbClust 14

1.3.2 Mise en application 15

2 Classification par mélange de densités **18**

2.1 Présentation générale 19

2.2 Estimation des paramètres par EM 21
2.3 Choix de modèles 22

3 Méthode neuronale : Cartes auto-organisatrices (SOM) 26
3.1 Principe 27
3.2 Algorithmes d'apprentissage 28
3.2.1 Apprentissage séquentiel 29
3.2.2 Apprentissage en mode différé 31
3.3 Évaluation de performance 32
3.4 Choix des paramètres du réseau 33
3.5 Critères de convergence 34

4 Application sur différents jeux de données 36
4.1 Étude statistique du jeu de données «Mnist» 36
4.1.1 Application de la méthode k-means dans la base de données Mnist 37
4.1.2 Application des modèles de mélange de densités dans la base de données Mnist 39
4.1.3 Application des cartes auto-organisatrices (SOM) dans la base de données Mnist. 42
4.2 Étude statistique du jeu de données «Mybasket» 46
4.2.1 Application de la méthode k-médoïdes 47
4.2.2 Application des modèles de mélange de densités 47
4.2.3 Application de la méthode SOM 48

Conclusion et perspectives 52

Bibliographie 54

annexeA 57

annexeB 59

Liste des tableaux

1.1 19 indices existants dans des packages SAS et R 13
1.2 Liste des trente (30) indices implémentés dans le package NbClust . . . 15

2.1 Caractéristiques géométriques des quatorze (14) modèles de mélange gaussien obtenues à partir de la paramétrisation de la matrice de covariance 25

4.1 Tableau récapitulatif sur le taux de bonne classification 46
4.2 Formes fortes résultant de la méthode des k-médoïdes et des modèles de mélange 49
4.3 Formes fortes résultant de la méthode des k-médoïdes et des cartes auto-organisatrices 49
4.4 Formes fortes résultant des modèles de mélange et des cartes auto-organisatrices 50
4.5 Tableau récapitulatif des formes fortes 50

Table des figures

1.1 *Illustration de la résistance des médoïdes aux points atypiques (Tirée des tutoriels de Ricco Raccotomalala, Algorithme des k- médoïdes p.12)* 11

1.2 *Données simulées avec 3 classes* 16

1.3 *Méthodes graphiques pour déterminer le meilleur nombre de classes* . . 17

3.1 *Schéma d'une carte auto-organisée de Kohonen (1984).* 27

3.2 *Mise à jour des vecteurs prototypes (tirée de Chantal Hajjar (avril 2015). Cartes auto-organisatrices pour la classification de données symboliques mixtes, de données de type intervalle et de données discrétisées, p.28).* 29

3.3 *Régions de Voronoï avec une structure de voisinage* 31

4.1 *Images des centres de classe pour les 10 classes identifiées par k-means.* 38

4.2 *Matrice de confusion illustrant comment l'algorithme k-means a regroupé les chiffres prédits et les étiquettes réelles* 38

4.3 *Résumé des résultats de la classification par mélange de densité* 40

4.4 *Images des centres de classe pour les 10 classes identifiées par mélange de densités* 40

4.5 *Matrice de confusion illustrant comment l'algorithme EM a regroupé les chiffres prédits et les étiquettes réelles* 41

4.6 *Carte coloriée en fonction des effectifs des neurones* 43

4.7 *Distance au voisinage* 43

4.8 *Représentation des classes dans la carte topologique* 44

4.9 *Images des centres de classe pour les 10 classes identifiées par la carte SOM* 45
4.10 *Matrice de confusion illustrant comment l'algorithme SOM a regroupé les chiffres prédits et les étiquettes réelles* 45
4.11 *résumé des résultats de la classification par mélange de densité* 48
4.12 *Interprétation géométrique de chacun des quatorze modèles gaussiens avec trois classes en deux dimensions* 57
4.13 *Nombre de classes optimal du jeu de données "my basket"* 57
4.14 *- Représentation des classes dans la carte topologique* 58

Introduction

La classification consiste à regrouper les individus qui se ressemblent au vue des variables de l'analyse. On cherche donc à former des classes homogènes à l'intérieur et hétérogènes entres elles. A priori on ne connait pas le nombre de classes à former sauf dans le cas où les données présentent des classes naturelles. Il existe deux catégories de classification : la classification supervisée (ou analyse discriminante) et la classification non supervisée (ou classification automatique). Dans la première catégorie, on essaye de bien séparer les classes et prédire la meilleure classe pour chaque nouvel individu, étant donné un ensemble de classes prédéfinies. Alors que la classification non supervisée consiste à identifier des classes homogènes dans un ensemble de données. Nous nous limiterons dans ce mémoire aux méthodes de classification non supervisée.

La classification non supervisée de données (clustering) vise à développer notamment des programmes informatiques qui apprennent automatiquement avec les données traitées, l'objectif principal étant de regrouper des données en classes partageant des caractéristiques similaires sans aucune connaissance préalable. Ces classes peuvent être définies suivant les individus (on parlera de regroupement) ou suivant les variables (on parlera de réduction de dimension).

La plupart des algorithmes de classification non supervisées font appel à différents paramètres d'entrée tels que le choix du nombre de classes ou l'espace de représentation des données. A cet effet, plusieurs d'entre elles se basent sur des représentations

vectorielles qui, en plus d'accepter différentes métriques, permettent de définir des centres de gravité de groupes, des densités de probabilité, des intervalles, des sous-espaces, etc.

Ce mémoire est réparti en quatre chapitres. Le premier présente les méthodes classiques que sont la classification hiérarchique et celle non hiérarchique. Ce chapitre traite également du problème de choix optimal du nombre de classes. Le deuxième chapitre aborde l'aspect probabiliste de la classification par les modèles de mélanges de densité. Le troisième chapitre traite d'une méthode neuronale dite méthode de Kohonen ou encore Self Organizing Maps (SOM). Le dernier chapitre quant à lui présente une étude comparative des différentes méthodes citées précédemment sur deux jeux de données. La raison principale du choix de ces différentes méthodes de clustering est due au fait de leur popularité, leur souplesse, mais également de leur applicabilité à de grands jeux de données.

Chapitre 1

Méthodes de classification classiques

Dans la littérature scientifique, plusieurs méthodes de classification ont été présentées. Dans ce chapitre nous allons présenter les plus connues. Nous passerons notamment en revue la classification ascendante hiérarchique et celle non hiérarchique. Nous y aborderons également le problème du choix optimal du nombre de classes. En effet, ce choix est assez complexe du fait que cela s'opère dans un contexte d'apprentissage non supervisé (donc aucune connaissance préalable sur les données). Si les paramètres de l'algorithme de classification sont incorrects, ce dernier peut mener à une mauvaise qualité du partitionnement.

1.1 Méthodes hiérarchiques

Les méthodes hiérarchiques produisent une hiérarchie de classes, représentée sous forme d'arbre de classification nommé dendrogramme. Ce dernier montre comment

les classes sont organisées selon certains critères. Les méthodes de classification hiérarchiques sont divisées en deux types d'approches : ascendante, dite agglomérative et descendante, dite divisive.

1.1.1 Classification ascendante hiérarchique (CAH)

Il s'agit de regrouper itérativement les individus, en commençant par le bas (les deux plus proches) et en construisant progressivement un dendrogramme, regroupant finalement tous les individus en une seule classe. Ceci suppose de savoir calculer, à chaque étape ou regroupement, une mesure d'éloignement entre les individus i et j, mais également une distance entre les classes contenant les individus D(A,B).

Notons $\Omega = \{1, .., i, .., n\}$ l'ensemble des individus. Il existe différentes mesures d'éloignement entre individus :

- Indice de ressemblance ou similarité

  C'est une mesure de proximité définie de Ω^2 dans R_+ et vérifiant :

  $\forall (i, j) \in \Omega^2$

  - $s(i, j) = s(j, i)$: symétrie ;

  - $s(i, i) = S > 0$, : ressemblance d'un individu avec lui-même ;

  - $s(i, j) \leq S$: la ressemblance est majorée par S.

- Indice de dissemblance ou dissimilarité

  Les notions de similarité et dissimilarité se correspondent de façon élémentaire. Si s est un indice de ressemblance, alors $d(i, j) = S - s(i, j), \forall (i, j) \in \Omega^2$ est un indice de dissemblance.

  Une dissimilarité est une application de Ω^2 dans R_+ vérifiant :

  $\forall (i, j) \in \Omega^2$

  - $d(i, j) = d(j, i)$;

- $d(i,j) = 0 \Leftrightarrow i = j$.

- Distance

  Une distance sur Ω est, par définition, une dissimilarité vérifiant en plus la propriété d'inégalité triangulaire. Autrement dit, une distance d est une application de Ω^2 dans R_+ vérifiant :

  $\forall (i,j,k) \in \Omega^3$.

  - $d(i,j) = d(j,i)$;

  - $d(i,j) = 0 \Leftrightarrow i = j$;

  - $d(i,j) \leq d(i,k) + d(j,k)$.

La mesure d'éloignement entre deux ensembles est appelée indice d'agrégation. Notons A et B deux classes d'une partition donnée, w_A et w_B leurs pondérations, g_A et g_B leurs barycentres et d(i,i') la distance entre deux individus quelconques i et i'. Différents indices d'agrégation sont utilisés :

- Méthode du plus proche voisin (saut minimum) : $D(A,B) = min\{d(i,i')\}$ avec $i \in A$ et $i' \in B$

- Méthode du voisin le plus éloigné (saut maximum) : $D(A,B) = max\{d(i,i')\}$ avec $i \in A$ et $i' \in B$

- Méthode du saut moyen : $D(A,B) = \frac{1}{w_A . w_B} \sum_{i \in A, i' \in B} d(i,i')$

- Méthode du centroide (distance des barycentres) : $D(A,B) = d(g_A, g_B)$

- Méthode de Ward : $D(A,B) = \frac{w_A . w_B}{w_A + w_B} d(g_A, g_B)$

Dans sa version la plus simple, le principe de la CAH est le suivant :

— Etape 0 : Chaque individu forme sa propre classe.

— Etape1 : On regroupe les deux individus les plus proches. On aura donc $n-1$ classes où n est le nombre d'individus.

— Etape k : Ainsi de suite, en regroupant les deux classes les plus proches on aura $(n-k)$ classes à l'étape k.

— Etape n-1 : A cette étape on aura une seule classe contenant tous les individus.

1.1.2 Classification descendante hiérarchique

La méthode de classification ascendante part d'un ensemble où chaque individu forme sa propre classe et procède par regroupement binaire et multiple alors que la méthode descendante fait le contraire. Elle consiste à construire une hiérarchie à partir d'une seule classe regroupant tous les objets, puis procède par divisions successives des classes jusqu'à l'obtention de classes vérifiant certaines règles d'arrêt. Les méthodes descendantes sont plus rapides, mais moins efficaces. En fait, la complexité d'un algorithme ascendant est généralement polynomiale, tandis que celle d'un algorithme descendant est généralement exponentielle. En effet, lors de la première étape d'une méthode ascendante, il faut évaluer toutes les agrégations possibles de deux individus parmi n, soit n(n - 1)/2 possibilités, tandis qu'un algorithme descendant basé sur l'énumération complète évalue toutes les divisions des n individus en deux sous-ensembles non vides, soit $2^{n-1}-1$ possibilités. Dans ce mémoire nous nous intéressons à la CAH.

1.1.3 Mise en application

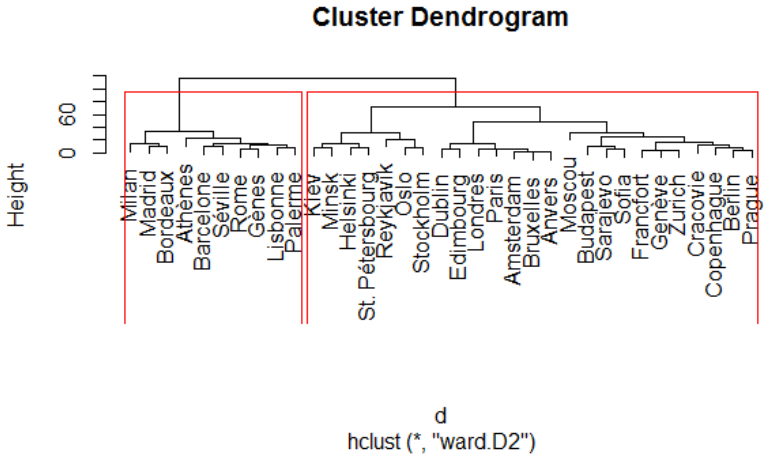
Pour l'application de la CAH, on s'intéresse au climat des différents pays d'Europe. Ainsi, notre jeu de données [32] présente les températures moyennes mensuelles (en degrés Celsius) pour les principales capitales européennes ainsi que pour certaines grandes villes (35 au total). En plus des températures mensuelles, on donne pour chaque ville, la température moyenne annuelle ainsi que l'amplitude thermique (diffé-

rence entre la moyenne mensuelle maximum et la moyenne mensuelle minimum d'une ville). On donne également deux variables de positionnement (la longitude et la latitude). La variable qualitative donnant l'appartenance à une région d'Europe (variable à quatre modalités : Europe du nord, du sud, de l'est, de l'ouest) nous permettra de vérifier la qualité du partitionnement.

Dans le logiciel R, nous utiliserons la fonction ≪ *hclust* ≫ pour réaliser la classification hiérarchique sur nos données. Cette fonction permet en effet de réaliser une CAH à partir d'un tableau de distances entre individus. Et elle fournit en plus des résultats de la classification, le dendrogramme.
≪ *hclust* ≫ prend deux principaux arguments que sont la matrice de distance et la méthode d'agrégation. La matrice de distance peut être obtenu sous le logiciel R à l'aide de la commande ≪ *dist* ≫. Cette dernière utilise, par défaut, la distance euclidienne pour le calcul de la matrice de distance entre individus. Toutefois, il est possible de spécifier le type de dissimilarité ou distance que l'on souhaite utiliser. On peut spécifier aussi le critère d'agrégation entre les classes. Par défaut, ≪ *hclust* ≫ utilise la méthode ≪ *complete* ≫ correspondant au critère du saut maximum. De même, il est possible de choisir d'autres critères d'agrégation tels que définis précédemment.

- **Script R**

```
temperat <- read.csv("C :/Users/ucer/Downloads/temperat.csv", header=TRUE,
sep=";", dec=".", row.names=1)
d<-dist(temperat)
cah<-hclust(d, method="ward.D2")
plot(cah)
cut1<-cutree(cah, k=2)
cut2<-cutree(cah, k=3)
rect.hclust(cah, k=2)
```



1.2 Méthodes non hiérarchiques

Elles permettent de traiter les grands ensembles de données. Ces méthodes optimisent un critère de type inertie. Le principe général de ces méthodes se présente comme suit :

1. On fixe le nombre de classes q de la partition.
2. On part d'une partition P_0 en q classes qu'on choisit au hasard ou par connaissance des données.
3. Ensuite, on cherche à améliorer la partition initiale de manière à minimiser la variance intra-classe.

Il existe plusieurs types de partitionnement dont les approches se différencient dans la définition des représentants des classes. Parmi les plus utilisés, on peut citer les k-means (MacQueen, 1967), les k-médoïdes (Kaufman and Rousseeuw, 1987) et les nuées dynamiques (Diday, 1971).

1.2.1 Méthode des k-means

La méthode des k-means est un type d'apprentissage non supervisé, qui est utilisé lorsqu'on a des données non étiquetées (c'est-à-dire des données sans catégories ni groupes définis). C'est une méthode efficace qui permet de diviser un ensemble de données en k classes homogènes. L'algorithme fonctionne de manière itérative pour affecter chaque point de données à l'un des k groupes en fonction des caractéristiques fournies. Les classes de données sont regroupées en fonction de la similarité des éléments.

Le fonctionnement de l'algorithme des k-means se résume dans les étapes suivantes :

- Etape1 : Fixer a priori le nombre de classes k ;
- Etape2 : Initialiser k centres de classes G_k (Peut être k individus choisis au hasard. Ou encore, k moyennes calculées à partir d'une partition au hasard ou par classification ascendante hiérarchique des individus en k classes) ;
- Etape3 : Affecter chaque individu à la classe dont le centre est le plus proche ;
- Etape4 : Recalculer les centres de classes à chaque affectation ou à la fin d'une itération.

Ainsi, les étapes 3 et 4 se répètent donc jusqu'à ce que l'algorithme converge, c'est-à-dire jusqu'à ce que les objets ne changent plus de classe ou après un nombre fixé d'itérations.

L'avantage avec cette méthode est qu'en plus d'être simple à mettre en œuvre, elle a la capacité de traiter les grandes bases de données. Seuls les vecteurs des moyennes sont à conserver en mémoire centrale. Toutefois, elle présente aussi des limites car le nombre de classe doit être fixé au départ, le résultat dépend de la configuration initiale des centres des classes, et les classes sont construites par rapport à des objets potentiellement inexistants (les centres).

1.2.2 Méthode des k-médoïdes

La méthode des k-médoïdes est une technique de classification automatique par réallocation. Sa particularité repose essentiellement sur la notion de médoïde, qui correspond au point représentatif d'un ensemble d'observations associées à un groupe. Si les classes sont relativement denses, sa position dans l'espace de représentation est très proche du centre de gravité. En revanche, pour les classes avec des individus dispersés, ou en présence de points atypiques, cette méthode se révèle autrement plus robuste en dépassant le caractère artificiel du barycentre (voir Figure 1.1).

Dans cet exemple, on peut remarquer que pour les k-médoïdes, au lieu de prendre la valeur moyenne des objets dans une classe pour centroïde (qui est un objet fictif), nous prenons plutôt l'objet le plus central de la classe.

Le fonctionnement de l'algorithme des k-médoïdes se résume dans les étapes suivantes :

- Etape1 : Fixer a priori le nombre de classes k ;
- Etape2 : Initialiser k médoïdes M_k (Peut être k individus choisis au hasard. Ou encore, k points les moins distants des autres) ;
- Etape3 : Affecter chaque individu à la classe dont le médoïde est le plus proche ;
- Etape4 : Recalculer les médoïdes de classes à partir des individus rattachés ;

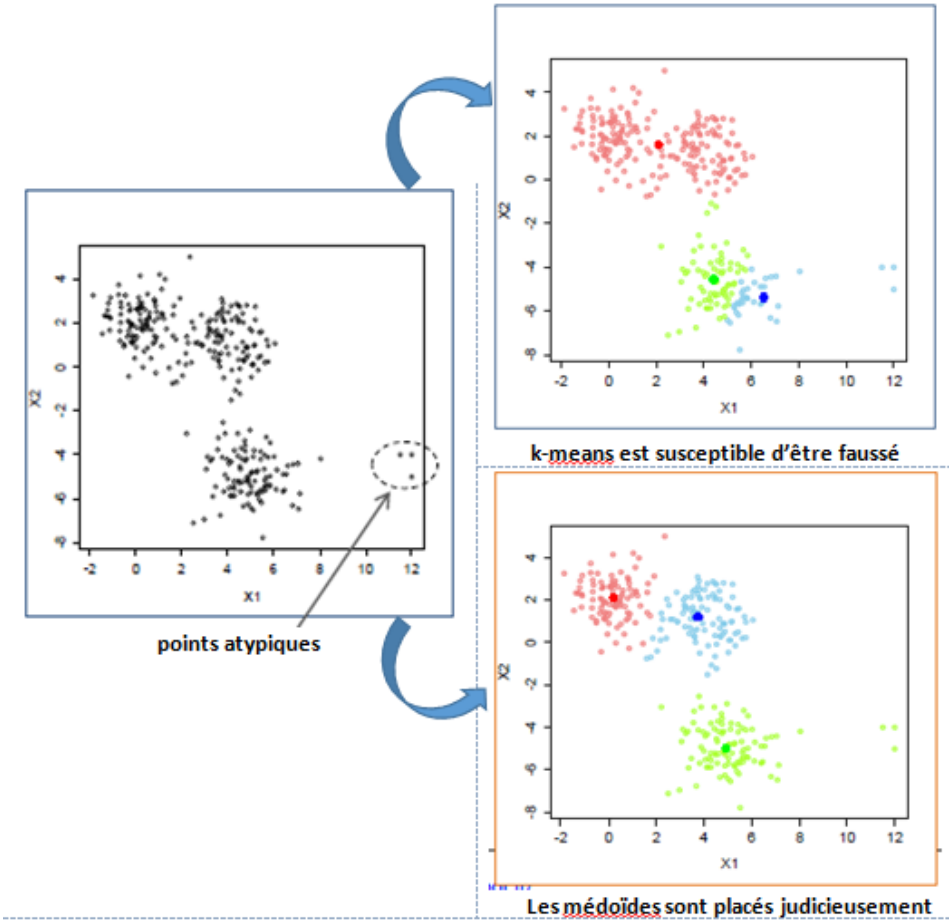


FIGURE 1.1 – *Illustration de la résistance des médoïdes aux points atypiques (Tirée des tutoriels de Ricco Raccotomalala, Algorithme des k- médoïdes p.12)*

Ainsi, les étapes 3 et 4 se répètent donc jusqu'à ce que l'algorithme converge, c'est-à-dire jusqu'à ce que les objets ne changent plus de classe ou après un nombre fixé d'itérations.

1.2.3 Nuées dynamiques

L'algorithme des nuées dynamiques est une généralisation de l'algorithme des k-means, où chaque classe est représentée non pas par son barycentre (qui peut être extérieur à la classe), mais par un noyau, qui peut être un sous-ensemble de points

de la classe, un axe principal ou un plan principal, etc. Ce noyau s'avère parfois plus représentatif de la classe.

La présentation qu'en donne Michel Volle (Volle M., 1978) convient parfaitement pour expliquer les principes de la méthode et introduire les concepts qui lui sont propres. Supposons un ensemble de données Ω sur lequel sont définies une distance $d(x, y)$ entre éléments et une distance $D(X, Y)$ entre classes. Prenons au hasard k sous-ensembles de p points chacun dans Ω ; nous appellerons chacun de ces sous-ensembles un « noyau ». Ces noyaux nous permettent de définir une partition de Ω en k classes, chaque classe comprenant les éléments qui sont plus proches d'un des noyaux que de tous les autres noyaux. A partir de cette partition, on définit une nouvelle famille de noyaux en associant à chaque classe de la partition l'ensemble de p points qui en est le plus proche. Puis on recommence : à cette nouvelle famille de noyaux va être associée une nouvelle partition, etc. Il est facile de démontrer que le procédé converge sous certaines conditions : on finit par aboutir à une partition et à une famille de noyaux optimaux. Ainsi la méthode des nuées dynamiques permet de construire par itération, à partir d'une famille absolument quelconque de noyaux, une partition en k classes. Mais cette construction contient une part d'arbitraire : la partition obtenue dépend du choix initial des noyaux. Pour compenser dans une certaine mesure cet arbitraire, on applique la méthode plusieurs fois de suite, en partant à chaque fois d'une famille différente de noyaux tirée au hasard. On obtient ainsi plusieurs partitions en k classes. Si l'on considère deux éléments quelconques, ils peuvent avoir été classés ensemble dans certaines de ces partitions, et classés séparément dans d'autres. On appelle forme forte un ensemble d'éléments qui auront été classés ensemble dans toutes les classifications. Ces formes fortes déterminent une partition de E qui peut fort bien contenir plus de k classes. Les formes fortes qui ne comprennent qu'un élément ne présentent pas d'intérêt, car elles concernent des éléments qui semblent « inclassables ». Par contre, les formes fortes qui comprennent beaucoup d'éléments sont intéressantes : pour qu'un groupe d'éléments ait résisté aux aléas dus aux choix des différentes familles de noyaux, il fallait qu'il fût bien homogène.

1.3 Choix optimal du nombre de classes

Trouver le nombre optimal de classes représente l'un des plus grand défis en classification automatique de données. En effet, on a vu que la partition finale dépendait fortement de la partition initiale dont le nombre de classes est choisi de manière arbitraire. Diverses mesures visant à valider les résultats d'une analyse de regroupement ont été définies et proposées dans la littérature. Outre l'étude comparative de Milligan et Cooper (1985)[25], qui ont examiné trente indices avec des données simulées dans un espace euclidien, où le nombre de classes est connu à l'avance, de nouveaux critères de détermination du nombre de classes ont été proposés depuis lors. Cependant, seuls quelques-uns de ces critères ont été programmés et implémentés dans les logiciels d'analyse de données tels que SAS et R (Voir Table 1.1).

SAS	**R**			
Cluster	***clusterSim***	***cclust***	***clv***	***clvalid***
1. CH (Calinski and Harabasz 1974)		11. Ratkowsky (Ratkowsky and Lance 1978)	19. Dunn (Dunn 1974)	
2. CCC (Sarle 1983)	4. KL (Krzanowski and Lai 1988)	12. Scott (Scott and Symons 1971)		
3. Pseudot2 (Duda and Hart 1973)	5. Gamma (Baker and Hubert 1975)	13. Marriot (Marriot 1971)		
	6. Gap (Tibshirani et al. 2001)	14. Ball (Ball and Hall 1965)		
	7. Silhouette (Rousseeuw 1987)	15. Trcovw (Milligan and Cooper 1985)		
		16. Tracew (Milligan and Cooper 1985)		
		17. Friedman (Friedman and Rubin 1967)		
		18. Rubin (Friedman and Rubin 1967)		
	8. Hartigan (Hartigan 1975)			
	9. Cindex (Hubert and Levin 1976)			
	10. DB (Davies and Bouldin 1979)			

TABLE 1.1 – 19 indices existants dans des packages SAS et R

C'est dans ce sens que Charrad et al.(2014)[10] ont passé en revue ces 19 indices déjà implémentés dans un même package R nommé NbClust, en y rajoutant des indices qui ne sont pas encore programmés, pour un total de 30 indices, afin de fournir à l'utilisateur une liste plus exhaustive d'indices de validation lui permettant d'estimer le bon nombre de classes dans un jeu de données.

1.3.1 Présentation du package NbClust

Le package NbClust de R est développé afin d'aider l'utilisateur à déterminer le bon nombre de classes dans un jeu de données. Trente indices de validation sont proposés dans sa version actuelle, ainsi que des méthodes hiérarchiques (CAH) et non hiérarchiques (K-means) de classification. L'avantage de ce package est qu'il permet d'appliquer simultanément plusieurs indices et de déterminer la meilleure partition parmi toutes les partitions obtenues en combinant, nombre de classes minimal et maximal, les distances (distance euclidienne, distance maximale, distance Manhattan, distance de Canberra, distance binaire ou distance de Minkowski) et les méthodes d'agrégation (Ward.D1, Ward.D2, Saut minimum, Saut maximum, Saut moyen, McQuitty, Médiane ou Centroide).
NbClust propose ainsi le nombre pertinent de classes pour chacune des trente (30) critères utilisés, soit par le nombre minimal ou maximal de l'indice, soit par comparaison de l'indice par rapport à la valeur critique, soit aussi par des méthodes graphiques. Ces dernières concernent l'indice de Hubert et celui de Dindex. Ainsi, le nombre optimal de classes pour ces deux indices est déterminé par le pic le plus significatif en observant la courbe représentant les dérivées secondes des indices.

Le choix optimal du nombre de classes sera guidé par les résultats fournis par les trente (30) indices. Autrement dit, NbClust propose d'utiliser la règle du vote majoritaire. La liste des trente (30) indices utilisés, avec leurs caractéristiques et références, est présentée à la table 1.2 (Charrad et al., 2014).

Nom de l'indice dans NbClust	**Nombre de classes optimal**
1. "ch" (Calinski and Harabasz 1974)	Valeur maximale de l'indice
2. "duda" (Duda and Hart 1973)	Plus petit nombre de classes tel que l'indice >à la valeur critique
3. "pseudot2" (Duda and Hart 1973)	Plus petit nombre de classes tel que l'indice <à la valeur critique
4. "cindex" (Hubert and Levin 1976)	Valeur minimale de l'indice
5. "gamma" (Baker and Hubert 1975)	Valeur maximale de l'indice
6. "beale" (Beale 1969)	Le nombre de classes tel que la valeur critique >= alpha
7. "ccc" (Sarle 1983)	Valeur maximale de l'indice
8. "ptbiserial" (Milligan 1980, 1981)	Valeur maximale de l'indice
9. "gplus" (Rohlf 1974 ; Milligan 1981)	Valeur minimale de l'indice
10. "db" (Davies and Bouldin 1979)	Valeur minimale de l'indice
11. "frey" (Frey and Van Groenewoud 1972)	Niveau correspondant à la classe avant que la valeur de l'indice <1.00
12. "hartigan" (Hartigan 1975)	Différence maximale entre les niveaux de hiérarchie de l'indice
13. "tau" (Rohlf 1974 ; Milligan 1981)	Valeur maximale de l'indice
14. "ratkowsky" (Ratkowsky and Lance 1978)	Valeur maximale de l'indice
15. "scott" (Scott and Symons 1971)	Différence maximale entre les niveaux de hiérarchie de l'indice
16. "marriot" (Marriot 1971)	Valeur maximale des différences secondes entre les niveaux de l'indice
17. "ball" (Ball and Hall 1965)	Différence maximale entre les niveaux de hiérarchie de l'indice
18. "trcovw" (Milligan and Cooper 1985)	Différence maximale entre les niveaux de hiérarchie de l'indice
19. "tracew" (Milligan and Cooper 1985)	Valeur maximale des différences secondes entre les niveaux de l'indice
20. "friedman" (Friedman and Rubin 1967)	Différence maximale entre les niveaux de hiérarchie de l'indice
21. "mcclain" (McClain and Rao 1975)	Valeur minimale de l'indice
22. "rubin" (Friedman and Rubin 1967)	Valeur minimale des différences secondes entre les niveaux de l'indice
23. "kl" (Krzanowski and Lai 1988)	Valeur maximale de l'indice
24. "silhouette" (Rousseeuw 1987)	Valeur maximale de l'indice
25. "gap" (Tibshirani et al. 2001)	Plus petit nombre tel que la valeur critique >= 0
26. "dindex" (Lebart et al. 2000)	Méthode graphique
27. "dunn" (Dunn 1974)	Valeur maximale de l'indice
28. "hubert" (Hubert and Arabie 1985)	Méthode graphique
29. "sdindex" (Halkidi et al. 2000)	Valeur minimale de l'indice
30. "sdbw" (Halkidi and Vazirgiannis 2001)	Valeur minimale de l'indice

TABLE 1.2 – Liste des trente (30) indices implémentés dans le package NbClust

1.3.2 Mise en application

Nous proposons dans cette section d'expliquer le fonctionnement de la librairie NbClust, à travers un jeu de données simulées composé de 2 variables et de 300

observations réparties en 3 classes distinctes.

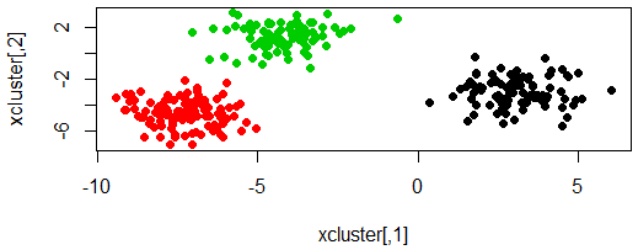


FIGURE 1.2 – *Données simulées avec 3 classes*

Dans cet exemple, nous cherchons la meilleure partition parmi toutes les partitions obtenues avec un nombre de classes qui varie entre 2 et 8, et en utilisant la distance euclidienne et la méthode d'agglomération "complete" qui correspond au saut maximum. D'autres distances sont disponibles dans la librairie NbClust comme indiqué précédemment. L'option "alllong" permet de calculer simultanément tous les indices implémentés dans NbClust sur chacune des partitions. Cependant, comme l'exécution de certains indices, à savoir : Gamma, Tau, Gap et Gplus est lente, il est possible de choisir l'option "all" afin de ne calculer que les autres indices, soit 26 parmi les 30 disponibles. NbClust permet également de tester les indices individuellement en utilisant les noms des indices présentés précédemment (voir Table 1.2).

Les sorties de NbClust sont : le nombre de classes optimal proposé selon la règle du vote majoritaire, les valeurs des indices obtenues pour chaque partition, les valeurs seuils des indices(duda, pseudoT2, beale et gap), le nombre de classes optimal proposé par chaque indice et la partition qui correspond au nombre de classes optimal.

- **Quelques résultats**

Concernant notre exemple, 20 indices parmi les 24 proposent 3 comme le nombre optimal de classes, 2 proposent 2 comme le bon nombre de classes et un seul en

```
*******************************************************************************
* Among all indices:
* 2 proposed 2 as the best number of clusters
* 20 proposed 3 as the best number of clusters
* 1 proposed 5 as the best number of clusters

                   ***** Conclusion *****

* According to the majority rule, the best number of clusters is  3

*******************************************************************************
```

propose 5 comme nombre optimal de classe. Le Dindex et l'indice de Hubert sont des méthodes graphiques. Par conséquent, ils ne sont pas comptabilisés dans le calcul du nombre de classes optimal proposé selon la règle du vote majoritaire. Pour ces deux indices, le nombre optimal de classes est identifié par un pic significatif dans le graphique des valeurs d'indice par rapport au nombre de classes. Ainsi, le nombre de classes est déterminé par le pic le plus significatif en observant la courbe de la seconde différence entre les niveaux de l'indice.

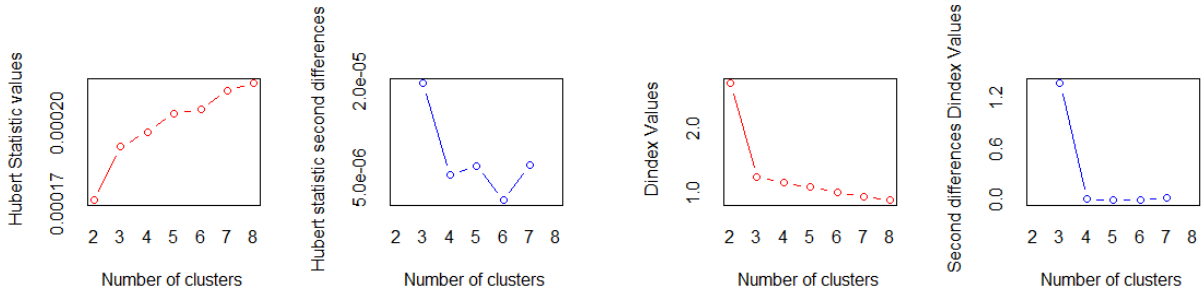


FIGURE 1.3 – *Méthodes graphiques pour déterminer le meilleur nombre de classes*

Comme le montre la figure 1.3, ces deux indices (Hubert et Dindex) proposent 3 comme meilleur nombre de classes. Finalement, 22 indices parmi les 26 proposent 3 comme le nombre optimal de classes, ce qui est réellement le bon nombre de classes dans le jeu de données simulées.

Chapitre 2

Classification par mélange de densités

Un outil clé de la résolution du problème de classification réside dans les modèles de mélange. Les modèles de mélange, apparus dans les travaux de Pearson (Pearson, 1894), sont utilisés avec succès dans bon nombre de disciplines comme l'astronomie, la biologie, la génétique, l'économie, les sciences de l'ingénieur, le marketing, la reconnaissance d'images... L'on pourrait donc se demander : quel est le rôle joué par les modèles de mélange dans la classification des données ?

D'un point de vue théorique, les modèles de mélange permettent de bénéficier de l'ensemble des résultats de la statistique mathématique : lois multivariées paramétriques (ou semi-paramétrique ou non paramétrique), estimation, choix de modèles, etc. Ils permettent aussi de retrouver, voire de généraliser, de nombreuses méthodes de classification classiques non probabilistes à la base, car plutôt géométriques. Dans ce chapitre, nous ferons une présentation du modèle de mélange de lois de densités, de l'estimation de ses paramètres et des méthodes habituelles de choix de modèles. En particulier, nous soulignerons le lien avec des méthodes géométriques classiques dans le cas gaussien et nous présenterons également des critères de choix de modèles

spécifiques.

2.1 Présentation générale

Le modèle de mélange fini de lois de probabilités consiste à supposer que les données proviennent d'une source contenant plusieurs sous-populations homogènes appelées composantes. La population totale est un mélange de ces sous-populations. Le modèle résultant est un modèle de mélange fini.

Supposons $x = (x_1, x_2, ..., x_n)$, un ensemble d'individus de taille n, et $Z =(Z_1,..., Z_n)$ leur partition en K classes : $G_1, G_2, ..., G_K$ avec $Z_i = k$ si i provient du groupe k. Si on échantillonnait dans une population formée de K sous-populations, on devrait avoir les couples (X_i, Z_i) où $X_i = x_i$ représente la mesure faite sur le ième individu et $Z_i = k$ indique le numéro de la sous-population à laquelle appartient cet individu.

Soit $X = (X_1, X_2, ..., X_n)$ un échantillon de variables aléatoires indépendantes identiquement distribués (iid) de loi de mélange fini à K composantes, la densité de probabilité $f(x)$ du mélange peut s'exprimer comme la somme pondérée des autres densités f_k :

$$f(x, \Theta) = \sum_{k=1}^{K} \Pi_k f_k(x, \Theta_k)$$

Avec Π_k, les proportions respectives des sous populations telle que : $0 < \Pi_k \leq 1$ et $\sum_{k=1}^{K} \Pi_k = 1$

f_k, la densité de la $k$ième composante (la paramétrisation des densités des composantes dépend de la nature continue ou discrète des données observées). Et Θ est le vecteur des paramètres du modèle de mélange.

L'approche la plus connue des lois de mélange est le modèle de mélange gaussien où les densités de base sont des lois normales multidimensionnelles. Le modèle de mélange gaussien est une combinaison linéaire de plusieurs composantes gaussiennes. Il est particulièrement utilisé dans le cas où les données en études ne peuvent pas être modélisées par une simple gaussienne. En d'autres termes, si la structure de données est composée naturellement par plusieurs groupes, il faut les représenter par un modèle de mélange gaussien plutôt qu'une simple distribution gaussienne. Pour le formaliser, il suffit juste de remplacer chaque $f(x, \Theta)$ par la fonction de densité de la loi gaussienne. Ici, chaque classe est décrite par une loi de distribution normale, paramétrée par sa moyenne μ_k et sa matrice de variance-covariance Σ_k. La fonction de densité de la loi gaussienne s'écrit alors comme suit :

$$f(x, \Theta) = \sum_{k=1}^{K} \Pi_k \mathcal{N}(x|\mu_k, \Sigma_k)$$

Avec : $\mathcal{N}(x|\mu_k, \Sigma_k) = \frac{1}{(2\pi)^{\frac{d}{2}} |\Sigma_k|^{\frac{1}{2}}} e^{\frac{-1}{2}(x-\mu_k)^T \Sigma_k^{-1}(x-\mu_k)}$

où μ_k est un vecteur de d-dimension, Σ_k est la matrice de covariance de dimension d^2, et $|\Sigma_k|$ désigne le déterminant de la matrice Σ_k.

Dans le contexte des modèles de mélange, le problème de la classification peut être traité en utilisant la maximisation de la vraisemblance des données observées. L'idée fondamentale de l'estimation par maximum de vraisemblance est, comme son nom l'indique, de trouver un ensemble d'estimations des paramètres, pour que la vraisemblance de l'échantillon utilisé, soit maximum. Étant donné la même hypothèse, que les donnés sont des réalisations indépendantes d'un vecteur aléatoire X, la fonction de vraisemblance des données relativement au modèle de paramètre Θ s'écrit :

$$L(X, \Theta) = \prod_{i=1}^{n} f(x_i, \Theta)$$

Cependant, il est plus facile de maximiser la log-vraisemblance au lieu de la fonction de vraisemblance elle-même. Si Θ maximise $ln(L(\Theta))$, alors il maximise également $L(\Theta)$. Cela est dû à la monotonie de la fonction logarithme. La fonction log-vraisemblance s'écrit :

$$ln(L(X,\Theta)) = \sum_{i=1}^{n} ln(f(x_i,\Theta))$$

Le problème d'estimation par la méthode de maximum de vraisemblance, revient donc à trouver les racines de l'équation :

$$\frac{\partial ln(L(X,\Theta))}{\partial \Theta} = 0$$

En général, la maximisation de la fonction de vraisemblance ne possède pas de solution analytique. C'est pour cela qu'il est nécessaire de recourir à des méthodes itératives. De nombreuses techniques classiques d'optimisation peuvent être appliquées, mais dans notre cas, nous utiliserons l'algorithme EM (Dempster et al., 1977) qui est sans doute le plus utilisé en raison de ses propriétés de convergence et de sa simplicité de mise en œuvre.

2.2 Estimation des paramètres par EM

L'algorithme espérance-maximisation (en anglais Expectation-maximisation algorithm, souvent abrégé EM), proposé par Dempster et al. (1977) [12], est une classe d'algorithmes qui permettent de trouver le maximum de vraisemblance des paramètres de modèles probabilistes lorsque le modèle dépend de variables qui ne sont pas ob-

servées directement, mais qui sont plutôt déduites d'autres variables observées. On utilise souvent l'algorithme EM pour la classification de données, l'apprentissage automatique, ou la vision artificielle. L'algorithme d'espérance-maximisation comporte :

— une étape d'évaluation de l'espérance (E), où l'on calcule l'espérance de la vraisemblance en tenant compte des dernières variables observées,

— une étape de maximisation (M), où l'on estime le maximum de vraisemblance des paramètres en maximisant la vraisemblance trouvée à l'étape E.

On utilise ensuite les paramètres trouvés en M comme point de départ d'une nouvelle phase d'évaluation de l'espérance, et l'on itère ainsi. L'algorithme s'arrête après un nombre prédéfini d'itérations ou bien à la stationnarité du critère de log-vraisemblance observée. Cette seconde option découle de la propriété de croissance de la log-vraisemblance à chaque itération.

Il est cependant fréquent que la structure de modélisation d'où sont issues les données x soit elle-même incertaine. Par exemple, on ignore le nombre K de composantes du modèle et ce nombre doit donc être lui aussi estimé. D'où la nécessite de choix du modèle adéquat. De façon générique, un modèle m représente un ensemble de contraintes sur l'espace du paramètre Θ.

2.3 Choix de modèles

Pour choisir parmi plusieurs modèles en compétition, on peut utiliser des critères très classiques en statistique mathématique qui s'expriment généralement comme une pénalisation de la log-vraisemblance maximale de la forme :

$$*IC_m = L(\widehat{\Theta}_m; x) - \upsilon_m \times f(n)$$

où $\widehat{\Theta}_m$ correspond à l'estimateur du maximum de la vraisemblance sous le modèle m, υ_m donne le nombre de paramètres à estimer et f est une fonction de n. On retient alors le modèle ayant obtenu la plus grande valeur du critère. Fondamentalement, ce type de critère tente de réaliser un compromis entre l'adéquation du modèle aux données mesurées par la log-vraisemblance maximale et la complexité de celui-ci, mesurée par sa dimension.

- Du point de vue fréquentiste, un « bon » modèle est celui qui réalise un compromis en terme de « biais-variance ». Dans ce cadre, le critère AIC (An Information Criterion)[3] est proposé avec $f(n) = 1$. La pénalité ne dépend donc pas de n.

$$AIC = L(\widehat{\Theta}_m; x) - \upsilon_m$$

- Du point de vue bayésien, un « bon » modèle est celui qui maximise la vraisemblance intégrée (sur les paramètres) lorsque chaque modèle en compétition est équiprobable a priori. Le critère BIC (Bayesian Information Criterion) propose alors la pénalité $f(n) = 0.5ln(n)$, dépendant cette fois de n.

$$BIC = L(\widehat{\Theta}_m; x) - \upsilon_m 0.5ln(n)$$

Tous deux sont des critères dits asymptotiques en le sens que les propriétés idéales dont ils sont issues sont atteintes seulement asymptotiquement. Ainsi, asymptotiquement, AIC retiendra le modèle minimisant l'écart moyen entre la vraie distribution et la distribution avec paramètres estimés par maximum de vraisemblance tandis que BIC sélectionnera le modèle minimisant la divergence d'information avec la vraie loi. En pratique, les deux critères donnent cependant de bons résultats avec une préférence assez marqués pour BIC par les utilisateurs, AIC favorisant souvent des modèles trop complexes. Toutefois, il faut noter que les arguments de construction et de propriétés

de AIC et BIC sont affaiblis dans le cas du choix de K pour les mélanges. En effet, on peut s'attendre à ce que ces critères classiques sélectionnent un nombre très important de composantes pour le mélange afin de réaliser une bonne estimation de la loi inconnue de la population.

Raison pour laquelle, Biernacki et al. [7] proposent un nouveau critère, ICL (Integrated Classification Criterion) qui prend en compte l'objectif de regroupement des modèles de mélange. L'idée est de reporter l'objectif de classification de la méthode d'estimation vers la méthode de choix de modèle. En d'autres termes c'est à la méthode de choix de modèle de retenir un modèle produisant des classes bien séparées tout en respectant « au mieux » la distribution des données. Dans cette perspective, ICL pénalise la log-vraisemblance complétée calculée en $\widehat{\Theta}$ par le même terme de complexité que le critère BIC. Ce critère, à maximiser, s'écrit sous la forme suivante :

$$ICL = BIC - E(\widehat{t}, \widehat{z})$$

avec $E(\widehat{t}, \widehat{z})$ mesurant l'écart entre la partition z et les probabilités d'appartenance aux composantes du mélange. D'une part, on remarque donc que ICL est tout aussi simple à calculer que BIC. Et d'autre part, on déduit que ICL pénalisera les modèles produisant des classes trop imbriquées, ce que ne faisait pas BIC.

Le package "mclust" de R est l'un des plus populaires pour la modélisation des mélanges gaussiens. Depuis ses premiers développements ([5] ; [16] ; [17]), mclust a connu des mises à jour majeures au fil des ans, ce qui a élargi ses capacités et ses fonctionnalités, augmentant sa popularité et élargissant son domaine d'utilisation.

La fonction Mclust() donne les résultats de la classification, tels que le modèle gaussien retenu avec le nombre de classes, l'effectif de chaque classe, la valeur de la logvraisemblance et celle du critère de sélection du modèle souhaité (BIC par défaut).

Une combinaison de trois (3) lettres est définie pour le choix du modèle parmi les quatorze (14) existants (Voir Table 2.1).

On a : E (égalité), V (Variation) et I (distribution qui peut être diagonale, sphérique ou ellipsoïdale). La première position de la lettre correspond au volume, la deuxième à la forme et la troisième à l'orientation.

Par exemple, VEV correspond à un modèle avec des classes de même forme mais avec des volumes et des orientations différentes.

Modèle	**Distribution**	**Volume**	**Forme**	**Orientation**
EII	Sphérique	Égal	Égal	-
VII	Sphérique	Variable	Égal	-
EEI	Diagonale	Égal	Égal	Axes de coordonnées
VEI	Diagonale	Variable	Égal	Axes de coordonnées
EVI	Diagonale	Égal	Variable	Axes de coordonnées
VVI	Diagonale	Variable	Variable	Axes de coordonnées
EEE	Ellipsoïdale	Égal	Égal	Égal
EVE	Ellipsoïdale	Égal	Variable	Égal
VEE	Ellipsoïdale	Variable	Égal	Égal
VVE	Ellipsoïdale	Variable	Variable	Égal
EEV	Ellipsoïdale	Égal	Égal	Variable
VEV	Ellipsoïdale	Variable	Égal	Variable
EVV	Ellipsoïdale	Égal	Variable	Variable
VVV	Ellipsoïdale	Variable	Variable	Variable

TABLE 2.1 – Caractéristiques géométriques des quatorze (14) modèles de mélange gaussien obtenues à partir de la paramétrisation de la matrice de covariance

Chapitre 3

Méthode neuronale : Cartes auto-organisatrices (SOM)

Ce chapitre présente une nouvelle famille de classification automatique de données basée sur les réseaux de neurones artificiels. La méthode basée sur les cartes auto-organisatrices a été utilisée avec succès notamment dans la reconnaissance des formes et le traitement des images. Désignée en anglais par Self-Organizing Maps (SOM), cette méthode fût inventée par Kohonen (Kohonen, 1984). C'est un réseau de neurones artificiels soumis à un apprentissage non supervisé pour projeter des données de haute dimensionnalité sur un espace de faible dimension dans le but d'en faire des tâches de discrétisation, de quantification vectorielle ou de classification. Outre leur capacité de classification, elles permettent de renseigner sur la proximité des classes en préservant la topologie des données.

Dans ce qui suit, nous allons plus nous inspirer de la thèse de doctorat de Chantal Hajjar [19] pour expliquer le principe des cartes auto-organisatrices en détaillant ses algorithmes d'apprentissage. Nous allons également parler de l'évaluation de la performance de la carte, du choix de ses paramètres d'apprentissage, de ses critères de convergence et enfin de son rôle dans la classification de données.

Étant donné que, dans ce mémoire, notre étude porte sur l'apprentissage non supervisé, nous ne présentons pas les autres types d'apprentissage dont, notamment, l'apprentissage supervisé ou encore par correction des erreurs et l'apprentissage par renforcement (Haykin, 1994).

3.1 Principe

Une carte auto-organisatrice (SOM) est composée d'une grille des neurones (le plus souvent uni- ou bidimensionnelle). Si la grille est unidimensionnelle, chaque neurone possède deux voisins. Sinon la répartition des neurones se fait selon deux manières : soit d'une façon rectangulaire où chaque neurone possède 4 voisins (topologie rectangulaire) soit d'une façon hexagonale où chaque neurone possède 6 voisins (topologie hexagonale).

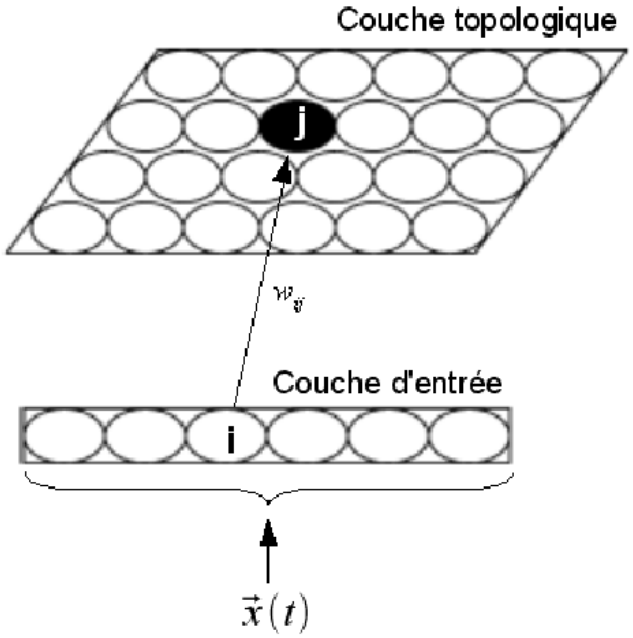


FIGURE 3.1 – *Schéma d'une carte auto-organisée de Kohonen (1984).*

À chaque neurone est associé un vecteur référent (appelé aussi vecteur prototype)

dans l'espace de données (ou espace d'entrée) et un emplacement sur la carte (ou espace de sortie) formé par le numéro de ligne et le numéro de colonne du neurone (voir Figure 3.1). Ces vecteurs référents sont de ce fait positionnés de telle façon qu'ils conservent la forme topologique de l'espace d'entrée. Chaque neurone de la grille marqué par une étiquette correspond à une classe. Ainsi, tous les éléments de l'espace d'entrée qui se projettent sur un même neurone appartiennent à la même classe. Une classe peut être associée à plusieurs neurones.

L'apprentissage du réseau se fait de manière non supervisée, en mode incrémental ou en mode différé (en mode batch), en vue de projeter les observations sur la carte. Dans la version incrémentale, chaque itération du processus d'apprentissage consiste à présenter aléatoirement une observation au réseau, et à mettre à jour les vecteurs prototypes des neurones en vue de les rapprocher de cette observation. Par contre, en mode différé, toutes les observations sont présentées au réseau à chaque itération, et les vecteurs prototypes des neurones sont mis à jour. En fin d'apprentissage, soit incrémental soit en mode différé, une observation sera affectée au neurone dont le vecteur prototype est le plus proche appelé le neurone vainqueur ou le Best Matching Unit (BMU). La classification des données peut se faire en supposant que les observations affectées au même neurone appartiennent à la même classe. Dans ce cas, le nombre de neurones doit correspondre au nombre de classes et deux neurones voisins sur la carte représenteront des classes voisines dans l'espace des observations.

3.2 Algorithmes d'apprentissage

L'algorithme de Kohonen nécessite de fixer au préalable un certain nombre de paramètres. Bien souvent, la connaissance du problème s'avère utile pour effectuer ces choix. Ainsi, il est nécessaire de fixer la topologie du réseau, le nombre de vecteurs

prototypes, la fonction de voisinage, le critère de convergence et le type d'apprentissage. Dans ce qui suit, nous mettrons en exergue les deux types d'apprentissage qui existent.

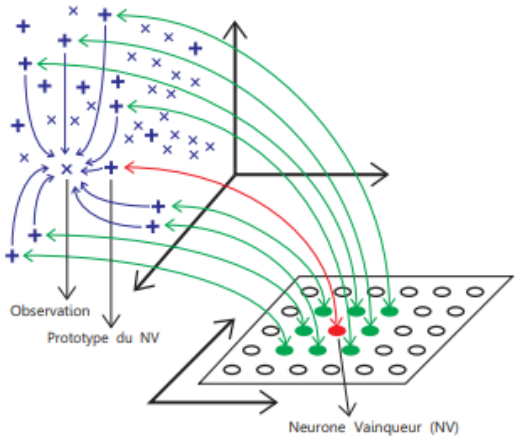


FIGURE 3.2 – *Mise à jour des vecteurs prototypes (tirée de Chantal Hajjar (avril 2015). Cartes auto-organisatrices pour la classification de données symboliques mixtes, de données de type intervalle et de données discrétisées, p.28).*

3.2.1 Apprentissage séquentiel

En désignant par K le nombre total des neurones de la carte, le vecteur référent du neurone k est donné par w_k avec $k \in 1, ..., K$.

Chaque itération **t** de l'apprentissage séquentiel comprend deux étapes. La première étape consiste à choisir au hasard une observation $x(t)$ de l'ensemble Ω, et à la présenter au réseau dans le but de déterminer son neurone vainqueur. Le neurone vainqueur d'une observation est le neurone dont le vecteur référent en est le plus proche au sens d'une distance donnée (ex : distance euclidienne). Si **c** est le neurone vainqueur du vecteur $x(t)$, **c** est déterminé comme suit :

$$d(w_c(t), x(t)) = \min_{k \in 1,...,K} d(w_k(t), x(t))$$

Dans la deuxième étape, le neurone vainqueur est activé. Son vecteur référent est mis à jour pour se rapprocher du vecteur d'entrée présenté au réseau. Cette mise à jour ne concerne pas seulement le neurone vainqueur, mais également les neurones qui lui sont voisins et qui voient alors leurs vecteurs référents s'ajuster vers ce vecteur d'entrée (voir figure 3.2). L'amplitude de cet ajustement est déterminée par la valeur d'un pas d'apprentissage $\alpha(t)$ et la valeur d'une fonction de voisinage $\mathbf{h(t)}$. Le paramètre $\alpha(t)$ règle la vitesse de l'apprentissage. Il est initialisé avec une grande valeur au début puis décroît avec les itérations en vue de ralentir au fur et à mesure le processus d'apprentissage. La fonction $\mathbf{h(t)}$ définit l'appartenance au voisinage. Elle dépend à la fois de l'emplacement des neurones sur la carte et d'un certain rayon de voisinage. Dans les premières itérations, le rayon de voisinage est assez large pour mettre à jour un grand nombre de neurones voisins du neurone vainqueur, mais ce rayon se rétrécit progressivement pour ne contenir que le neurone vainqueur avec ses voisins immédiats, ou bien même le neurone vainqueur seulement. La règle de mise à jour des vecteurs référents est la suivante :

$$w_k(t+1) = w_k(t) + \alpha(t) h_{kc}(t)[x(t) - w_k(t)]$$

$$k \in 1, ..., K$$

où **c** est le neurone vainqueur du vecteur d'entrée $x(t)$ présenté au réseau à l'itération **t** et $h_{kc}(t)$ est la fonction de voisinage qui définit la proximité entre les neurones **k** et **c**. Il décrit comment les neurones dans la proximité du vainqueur **c** sont entraînés dans le mouvement de correction. On utilise en général :

$$h_{kc}(t) = exp(-\frac{||\vec{c} - \vec{k}||^2}{2\sigma^2(t)})$$

où $\vec{c}$ et $\vec{k}$ sont respectivement l'emplacement du neurone c et du neurone k sur la carte. σ est le coefficient de voisinage à l'itération t du processus d'apprentissage. Son rôle est de déterminer un rayon de voisinage autour du neurone vainqueur.

3.2.2 Apprentissage en mode différé

En mode différé, à chaque itération t, toutes les observations sont présentées au réseau et la mise à jour des vecteurs prototypes se fait en prenant en compte toutes les observations de l'ensemble de données. Chaque vecteur prototype est une moyenne pondérée des vecteurs d'observations $x_i, i \in 1, ..., n$ quand le carré de la distance euclidienne est utilisée pour le calcul du neurone vainqueur. Les poids correspondants sont dans ce cas les valeurs de la fonction de voisinage $h(t)$. La règle de mise à jour des vecteurs prototypes est donnée par :

$$w_k(t+1) = \frac{\sum_{i=1}^{n} h_{kc_i}(t) x_i}{\sum_{i=1}^{n} h_{kc_i}(t)}$$

où h_{kc_i} est la valeur de la fonction de voisinage entre le neurone vainqueur c_i du vecteur x_i et le neurone k.

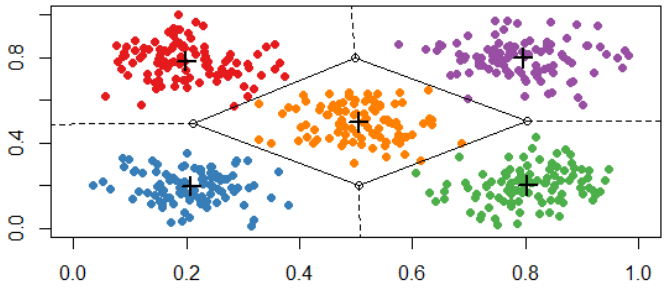


FIGURE 3.3 – *Régions de Voronoï avec une structure de voisinage*

La mise à jour des vecteurs prototypes peut être formulée autrement en utilisant le fait que les observations qui ont le même neurone vainqueur ont la même valeur pour la fonction de voisinage et appartiennent à la région dite de Voronoï dont le centre

est leur neurone vainqueur (figure 3.3) :

$$w_k(t+1) = \frac{\sum_{l=1}^{n} h_{kl}(t) n_l \overline{x}_l}{\sum_{l=1}^{n} h_{kl}(t) n_l}$$

où n_l est le nombre d'observations appartenant à la région de Voronoï représentée par le neurone l et $\overline{x}_l$ est la moyenne des observations de cette même région.

Vers la fin de l'apprentissage, quand le rayon de voisinage devient trop petit pour activer seulement le neurone vainqueur, chaque vecteur prototype constitue le centre de gravité des observations qu'il représente et on retombe alors sur l'algorithme des centres-mobiles, ce qui garantit une meilleure approximation de la fonction de densité des observations. Toutefois, le mode différé pourrait causer des torsions dans les cartes à grandes dimensions. Pour cette raison, on procède souvent à une analyse en composantes principales pour réduire les dimensions et initialiser les vecteurs prototypes.

3.3 Évaluation de performance

Une carte auto-organisatrice est généralement évaluée pour ses capacités de quantification et ses capacités de préservation de la topologie. Pour mesurer le degré de déploiement de la carte sur les données ou le degré de quantification, on calcule la moyenne des erreurs de quantification (Mean Quantization Error) qui est définie par :

$$mqe = \frac{\sum_{i=1}^{n} d_2^2(x_i, w_{c_i})}{n}$$

où d_2^2 est le carré de la distance euclidienne et c_i le neurone vainqueur de l'observation x_i. Plusieurs critères existent pour quantifier la préservation de la topologie. Dans le

cas où la carte est unidimensionnelle, on utilise souvent le critère suivant :

$$J = \sum_{k=1}^{K} d_2(w_k, w_k - 1) - d_2(w_K, w_1)$$

La topologie est parfaitement préservée quand ce critère vaut 0. Dans le cas général, un critère de préservation de la topologie sert à comparer les positions relatives des vecteurs référents par rapport aux positions relatives des neurones correspondants (Bauer et Pawelzik, 1992). Une autre approche consiste à mesurer le nombre de fois où la région de Voronoï d'un autre neurone de la carte s'introduit entre les vecteurs référents de deux unités voisines (Zrehen et Blayo, 1992).

3.4 Choix des paramètres du réseau

Le réglage des paramètres de la carte auto-organisatrice reste déterminant avant l'apprentissage, surtout qu'un mauvais choix de l'un de ces paramètres peut conduire à des résultats incohérents.

D'abord, les vecteurs prototypes initiaux ont une grande influence sur les résultats finaux au cas où ils sont initialisés aléatoirement. Une solution à ce problème consiste à exécuter plusieurs lancées de l'algorithme d'apprentissage, avec à chaque fois des vecteurs initiaux différents, et à adopter les résultats obtenus par la lancée qui permet de minimiser le plus l'erreur de quantification définie plus haut. Sinon, le choix des prototypes initiaux peut se faire de manière déterministe en réalisant une analyse en composantes principales (ACP) des données. Les vecteurs prototypes seront alors positionnés sur les deux axes formés par les deux premiers vecteurs propres de la matrice de covariance des observations. Dans ce cas, l'initialisation des prototypes est linéaire. Une telle initialisation est recommandée du fait qu'elle minimise le risque des torsions de la carte.

Il y a aussi l'influence du choix des valeurs initiales et finales du rayon de voisinage ainsi que du pas d'apprentissage sur les résultats obtenus. Pour les cartes à grandes dimensions, l'apprentissage se fait en deux phases. Dans la première phase, nommée phase d'organisation, le rayon de voisinage et le pas d'apprentissage sont initialisés avec de grandes valeurs et décroissent rapidement dans le but d'organiser les vecteurs prototypes. Le rayon de voisinage est souvent initialisé par une valeur entière supérieure à $\frac{D}{2}$ (où D est le diamètre de la grille de neurones, $D = \frac{L+K}{2}$ avec L nombre de lignes et K nombre de colonnes).

La deuxième phase nommée phase de convergence a pour but d'ajuster les vecteurs prototypes vers leurs positions finales sans changer l'ordre qu'ils ont appris dans la phase précédente. Pour cette raison, le rayon de voisinage et le pas d'apprentissage sont initialisés avec de petites valeurs. En général, le rayon de voisinage décroît pour atteindre la valeur 1, ce qui garantit une bonne préservation de la topologie. Cependant, ce rayon peut décroître au-dessous de 1 pour assurer un meilleur déploiement de la carte sur les données, surtout quand le but principal de l'utilisation de carte est la classification. Il est à noter que quand les prototypes sont initialisés moyennant une ACP, l'apprentissage se réduit à la phase de convergence seulement.

3.5 Critères de convergence

Il existe différents critères de convergence parmi lesquels :
- le nombre d'itérations : l'algorithme s'arrête dès que le nombre d'itérations fixé a priori est atteint ;
- le changement d'affectation des observations : l'algorithme s'arrête lorsque le nombre de changements d'affectation pendant la quantification vectorielle est inférieur à une certaine valeur prédéfinie,
- l'erreur de quantification : l'algorithme s'arrête lorsque l'erreur quadratique est in-

férieure à un certain seuil prédéfini.

L'algorithme d'apprentissage des cartes auto-organisatrices dans sa version séquentielle ou en mode différé optimise le critère suivant :

$$G_{SOM} = \frac{1}{2n} \sum_{i=1}^{n} \sum_{k=1}^{K} h_{kc_i} d_2^2(x_i, w_k)$$

Toutefois, ce critère de convergence est plus applicable pour les ensembles de données finis et discrets. En effet, il pose un problème pour les observations qui se trouvent sur le bord du pavage des classes voisines, ce qui entraîne des points de discontinuités pour la fonction de voisinage h_{kc_i}. De ce fait, pour une distribution discrète, la probabilité qu'une observation se trouve entre deux vecteurs prototypes sur le bord de la partition de Voronoï est nulle. Dans le cas où les données sont les réalisations d'une distribution continue, les cartes auto-organisatrices optimisent toujours le même critère, mais à condition d'adopter la règle suivante pour la recherche du neurone vainqueur :

$$c_i = \arg\min_{k \in 1,\ldots,K} \sum_{l=1}^{K} h_{kl} d_2^2(x_i, w_l)$$

L'apprentissage avec cette nouvelle règle pour la recherche du neurone vainqueur mène à des résultats très proches de ceux obtenus avec l'algorithme standard de Kohonen, mais nécessite des opérations de calcul plus complexes.

Chapitre 4

Application sur différents jeux de données

4.1 Étude statistique du jeu de données «Mnist»

La base de données Mnist (Modified National Institute of Standards and Technology Database) est une grande base de données de chiffres manuscrits qui est couramment utilisée pour la formation de divers systèmes de traitement d'images. Elle est constituée de 60000 images d'entrainement et 10000 images de test. L'ensemble d'images de la base de données Mnist est une combinaison de deux des bases de données du nist : Special Database 1 et Special Database 3. Ces dernières se composent de chiffres écrits respectivement par des lycéens et des employés du United States Census Bureau [31]. Le nombre de classes est connu d'emblée, soit 10. Il n'est donc pas nécessaire de faire appel au package Nbclust pour déterminer le nombre optimal de classes. De nombreuses méthodes d'analyse des données ont été appliquées avec ce jeu données. La classification hiérarchique n'étant pas adapté aux grands volumes de données, elle ne sera pas étudiée dans cette partie.

Dans cette étude, nous utiliserons seulement la base d'entrainement dans le cadre de la classification non supervisée avec une méthode classique (k-means), puis avec les mélanges de densité et enfin par une méthode neuronale (SOM). En d'autres termes, nous allons analyser un jeu de données constitué de 60000 observations et de 784 variables.

4.1.1 Application de la méthode k-means dans la base de données Mnist

Comme vu précédemment, l'idée de base de la classification par la méthode k-means consiste à construire des classes de manière à minimiser la variance totale intra-classe. L'objectif sera d'essayer d'identifier des groupes homogènes de chiffres sans utiliser la variable de réponse. Ici, nous fixons le nombre de classes $k = 10$ uniquement parce que nous savons déjà qu'il y a 10 chiffres distincts représentés dans les données. Nous allons donc initialiser notre algorithme avec 10 observations choisies aléatoirement (nstart = 10).

La sortie de notre modèle contient de nombreuses métriques dont nous avons déjà discuté, telles que la variation totale intra-classe (withinss), la somme totale des carrés intra-classe (tot.withinss), la taille de chaque classe (size) et le nombre d'itérations. Il donne également pour chaque observation, la classe à laquelle elle appartient. Les sorties résultat obtenues sur les centres de classes sont présentées sous forme d'une matrice 10×784. Cette matrice contient la valeur moyenne de chacune des 784 caractéristiques pour les 10 classes. Nous pouvons illustrer cela à travers la figure 4.1 qui nous montre quel est le chiffre typique dans chaque classe.

Nous pouvons comparer les chiffres de notre classification avec les étiquettes de chiffres réelles pour voir les performances de notre clustering. Pour ce faire, nous

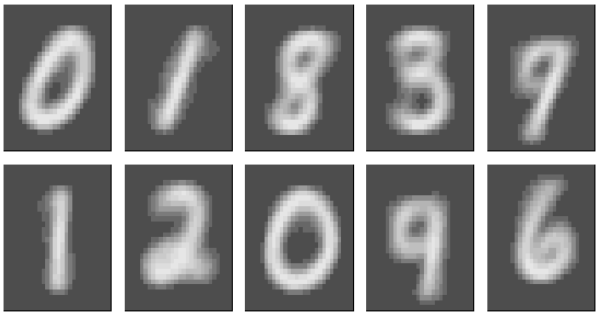


FIGURE 4.1 – *Images des centres de classe pour les 10 classes identifiées par k-means.*

comparons le chiffre le plus courant dans chaque classe (c'est-à-dire avec le mode) aux étiquettes d'apprentissage réelles. Ainsi, nous voyons que certains chiffres sont souvent regroupés avec des chiffres différents (par exemple, les « 4 » et les « 7 » sont souvent confondus avec des « 9 ». Les « 5 » et les « 6 » sont souvent confondus avec des « 0 »). Nous voyons également que les « 0 » et « 9 » ne sont jamais le chiffre dominant dans une classe. Par ailleurs, notre classification regroupe de nombreux chiffres qui ont une certaine ressemblance (« 4 » et « 7 » puis « 3 » et « 8 » sont souvent regroupés) et comme il s'agit d'une tâche non supervisée, il n'y a pas de mécanisme pour superviser l'algorithme autrement. Nous voyons clairement des chiffres reconnaissables même si k-means n'avait aucune idée de la variable de réponse.

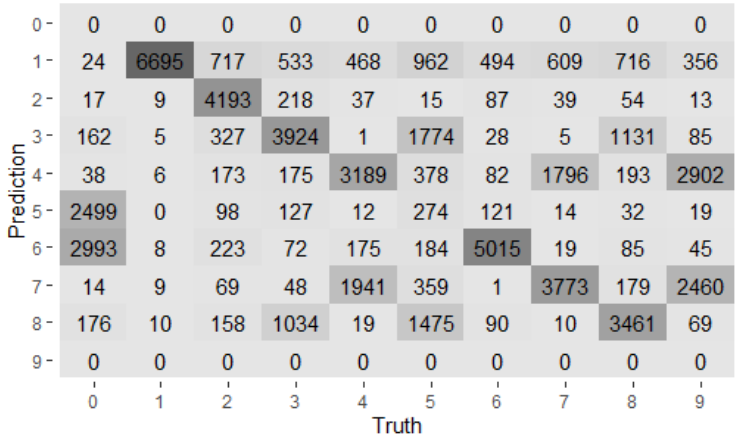


FIGURE 4.2 – *Matrice de confusion illustrant comment l'algorithme k-means a regroupé les chiffres prédits et les étiquettes réelles*

Le pourcentage de bonne classification calculé à partir de la matrice de confusion (Voir Figure 4.2) pour cette méthode est de 36%.

4.1.2 Application des modèles de mélange de densités dans la base de données Mnist

De nombreuses études ont montré l'intérêt de travailler avec un modèle de mélange pour pallier aux insuffisances des autres méthodes de classification comme le k-means. Cette approche de modélisation statistique fait appelle à la probabilité pour établir un classement automatique des individus en classes homogènes. En effet, elle présente deux avantages. D'une part elle permet d'avoir accès à des probabilités d'appartenance des individus aux différents groupes. D'autre part le cadre formel de cette approche permet de proposer des bases théoriques aux problèmes du choix du nombre de classes dont souffre la plupart des méthodes de classification.

Notons qu'en plus d'appliquer BIC pour identifier les paramètres de covariance optimaux, nous pouvons également l'utiliser pour identifier le nombre optimal de classes. En effet, plutôt que de spécifier le choix du nombre de classes dans Mclust(), on peut laisser l'argument G = NULL et permettre à la fonction d'évaluer entre 1 à 9 classes possibles. Ainsi Mclust() sélectionnera le nombre optimal de composants en fonction du BIC. Comme mentionné précédemment, ici nous fixons le nombre de classes $G = 10$ uniquement parce que nous savons déjà qu'il y a 10 chiffres distincts représentés dans nos données.

```
----------------------------------------------------
Gaussian finite mixture model fitted by EM algorithm
----------------------------------------------------

Mclust EII (spherical, equal volume) model with 10 components:

 log-likelihood     n   df        BIC        ICL
     -257198879 60000 7850 -514484125 -514484573

Clustering table:
    1     2     3     4     5     6     7     8     9    10
 7660 10182  4675  3042  9213  4484  8670  2883  3060  6131
```

FIGURE 4.3 – *Résumé des résultats de la classification par mélange de densité*

La fonction Mclust() en appliquant les 14 modèles du tableau 2.1, a identifié celui qui caractérise le mieux les données (ici EII pour distribution sphérique avec volumes égales tel que illustré par la figure 4.3).

Par ailleurs en exploitant les données de la classification par mélange de densité, on peut également trouver pour chaque observation, la classe à laquelle elle appartient. Les résultats obtenues sur les moyennes par classes sont présentés sous forme d'une matrice 784×10. Cette matrice contient la valeur moyenne de chacune des 784 caractéristiques pour les 10 classes proposées. Nous pouvons illustrer cela à travers la figure 4.4 qui nous montre quel est le chiffre typique dans chaque classe.

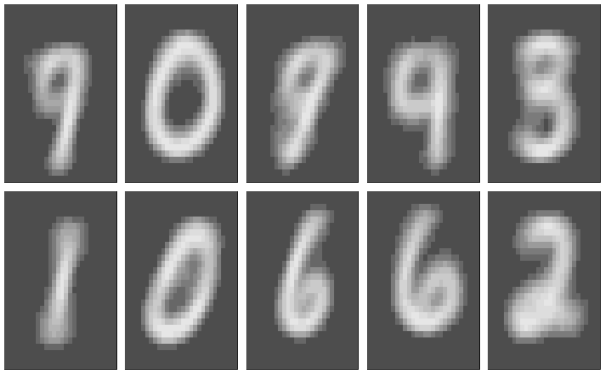


FIGURE 4.4 – *Images des centres de classe pour les 10 classes identifiées par mélange de densités*

De même nous voyons clairement des chiffres reconnaissables même si l'algorithme n'avait aucune idée de la variable de réponse.

En essayant de mesurer la qualité de notre partition via une matrice de confusion, nous voyons que l'algorithme EM fait un assez bon travail en regroupant certains des chiffres. En effet, la plupart des chiffres sont regroupés plus souvent avec des chiffres similaires qu'avec des chiffres différents. Cependant, nous voyons également que certains chiffres sont souvent regroupés avec des chiffres différents (par exemple, les 5 sont souvent regroupés avec des 0, les 7 sont souvent regroupés avec des 9 et les 3 souvent regroupés soit avec des 5 ou des 8). Nous voyons également que les 0 et 9 ne sont jamais le chiffre dominant dans une classe.

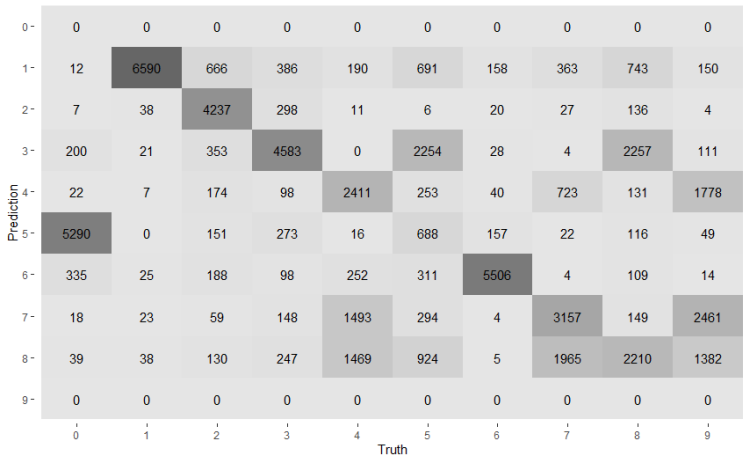


FIGURE 4.5 – *Matrice de confusion illustrant comment l'algorithme EM a regroupé les chiffres prédits et les étiquettes réelles*

Le pourcentage de bonne classification pour cette méthode est de 40% (Voir Figure 4.5).

4.1.3 Application des cartes auto-organisatrices (SOM) dans la base de données Mnist.

Comme énoncé dans le chapitre 3, les cartes auto-organisées représentent une alternative intéressante aux algorithmes d'apprentissage non supervisé et de réduction de dimensionnalité les plus couramment utilisés. La visualisation de la topologie du jeu de données peut être utile pour identifier les modèles cachés et les relations entre des classes données. De plus, l'algorithme peut être facilement étendu et utilisé en conjonction avec d'autres méthodes d'analyse de données (telles que l'ACP pour initialiser les poids ou k-means pour regrouper les données projetées au-dessus de la carte) pour résoudre des problèmes plus difficiles. Dans cette partie, nous allons traiter des cartes auto-organisatrices pour réaliser des tâches de classification automatique dans le jeu de données Mnist. Il existe plusieurs travaux d'implémentation de la carte SOM. Dans cette étude, nous utiliserons le package « kohonen » [30] du logiciel R en raison de sa flexibilité d'utilisation avec les grands jeux de données. La fonction som() de ce package permet de définir les paramètres d'apprentissage de l'algorithme. On a :

- Grid : qui définie la grille, sa taille, sa forme, le type de fonction de voisinage, etc.
- Rlen : qui donne le nombre de fois que l'ensemble des données sera présenté au réseau
- Alpha : qui fixe le pas d'apprentissage pour contrôler la vitesse d'apprentissage
- Radius : pour définir le rayon de voisinage

Nous avons défini une grille hexagonale de taille 10×10 (100 neurones) avec voisinage hexagonal. La librairie «kohonen» de R présente plusieurs types de visualisation permettant de mesurer de la pertinence de la carte obtenue. Parmi les plus importants on a :

- **Effectifs dans les nœuds**

  Ce type de graphique (Figure 4.6) renseigne sur le nombre d'individus capturés par un neurone. Les effectifs dans les nœuds permettent d'identifier les zones

à forte densité. En nous référant à la légende, on peut ainsi remarquer qu'il y a plus d'observations dans les neurones situés dans le côté supérieur droit.

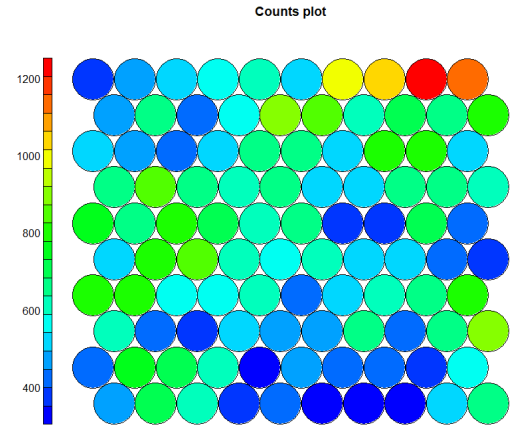


FIGURE 4.6 – *Carte coloriée en fonction des effectifs des neurones*

- **Distance au voisinage**

  Souvent appelée «matrice U», cette visualisation (Figure 4.7) indique la somme des distances aux voisins immédiats pour chaque nœud. De ce fait, les zones de faible distance voisine indiquent des groupes de nœuds similaires. Les zones avec de grandes distances indiquent que les nœuds sont beaucoup plus dissemblables. Cette représentation indique les limites naturelles entre les groupes de nœuds. La matrice U est ainsi utilisée pour identifier les classes dans la carte SOM.

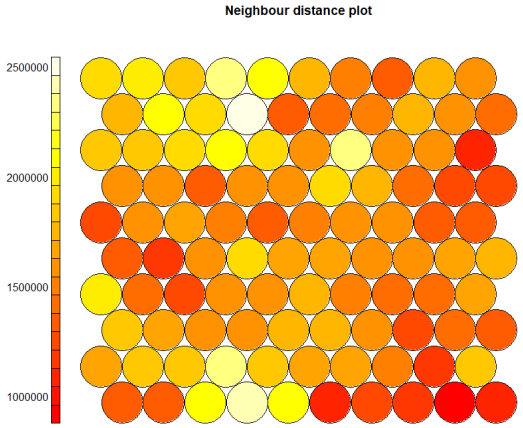


FIGURE 4.7 – *Distance au voisinage*

- **Regroupement des nœuds**

  Pouvoir enchaîner avec une classification automatique est un des intérêts des cartes topologiques de Kohonen. En effet, nous disposons d'une représentation 2D de nos observations avec des zones et des proximités que l'on sait caractériser avec les variables (Voir Figure 4.8) . Les nœuds constituent un excellent point de départ pour un regroupement itératif en classes. L'algorithme du k-means est ainsi mis à contribution dans ce contexte. L'idée étant de retrouver les dix classes qu'on connait de notre jeu de données «Mnist».

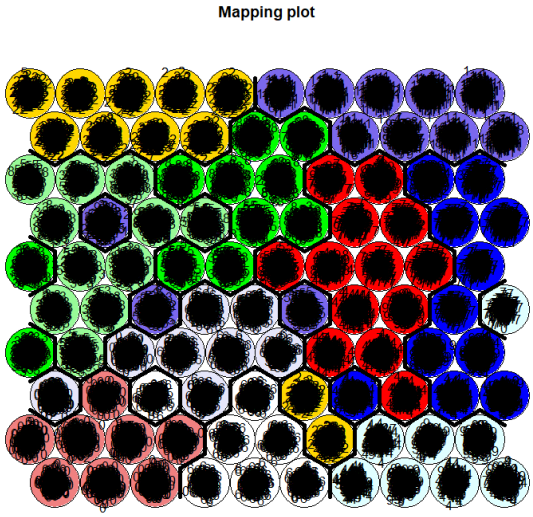


FIGURE 4.8 – *Représentation des classes dans la carte topologique*

De la même manière que les méthodes étudiées précédemment, on peut à partir des résultats obtenues sur le regroupement des nœuds, afficher les images moyennes par classes. Nous pouvons illustrer cela à travers la figure 4.9 qui nous donne une idée sur quel est le chiffre typique dans chaque classe.

De même, nous pouvons évaluer les performances de notre clustering en comparant les chiffres prédits avec les étiquettes de chiffres réelles. La figure 4.10 illustre les résultats. Nous voyons que notre algorithme SOM combiné avec k-means fait un meilleur résultat. En effet, la plupart des chiffres sont prédits dans les bonnes classes.

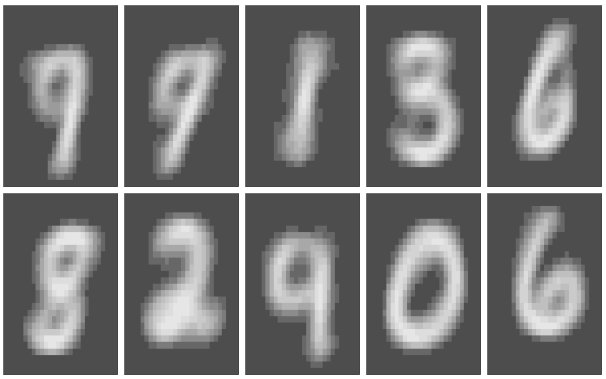


FIGURE 4.9 – *Images des centres de classe pour les 10 classes identifiées par la carte SOM*

Toutefois, nous voyons que les 0 ne sont toujours pas le chiffre dominant dans aucune classe. Cependant certains 0 sont prédits comme étant soit des 2 ou des 6. Par conséquent, on peut dire que notre clustering regroupe de nombreux chiffres qui ont une certaine ressemblance.

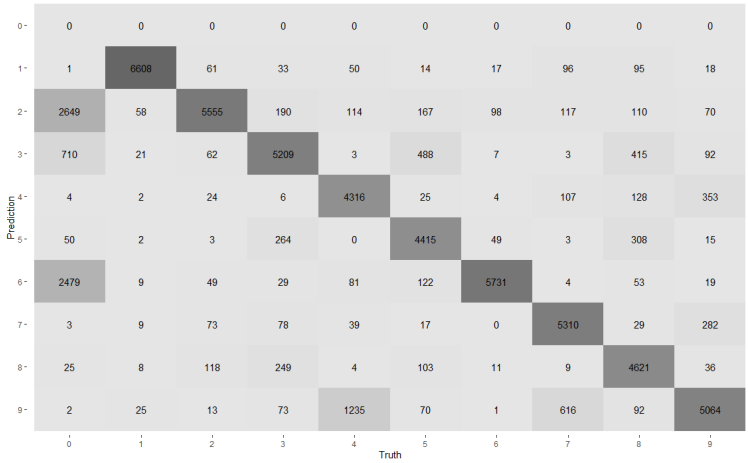


FIGURE 4.10 – *Matrice de confusion illustrant comment l'algorithme SOM a regroupé les chiffres prédits et les étiquettes réelles*

Le calcul du pourcentage de bonne classification extrait de la matrice de confusion pour cette méthode donne 64% (Voir Figure 4.10).

Il ressort ainsi de cette étude que l'algorithme SOM donne de meilleurs perfor-

	Taux de bonne classification
k-means	36%
Modèle de mélange	40%
SOM	64%

TABLE 4.1 – Tableau récapitulatif sur le taux de bonne classification

mances pour les données volumineuses (Voir Table 4.1). En effet, contrairement aux deux autres algorithmes, la plupart des chiffres sont bien prédits à défaut d'être regroupés plus souvent avec des chiffres similaires qu'avec des chiffres différents. Il n'y a que les 0 qui ne sont pas bien prédits dans leur classe réelle.

4.2 Étude statistique du jeu de données «Mybasket»

Tiré de la plateforme GitHub, ce jeu de données [34] fournit des informations sur certains articles d'épicerie et les quantités achetées pouvant ainsi permettre d'identifier les clients qui ont des préférences d'achat similaires. Chaque observation représente un seul panier de biens achetés ensemble par un client. La base de données est constituée de 2000 observations et de 42 variables.

Dans la même optique d'analyse, nous étudierons ce jeu de données dans le cadre de la classification non supervisée avec une méthode classique (k-médoïdes), puis avec les mélanges de densité et enfin avec les cartes auto-organisatrice (SOM). Cependant, contrairement à la classification des données faite sur la base de données « MNIST », où le nombre de classes que nous avons spécifié était basé sur la connaissance préalable des données. Pour ce jeu de données, nous ne disposons pas de ce type d'informations a priori, raison pour laquelle nous utilisons le package « NbClust » pour déterminer le choix optimal du nombre de classes. Et les résultats obtenus nous proposent sept (7) classes comme étant le nombre optimal pour la classification de nos 2000 observations

(voir annexe figure 4.13).

4.2.1 Application de la méthode k-médoïdes

L'algorithme des k-médoïdes, comme énoncé précédemment, est une alternative robuste aux k-means pour partitionner un ensemble de données en groupes d'observations. L'une des méthodes de classification des k-médoïdes les plus courantes est l' algorithme PAM (Partitioning Around Medoids,(Kaufman et Rousseeuw [21])).

La fonction pam() du package cluster de R peut être utilisée pour calculer cet algorithme. Le format simplifié est $pam(x, k)$, où x représente les données et k le nombre de classe à générer. Les observations les plus centraux peuvent être visualisées afin d'avoir une idée des habitudes alimentaires dans chaque groupe. Les médoïdes pour chaque classe de notre jeu de données sont présentés comme suit :

801 199 397 84 497 1781 251

En scrutant les achats pour ces différents paniers, il en ressort que la première classe est constituée en majorité de produits chocolatiers. La deuxième classe est composée de produits destinés au petit déjeuner tel que lait,thé et café. La troisième classe est en particulier constituée de boisson alcoolisées. Dans la quatrième classe on retrouve plus les boissons gazeuses. La cinquième classe est caractérisée par les légumes. La sixième par les produits de fast-food (pizza, lasagne, chicken tikka,...). Et pour la dernière classe on retrouve les aliments pour dessert tels que pain, mayonnaise, fromage...

4.2.2 Application des modèles de mélange de densités

Pour appliquer les modèles de mélange à nos données, nous permettrons à notre algorithme de rechercher le meilleur parmi les 14 modèles de mélange gaussien définis précédemment afin de trouver le choix optimal du nombre de classe. En effet le choix de ce dernier est basé sur la maximisation de la log-vraisemblance et la probabilité d'appartenance à une classe donnée. Un critère qui n'est pas pris en compte par le package NbClust. La figure 4.11 illustre les scores BIC et nous voyons que le modèle optimal est celui avec des classes de volumes et formes identiques, mais d'orientations différente (EEV) avec six classes.

```
----------------------------------------------------
Gaussian finite mixture model fitted by EM algorithm
----------------------------------------------------

Mclust EEV (ellipsoidal, equal volume and shape) model with 6 components:

 log-likelihood    n   df      BIC       ICL
       8308.915 2000 5465 -24921.1 -25038.38

Clustering table:
  1   2   3   4   5   6
391 403  75 315 365 451
```

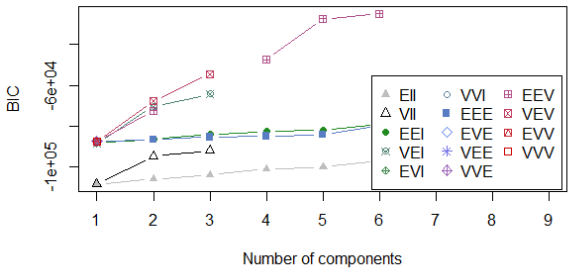


FIGURE 4.11 – *résumé des résultats de la classification par mélange de densité*

La comparaison avec les k-médoides permet de constater des observations qui se retrouvent dans les mêmes classes malgré la différence qui existe entre les deux algorithmes. Il en ressort 53% de formes fortes, tel qu'illustré dans le tableau 4.2. On peut ainsi remarquer que la classe 1 des k-médoides correspond plus à la classe 2 des modèles de mélange de densité et inversement. la classe 3 des k-médoides ressemble plus à la classe 4 des modèles de mélange. Les classes 4 et 5 des k-médoides correspondent à la classe 2 des modèles de mélange. Les classes 6 et 7 des k-médoides correspondent respectivement aux classes 5 et 4 des modèles de mélange de densité.

	Modèle de mélange					
k-Médoide	1	2	3	4	5	6
1	45	**176**	9	21	0	55
2	**188**	17	11	36	3	44
3	1	44	15	**89**	29	8
4	21	18	5	56	39	**182**
5	37	77	11	5	66	**140**
6	50	53	9	1	**180**	17
7	49	18	15	**107**	48	5

TABLE 4.2 – Formes fortes résultant de la méthode des k-médoïdes et des modèles de mélange

4.2.3 Application de la méthode SOM

L'application des cartes auto-organisatrices à notre jeu de données nous permet d'affecter les différentes observations aux nœuds puis à l'affectation de ces derniers aux classes. Les 7 classes identifiées dans les différents algorithmes précédents sont clairement mises en évidence (Voir figure 4.14 annexe). Et nous savons les interpréter directement maintenant.

La comparaison avec l'algorithme des k-médoides permet de constater des résultats assez proches dans la mesure où il en ressort 75% de formes fortes, tel qu'illustré dans le tableau 4.3.

On peut ainsi distinguer que les classes 1 et 5 des k-médoides correspondent à la classe 3 des cartes auto-organisatrices. Et les classes 2, 3, 4, 6 et 7 correspondent respectivement aux classes 1, 4, 5, 6 et 7 des cartes auto-organisatrices.

La comparaison entre les résultats tirés des cartes auto-organisatrices et ceux des modèles de mélange de densités donne un pourcentage de 60% de formes fortes (Table 4.4).

	SOM						
k-Médoide	1	2	3	4	5	6	7
1	11	3	**261**	4	8	11	8
2	**223**	2	40	6	6	12	10
3	3	3	19	**153**	3	3	2
4	6	0	30	22	**244**	7	12
5	21	67	**182**	8	17	40	1
6	5	3	6	7	5	**281**	3
7	8	3	43	7	9	12	**160**

TABLE 4.3 – Formes fortes résultant de la méthode des k-médoïdes et des cartes auto-organisatrices

	SOM						
Modèle de mélange	1	2	3	4	5	6	7
1	**180**	12	71	2	23	63	40
2	13	5	**259**	45	10	59	12
3	12	9	16	14	5	8	11
4	29	0	55	**99**	43	1	**88**
5	1	23	9	40	34	**218**	40
6	42	32	**171**	7	177	17	5

TABLE 4.4 – Formes fortes résultant des modèles de mélange et des cartes auto-organisatrices

Il en ressort ainsi que l'approche utilisée par l'algorithme des modèles de mélange donne de moins bons résultats concernant l'identification des formes fortes par rapport aux deux autres méthodes. En effet, ces résultats viennent conforter l'idée selon laquelle la méthode SOM est plus proche des méthodes statistiques à approche géométrique (k-means et k-médoïdes) qu'aux méthodes à approche probabiliste (voir Table 4.5).

Après une comparaison globale de ces différentes approches de clustering sur nos

	Modèle de mélange	k-médoïde	SOM
Modèle de mélange	100%		
k-médoïdes	53%	100%	
SOM	60%	75%	100%

TABLE 4.5 – Tableau récapitulatif des formes fortes

deux jeux de données, les résultats des expériences montrent que l'algorithme SOM donne plus de précision dans l'identification de la plupart des objets dans leurs classes appropriées que les autres. De plus, cet algorithme révèle des ressemblances par rapport aux méthodes classiques non hiérarchiques dans la détection de formes fortes. Ce qui pourrait s'expliquer par la proximité entre les approches de classification. La méthode SOM, tout comme le k-means et la méthode des k-médoïdes, utilise la distance comme mesure de ressemblance entre individus pour la classification. Alors que les modèles de mélange sont basés sur le fait que des individus appartenant à une même classe suivent la même distribution de probabilité.

Conclusion et perspectives

La classification automatique des données est une tâche complexe impliquant le choix entre de nombreuses méthodes, paramètres et mesures de performance différents, avec des implications dans de nombreux problèmes du monde réel. Par conséquent, l'analyse des avantages et inconvénients des algorithmes de classification non supervisée (clustering) est également une tâche difficile qui a reçu beaucoup d'attention.

Dans ce mémoire, nous avons exploré différentes méthodes de classification non supervisée dans une approche comparative pour évaluer leurs performances sur les mêmes jeux de données. Pour ce faire, nous avons considéré trois principales méthodes de classification à savoir, l'algorithme des k-means et celui des k-médoides d'une part, les modèles de mélange de densités d'autre part qui donne une approche probabiliste de la classification et enfin les cartes auto-organisatrices (SOM).

Ici, nous avons abordé cette tâche à l'aide de packages et fonctions du logiciel statistique R. En effet, nous avons réaliser une étude comparative des algorithmes permettant de comparer les performances de trois méthodes de clustering populaires appliquées à deux grands jeux de données à savoir Mnist et Mybasket. Les résultats obtenus pour l'ensemble des méthodes citées sont assez bons dans l'ensemble. Mais les méthodes neuronales donnent de meilleures performances pour les données volumineuses et présentent beaucoup plus de similarité avec les méthodes classiques non hiérarchiques, telles que les k-means et les k-médoides, qu'avec les modèles de mélange

de densités. Toutefois il faut noter que ces derniers bien qu'ils soient performants, sont appliqués sur des distributions supposées gaussiennes. Ce qui n'est pas le cas pour un bon nombre d'ensemble de données réelles.

En guise de perspective, il serait intéressant d'élargir ces modèles de mélange de densités à d'autres types de distributions telles que les distributions de poisson, exponentielle, etc. Mais également une autre perspective de ce travail serait de comparer les méthodes étudiées avec d'autres méthodes neuronales non supervisées telle que le réseau «Growing Neural Gas» (GNG) [18].

Bibliographie

[1] Abbas, O. A. Comparisons between data clustering algorithms. International Arab Journal of Information Technology (IAJIT), 2008.

[2] Ahmad, P. H., et Dang, S. , Performance evaluation of clustering algorithm using different datasets. International Journal of Advance Research in Computer Science and Management Studies, 2015.

[3] Akaike H. A new look at statistical model identification. IEEE Transactions on Automatic Control, AC-19 :716-723, 1974.

[4] Baillargeon, Notes de cours STT-4230 / STT-6230 : R pour scientifique, 2020.

[5] Banfield, J and Raftery A. E., Model-based Gaussian and non-Gaussian clustering. Biometrics, 49 : 803-821, 1993.

[6] Bauer H-U. et Pawelzik K.R. : Quantifying the neighborhood preservation of self-organizing feature maps. IEEE Transactions on Neural Networks, 3(4) :570-579, 1992.

[7] Biernacki C., Celeux G. and Govaert.G. *Assessing a mixture model for clustering with the integrated completed likelihood. IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no 7, 719-725, 2000.

[8] Boehmke, B. et Greenwell, B. M., Hands-on machine learning with R. CRC Press, 2019.

[9] Chair S. *Détermination du nombre de classes dans les méthodes de classification non supervisée, en particulier la méthode Multi-SOM.* Mémoire de Maîtrise. 2017

[10] Charrad, M., Ghazzali, N., Boiteau, V., and Niknafs, A. *Nbclust : An R package for determining the relevant number of clusters in a data set.* Journal of Statistical Software, vol. 61, no 6, 1-36, 2014.

[11] Chen, G., Jaradat, S. A., Banerjee, N., Tanaka, T. S., Ko, M. S., and Zhang, M. Q. . Evaluation and comparison of clustering algorithms in analyzing ES cell gene expression data. Statistica Sinica, 241-262, 2002.

[12] Dempster A. P., Laird N. M., and Rubin D. B. Maximum likelihood from incomplete data via the em algorithm. Journal of the royal statistical society, series b, 39(1) :1-38, 1977.

[13] Diday, E. Une nouvelle méthode en classification automatique et reconnaissance des formes la méthode des nuées dynamiques. Revue de statistique appliquée, 19(2), 19-33, 1971.

[14] Dutt, A., Aghabozrgi, S., Ismail, M. A. B., and Mahroeian, H., Clustering algorithms applied in educational data mining. International Journal of Information and Electronics Engineering, 2015.

[15] El Attar A. *Estimation robuste des modèles de mélange sur des données distribuées. Apprentissage [cs.LG]. Université de Nantes, 2012.*

[16] Fraley, C. and Raftery, A. E. How many clusters ? Which clustering method ? Answers via model-based cluster analysis. The Computer Journal, 41 :578-588, 1998.

[17] Fraley, C. and Raftery, A. E. MCLUST : Software for model-based cluster analysis. Journal of Classification, 16(2) :297-306, 1999.

[18] Fritzke B., «A Growing Neural Gas Network Learns Topologies», publié dans Advances in Neural Information Processing Systems 7, G. Tesauro, D.S. Touretzky et T.K. Leen (editeurs), MIT Press, Cambridge MA, 1995.

[19] Hajjar C. *Thèse de doctorat : Cartes auto-organisatrices pour la classification de données symboliques mixtes, de données de type intervalle et de données discrétisées*, 2015.

[20] Haykins S., Neural Networks : A comprehensive Foundation, IEEE Press, 1994.

[21] Kaufman L. and Rousseeuw P. J, Partitioning around medoids (program pam), Finding groups in data : an introduction to cluster analysis, volume 344, pages 68-125, Wiley New York, 1990.

[22] Kohonen T. : Self organization and associative memory, Second Edition. Springer-Verlag, 1984.

[23] MacQueen, J. Some methods for classification and analysis of multivariate observations. In Proceedings of the fifth Berkeley symposium on mathematical statistics and probability 1(14), 281-297, 1967.

[24] Maulik, U., and Bandyopadhyay, S., Performance evaluation of some clustering algorithms and validity indices. IEEE Transactions on pattern analysis and machine intelligence, 2002.

[25] Milligan, G. W., and Cooper, M. C. An examination of procedures for determining the number of clusters in a data set. Psychometrika, 50(2), 159-179, 1985.

[26] Parizeau, M., Réseaux de neurones. Université Laval, 2009.

[27] Pearson, K. Contributions to the mathematical theory of evolution. Philosophical Transactions of the Royal Society of London. A, 185, 71-110, 1894.

[28] Schwarz G. Estimating the number of components in a finite mixture model. Annals of Statistics, 6 :461-464, 1978.

[29] Volle M. *L'analyse des données. In : Economie et statistique, N° 96, 3-23, 1978.*

[30] Wehrens R, Kruisselbrink J . kohonen : Supervised and Unsupervised Self-Organising Maps. R package version 3.0.7, URL https ://CRAN.R-project.org/package=kohonen, 2018.

[31] Zrehen S. et Blayo F. : A geometric organization measure for Kohonen maps. In Proceedings of NeuroNîmes, 611-621, 1992.

[32] `http://factominer.free.fr/livre/temperat.csv`

[33] `http://yann.lecun.com/exdb/mnist/`

[34] `https://github.com/koalaverse/homlr/tree/master/data`

[35] `https://bradleyboehmke.github.io/HOML/index.html`

Annexe A : Résultats connexes

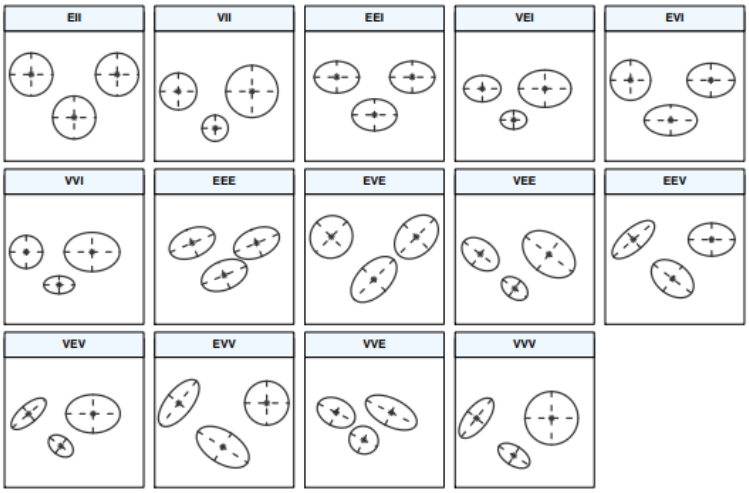


FIGURE 4.12 – *Interprétation géométrique de chacun des quatorze modèles gaussiens avec trois classes en deux dimensions*

```
******************************************************************
* Among all indices:
* 3 proposed 2 as the best number of clusters
* 2 proposed 3 as the best number of clusters
* 3 proposed 4 as the best number of clusters
* 3 proposed 5 as the best number of clusters
* 2 proposed 6 as the best number of clusters
* 5 proposed 7 as the best number of clusters
* 1 proposed 8 as the best number of clusters
* 2 proposed 10 as the best number of clusters

                 ***** Conclusion *****

* According to the majority rule, the best number of clusters is  7


******************************************************************
```

FIGURE 4.13 – *Nombre de classes optimal du jeu de données "my basket"*

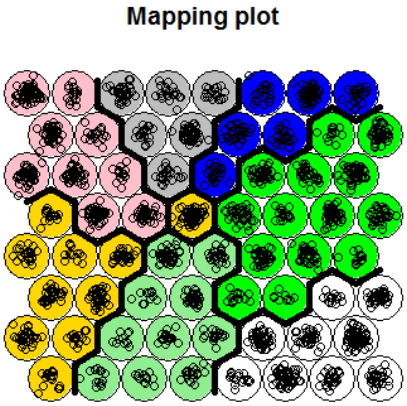


FIGURE 4.14 – - *Représentation des classes dans la carte topologique*

Annexe B : Scripts R

```
#Simulation de 5 classes pour illustrer les régions de Voronoï
#avec structure de voisinage

set.seed(1)
A <- c(rep(.2,100),rep(.2,100),rep(.5,100),rep(.8,100),rep(.8,100))
B <- c(rep(.2,100),rep(.8,100),rep(.5,100),rep(.2,100),rep(.8,100))
pts <- cbind(X=rnorm(500,A,.075),Y=rnorm(500,B,.075))

km <- kmeans(pts, centers=5, nstart = 1, algorithm = "Lloyd")
library(tripack)
library(RColorBrewer)
CL5 <- brewer.pal(5, "Set1")
V <- voronoi.mosaic(km$centers[,1],km$centers[,2])
P <- voronoi.polygons(V)
plot(pts,pch=19,xlim=0:1,ylim=0:1,xlab="",ylab="",col=CL5[km$cluster])
points(km$centers[,1],km$centers[,2],pch=3,cex=1.5,lwd=2)
plot(V,add=TRUE)


set.seed(301)

x<-matrix(rnorm(300*2),300,2)
x[1:300,]
```

```
plot(x,pch=19)
select<-sample(1:3,300,replace=TRUE)
plot(x,col=select,pch=19)
xmoy<-matrix(rnorm(3*2,sd=4),3,2)
xcluster<-x+xmoy[select,]
plot(xcluster,col=select,pch=19)

library(NbClust)
NbClust(xcluster, distance = "euclidean", min.nc = 2, max.nc = 8,
        method = "ward.D2",index = "all")


#Sorties résultats Mnist obtenus par K-means\\

str(Kmeans_clustering)\\
%List of 9
  cluster     : int [1:60000] 5 1 8 3 9 4 2 5 2 9 ...\\
   centers     : num [1:10, 1:784] 0 0 0 0 0 0 0 0 0 0 ...\\
  ..- attr(*, "dimnames")=List of 2\\
  .. ..  : chr [1:10] "1" "2" "3" "4" ...\\
  .. ..  : chr [1:784] "X1x1" "X1x2" "X1x3" "X1x4" ...\\
   totss       : num 2.06e+11\\
   withinss    : num [1:10] 8.84e+09 9.08e+09 1.03e+10\\
    1.51e+10 1.81e+10 ...\\
   tot.withinss: num 1.53e+11\\
   betweenss   : num 5.27e+10\\
   size        : int [1:10] 3157 5967 5588 4686 6569 3088
   \\ 7451 8929 8845 5720\\
   iter        : int 46\\
  ifault      : NULL\\
```

```
 - attr(*, "class")= chr "kmeans"

library(dplyr)
library(ggplot2)
library(stringr)
library(cluster)
library(factoextra)

#Importation du jeu de données mnist_train

library(readr)
mnist_train <- read.csv("F:/UQTR/mnist_train.csv")
Mnist_train <-mnist_train[,-1]


# Apprentissage par kmeans

Kmeans_clustering <- kmeans(Mnist_train, algorithm="Lloyd",
\\ centers = 10, nstart = 10, iter.max = 50)
str(Kmeans_clustering)
Centres <- Kmeans_clustering$centers


# Images des Centres de classe pour les 10 classes identifiées
# par k-means

par(mfrow = c(2, 5), mar=c(0.5, 0.5, 0.5, 0.5))
layout(matrix(seq_len(nrow(Centres)), 2, 5, byrow = FALSE))
for(i in seq_len(nrow(Centres))) {
  image(matrix(Centres[i, ], 28, 28)[, 28:1],
```

```
        col = gray.colors(12), xaxt="n", yaxt="n")
}

# Création de la fonction mode
mode1 <- function(x){
  which.max(tabulate(x))
}

mnist_compar <- data.frame(
  cluster = Kmeans_clustering$cluster,
  actual = mnist_train$label
) %>%
  group_by(cluster) %>%
  mutate(mode = mode1(actual)) %>%
  ungroup() %>%
  mutate_all(factor, levels = 0:9)


# Matrice de confusion
# par k-means
yardstick::
conf_mat(data = mnist_compar,
  truth = actual,
  estimate = mode
) %>%
  autoplot(type = 'heatmap')

accuracy(mnist_compar, truth= actual, estimate= mode,
\\ na_rm = TRUE, ...)%>%
  autoplot(type = 'heatmap')
```

```
# Apprentissage par mélange de densités

library(mclust)

mclust_clustering <-Mclust(Mnist_train, G=10)

save(mclust_clustering, file = "mclust_clustering")

summary(mclust_clustering)
str(mclust_clustering)

mnist_centers <- mclust_clustering$parameters$mean
tmnist_centers <- t(mnist_centers)


# Images des centres de classe pour les 10 classes identifiées
# par mélange de densités

par(mfrow = c(2, 5), mar=c(0.5, 0.5, 0.5, 0.5))
layout(matrix(seq_len(nrow(tmnist_centers)), 2, 5, byrow = FALSE))
for(i in seq_len(nrow(tmnist_centers))) {
  image(matrix(tmnist_centers[i, ], 28, 28)[, 28:1],
        col = gray.colors(12), xaxt="n", yaxt="n")
}

# Création de la fonction mode

mode_fun <- function(x){
  which.max(tabulate(x))}
```

```
mnist_compar1 <- data.frame(
  classification = mclust_clustering$classification,
  actual = mnist_train$label
) %>%
  group_by(classification) %>%
  mutate(mode = mode_fun(actual)) %>%
  ungroup() %>%
  mutate_all(factor, levels = 0:9)

# Matrice de confusion
# par mélange de densités

yardstick::conf_mat(
  mnist_compar1,
  truth = actual,
  estimate = mode
) %>%
  autoplot(type = 'heatmap')

adjustedRandIndex(mnist_compar1$actual, mnist_compar1$mode)


# Apprentissage par SOM
library(MASS)
library(ggplot2)
library(dplyr)
library(kohonen)
```

```
# Palette de couleur pour l'affichage des cartes
coolBlueHotRed <- function(n, alpha = 1) {
  rainbow(n, end=4/6, alpha=alpha)[n:1]}

matMnist<- as.matrix(Mnist_train)


# Choix du type de carte et de sa taille
som.grid <- somgrid(10, 10, topo="hexagonal",
neighbourhood.fct="bubble")
# Apprentissage
Mnist_som <- som(matMnist, grid =som.grid, rlen=100,
alpha=c(0.05,0.01), keep.data = TRUE)

# Affichage des cartes
plot(Mnist_som, type="count", palette.name=coolBlueHotRed)

#Carte coloriée en fonction de la cardinalité des neurones

plot(Mnist_som,type="dist.neighbours",
 palette.name=coolBlueHotRed)

#########Rôle des variables#########

plot(Mnist_som,type="codes",codeRendering = "segments")
######

coolBlueHotRed <- function(n, alpha = 1)
{rainbow(n, end=4/6, alpha=alpha)[n:1]
}
```

```
#graphique pour chaque variable
par(mfrow=c(6,5))
for (j in 1:ncol(bcovid)){
  plot(Mnist_som,type="property",
  property=somcovid$codes[[1]][,j],
 palette.name=coolBlueHotRed,
 main=colnames(bcovid)[j],cex=0.5)}
par(mfrow=c(1,1))
```

```
# On fixe le nombre de classes à 10
Mnist_somkmeans <- kmeans(data.frame(Mnist_som$codes),
centers = 10, nstart=10)
plot(Mnist_som, type="mapping",
bgcol = c("blue","gold","gray", "pink", "lightgreen",
"red", "purple", "yellow", "white",
 "green")[cov.somkmeans$cluster])
add.cluster.boundaries(Mnist_som,
clustering = Mnist_somkmeans$cluster)
```