# Contribution de la classification automatique à la fouille de données :

François-Xavier Jollois

**École doctorale IAEM lorraine** — **Université de Metz**
**Département de Formation Doctorale en Informatique** — **UFR MIM**

# Contribution de la classification automatique à la Fouille de Données



## THÈSE

présentée et soutenue publiquement le 12 décembre 2003

pour l'obtention du

**Doctorat de l'Université de Metz**

**spécialité Informatique**

par

François-Xavier Jollois

**Composition du jury**

| | | |
|---|---|---|
| *Directeur* | Maurice Margenstern | Professeur, directeur du LITA, Université de Metz |
| *Responsable* | Mohamed Nadif | Maître de Conférence, Université de Metz |
| *Rapporteurs* | Gildas Brossier | Professeur, Université de Rennes |
| | Yves Lechevallier | Directeur de Recherche, INRIA |
| *Examinateurs* | Edwin Diday | Professeur, Université de Paris 9 Dauphine |
| | Gérard Govaert | Professeur, Université Technologique de Compiègne |

**Laboratoires d'Informatique Théorique et Appliquée**

**École doctorale IAEM lorraine** — **Université de Metz**
**Département de Formation Doctorale en Informatique** — **UFR MIM**

# Contribution de la classification automatique à la Fouille de Données

# THÈSE

présentée et soutenue publiquement le 12 décembre 2003

pour l'obtention du

**Doctorat de l'Université de Metz**

**spécialité Informatique**

par

François-Xavier Jollois

**Composition du jury**

| | | |
|---|---|---|
| *Directeur* | Maurice Margenstern | Professeur, directeur du LITA, Université de Metz |
| *Responsable* | Mohamed Nadif | Maître de Conférence, Université de Metz |
| *Rapporteurs* | Gildas Brossier | Professeur, Université de Rennes |
| | Yves Lechevallier | Directeur de Recherche, INRIA |
| *Examinateurs* | Edwin Diday | Professeur, Université de Paris 9 Dauphine |
| | Gérard Govaert | Professeur, Université Technologique de Compiègne |

**Laboratoires d'Informatique Théorique et Appliquée**

# Remerciements

Je tiens à remercier tout d'abord Mohamed Nadif, qui m'a énormément aidé lors de ces trois années de thèse, ainsi que lors de mes précédentes études à l'IUT et à l'Université. Il était là pour me remotiver lorsque le cœur n'y était plus. Il m'a beaucoup conseillé, et les discussions que l'on a pu avoir se sont toujours révélées très intéressantes et instructives.

Je remercie également Maurice Margenstern pour avoir accepter de diriger cette thèse, ainsi que tout le Laboratoire d'Informatique Théorique et Appliquée de Metz pour m'avoir soutenu, encouragé et conseillé tout au long de ce travail.

Je remercie aussi tous les enseignants du département STID de l'IUT de Metz, ainsi que le personnel, pour tous les services qu'ils ont pu me rendre et leur sympathie.

Je souhaite également adresser mes plus sincères remerciement à Gildas Brossier et Yves Lechevallier pour avoir accepter d'être les rapporteurs de ma thèse. Je leur suis reconnaissant de leur intérêt pour mon travail.

Je remercie aussi Gérard Govaert et Edwin Diday de me faire l'honneur d'être membre du jury de ma thèse, d'autant plus que mes travaux se basent sur une partie des leurs.

Je voudrais aussi montrer toute ma reconnaissance et ma gratitude à tous les membres de ma famille qui ont toujours été là, même quand ça n'allait pas. Merci à vous : Mom, Val, Christine et Gilles, Rachel et Arnaud, et toute la famille Nigon, ainsi qu'aux Bardet, qui font presque partie de la famille maintenant.

Bien sûr, je tiens à montrer tout ma gratitude envers toutes les personnes qui ont pu m'aider, m'encourager, me soutenir, me remotiver pendant ces années de travail : Fanny, les amis de toujours de STID et plus (Olivier, Renaud, Yann, Damien, Sebastien) ainsi que leur épouse ou concubine, mes acolytes musicaux avec tous les bons et mauvais moments passés ensemble (Fred, Gilles, et tous les autres que ce soit en groupe ou avec l'association). J'en oublie certainement mais je leur dit encore merci pour tout.

*à la mémoire de mon père...*

# Table des matières

# Introduction

Depuis quelques années, une masse grandissante de données sont générées de toute part par les entreprises, que ce soit des données bancaires, telles que les opérations de carte de crédit, ou bien des données industrielles, telles que des mesures de capteurs sur une chaîne de production, ou toutes autres sortes de données possibles et imaginables. Ce flot continu et croissant d'informations peut de plus maintenant être stocké et préparé à l'étude, grâce aux nouvelles techniques d'**Entrepôt de Données** (ou *Data Wharehouse*).

Les techniques usuelles d'analyse de données, développées pour des tableaux de taille raisonnable ont largement été mises à mal lors de l'étude de tant de données. En effet, alors que le principal objectif de la statistique est de prouver une hypothèse avancée par un expert du domaine, et donc de confirmer une connaissance déjà connue ou bien présumée, le but de la **Fouille de Données** (ou *Data Mining*) est maintenant de découvrir, au sens propre du terme, des nouvelles connaissances. Et ceci sans faire appel à des hypothèses préconçues. Ce nouveau concept de Fouille de Données, bien qu'il paraît révolutionnaire pour certains, est en fait une autre vision et une autre utilisation de méthodes existantes, et combinées.

Ainsi, au vue de l'émergence de ces deux champs d'application (Fouille et Entrepôt de Données), une idée nouvelle s'est faite. Pourquoi ne pas associer toutes ces techniques afin de créer des méthodes puissantes de recherche de connaissances, intégrant toutes les étapes, du recueil des données à l'évaluation de la connaissance acquise. C'est ainsi qu'est né le terme d'**Extraction des Connaissances à partir des Données** (ECD), ou en anglais *Knowledge Discovery in Database* (KDD). Ce processus, décrit dans la figure 1, englobe donc le stockage et la préparation des données, l'analyse de celles-ci par différentes techniques, et enfin, l'interprétation et l'évaluation des connaissances acquises. Il est ici important de différencier les trois termes suivants :

- Donnée : valeur d'une variable pour un objet (comme le montant d'un retrait d'argent par exemple),
- Information : résultat d'analyse sur les données (comme la répartition géographique de tous les retraits d'argent par exemple),
- Connaissance : information utile pour l'entreprise (comme la découverte du mauvais emplacement de certains distributeurs).

Ainsi, le principe est, idéalement, à partir de données dont on ne sait rien et sur lesquelles on ne fait aucune hypothèse, d'obtenir des informations pertinentes, et à partir de celle-ci de découvrir de la connaissance, qui permettra à l'entreprise de

mieux gérer, comprendre, analyser, prévoir, économiser. Bref, en un mot, d'améliorer le travail de l'entreprise, dans tous les domaines possibles.

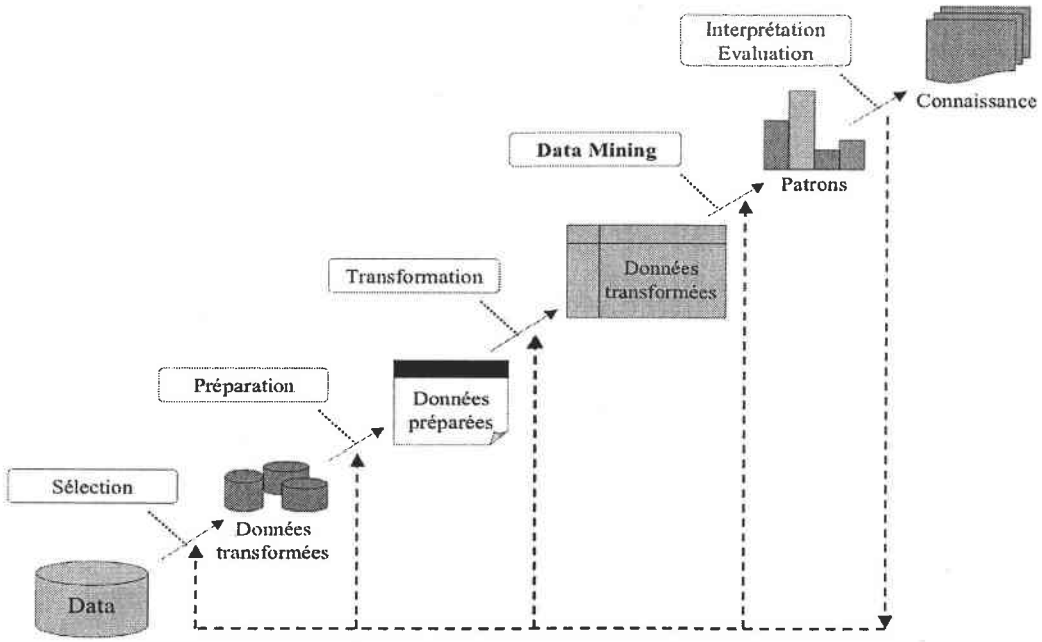



FIG. 1 – *Un aperçu des étapes de l'ECD*

La Fouille de Données est donc une des composantes de l'ECD, mais sûrement la plus importante. C'est un domaine multidisciplinaire, mélangeant la technologie des bases de données, l'intelligence artificielle, l'apprentissage automatique, les réseaux de neurones, les statistiques, la reconnaissance de formes, les systèmes à bases de connaissances, l'acquisition de connaissance, les systèmes de recherches d'informations, l'informatique haute-performance, et la visualisation de données. Elle permet d'obtenir des informations à partir de données réarrangées et préparées. Par contre, il est nécessaire de passer par une étape d'évaluation avec l'aide d'un expert du domaine afin de relever la pertinence de ces informations et de leur éventuelle apport pour la connaissance de l'entreprise.

Alors qu'habituellement, les statisticiens travaillent sur des bases de données de taille raisonnable, en échantillonnant parfois la population, les utilisateurs de la Fouille de Données désirent quant à eux garder un maximum d'information et donc travailler sur l'ensemble des données disponibles. Cette volonté engendre un certain nombre de complications dans l'analyse de tels tableaux. En effet, bien que ce ne soit qu'un aspect algorithmique des méthodes, il est absolument nécessaire qu'une méthode destinée à être utilisée dans le domaine de la Fouille de Données soit adaptée pour des grands volumes de données, voire pour travailler en temps réel pour certaines applications.

La classification automatique est une des méthodes statistiques largement utilisées dans la Fouille de Données. Elle est dans un cadre d'apprentissage non supervisé, qui a pour but de d'obtenir des informations sans aucune connaissance préalable, au contraire de l'apprentissage supervisé. Elle a plusieurs possibilités de combinaison avec d'autres méthodes, en pre- ou en post-processing. En effet, elle peut résumer l'information afin de la transmettre à une autre méthode et ainsi mieux analyser les données. Elle peut aussi, suite à un pré-traitement des données, être utilisée pour mieux comprendre la quintessence de l'information contenue dans les fichiers.

Grouper des objets ensemble selon certaines caractéristiques, afin de dissocier différents ensembles est un processus de classification. En effet, lorsque naturellement, nous regroupons ensemble d'un côté les êtres humains, de l'autre les animaux, eux-mêmes pouvant être séparés en différents groupes distincts, nous effectuons une classification. Ce processus naturel à l'homme permet, à partir d'un certain nombre de données et de règles, de diviser un ensemble d'objets en différentes classes, distinctes et homogènes.

Le principe de la classification automatique est d'imiter ce mécanisme par une machine. Pour ceci, il faut développer des méthodes qui s'appuient soit sur les données à proprement parlé, sans aucune connaissance autre, soit sur les données et sur un savoir acquis préalablement (automatiquement ou grâce à un expert du domaine).

La classification automatique a de multiples applications, et dans des domaines très diverses. En économie, la classification peut aider les analystes à découvrir des groupes distincts dans leur base clientèle, et à caractériser ces groupes de clients, en se basant sur des habitudes de consommations. En biologie, on peut l'utiliser pour dériver des taxinomies de plantes et d'animaux, pour catégoriser des gènes avec une ou plusieurs fonctionnalités similaires, pour mieux comprendre les structures propres aux populations. La classification peut tout aussi bien aider dans l'identification des zones de paysage similaire, utilisée dans l'observation de la terre, et dans l'identification de groupes de détenteurs de police d'assurance automobile ayant un coût moyen d'indemnisation élevé, ou bien dans la reconnaissance de groupes d'habitation dans une ville, selon le type, la valeur et la localisation géographique. Il est possible aussi de classifier des documents sur le Web, pour obtenir de l'information utile. Sa fonction dans un processus de Fouille de Données sera de découvrir la distribution des données, d'observer les caractéristiques de chaque groupe et de se focaliser pourquoi pas sur un ou plusieurs ensembles particuliers d'objets pour de prochaines analyses. Parallèlement, elle peut servir comme une étape de préparation de données, appelé aussi pre-processing, pour d'autres méthodes. Ainsi, ce type d'analyse est devenu un domaine hautement actif dans la recherche de type Fouille de Données.

En tant que branche de la statistique, la classification automatique a été énormément étudié depuis de nombreuses années, en se basant principalement sur des distances. Les algorithmes ainsi développés sont $k$-means, $k$-medoids, et plusieurs autres, disponibles dans différents logiciels d'analyse statistique. Dans le domaine de l'apprentissage automatique, la classification est un exemple d'**apprentissage non-supervisée**, contrairement à l'analyse discriminante, exemple typique d'**apprentissage supervisée**, où un *professeur* indique le résultat que l'on doit avoir (ici, la classe d'appartenance). Dans cette méthode, on cherche donc à comprendre comment on

obtient ce résultat connu. Dans les méthodes d'apprentissage non-supervisé, on cherche à retrouver ce résultat, comme par exemple une partition. On apprend donc ici avec des observations, et non avec des exemples.

On peut distinguer deux grandes familles de méthodes de classification : les **méthodes de partitionnement simple** et les **méthodes hiérarchiques**. Les premières cherchent une partition des objets, ou bien des variables, en un certain nombre de classes $s$ donné (cf $k$-means et nuées dynamiques). Deux critères doivent être satisfaits :

- Chaque groupe doit contenir au moins un objet, et donc les classes vides ne sont pas tolérées.
- Chaque objet doit appartenir à un seul groupe. Il est à noter que des versions floues tempèrent ce critère et permettent à un objet d'appartenir à plusieurs classes selon un certain degré (voire une certaine probabilité).

Un critère généralement utilisé pour savoir si une partition est bonne, est que les objets d'une même classe doivent être très *proches* les uns des autres, et très *éloignés* des autres classes. Cette notion de proximité ou d'éloignement induit forcément un calcul de distance ou plus généralement de dissimilarité. Il existe d'autres critères de nature différente pour juger de la qualité d'une partition.

Afin d'obtenir la meilleure partition, il faudrait une énumération exhaustive de toutes les partitions possibles. Pour 100 objets, et 4 classes, il nous faudrait donc explorer 100 millions de partitions possibles, ce qui prendrait énormément de temps pour un tableau de telle taille. On utilise donc une heuristique qui consiste à améliorer une partition étape par étape jusqu'à la convergence du critère de qualité de la partition. Le résultat obtenu est donc un optimum local de la solution, dépendant du point de départ de l'algorithme. Pour surmonter cet handicap, il faut lancer plusieurs fois l'algorithme avec des initialisations différentes.

Les autres méthodes de classification, dites de classification hiérarchique, consistent à créer une décomposition hiérarchique d'un tableau de données. On peut envisager deux stratégies : **ascendante** ou **descendante**. L'approche ascendante démarre avec chaque objet formant une classe distincte. On fusionne à chaque étape les deux classes les plus proches afin de n'obtenir plus qu'une seule classe, ou alors qu'une condition de terminaison soit vérifiée. L'approche descendante démarre avec tous les objets dans une seule et même classe. A chaque itération, une classe est décomposée en classes plus petites, jusqu'à n'avoir plus qu'un seul objet dans chaque classe, ou éventuellement qu'une condition d'arrêt soit vérifiée. Ici aussi, il est nécessaire d'introduire une notion de similarité ou de dissimilarité entre les objets et les classes.

Un des inconvénients de ce type de méthodes est qu'une action effectuée (fusion ou décomposition), elle ne peut être annulée. Cette rigidité permet de réduire le champ d'exploration. Malgré tout, un problème majeur est qu'un telle technique ne peut corriger une décision erronée. Pour améliorer la qualité d'une classification hiérarchique, il y a deux possibilités :

- analyser attentivement les liens entre objets à chaque étape (cf CURE (Guha *et al.*, 1998) et Chameleon (Karypis *et al.*, 1999)),
- améliorer la partition obtenue avec une classification de type partitionnement simple (cf BIRCH (Zhang *et al.*, 1996)).

Comme nous l'avons fait remarqué, ces deux approches de la classification font appel à une notion soit de dissimilarité, soit de similarité. Il est possible de passer ainsi par une distance. Dans le cas de données continues, on prend généralement la distance euclidienne, et dans le cas de données binaires, la distance L1. Dans le premier cas, lorsque nous appliquons un algorithme de types centres mobiles (ou $k$-means), avec une telle distance, nous obtenons généralement des classes sphériques. Or, il est important de ne pas faire d'hypothèses trop fortes sur la forme des classes à obtenir. De plus, la forme des classes n'est pas forcément la même pour toutes. Ainsi, à la vue des limitations imposées par cette notion de distance, un intérêt certain s'est développé autour de l'approche probabiliste, avec la notion de modèle de mélange. Un premier avantage est de fournir un cadre théorique cohérent. Et de plus, il nous est ainsi possible de retrouver et de mieux comprendre les méthodes de classification existantes, comme les centres mobiles par exemple.

Dans ce cadre, la partition optimale peut alors être obtenue en utilisant deux approches différentes : l'approche **Estimation** et l'approche **Classification**. La première cherche à maximiser la log-vraisemblance, alors que la seconde maximise la log-vraisemblance complétée (Symons, 1981). Pour l'approche Estimation, on estime les paramètres du modèle de mélange, puis, on en déduit la partition. Dans l'approche Classification, la partition constitue elle-même le paramètre à estimer. Dans ces deux approches, il est possible de définir des critères de choix du nombre de classes et du choix du modèle.

Pour maximiser la log-vraisemblance, il est courant de recourir à des algorithmes de type **EM** (Dempster *et al.*, 1977) (Estimation - Maximisation). En effet, cet algorithme est basé sur deux étapes : la première (Estimation) qui consiste à calculer les probabilités a posteriori d'appartenance d'un objet à une classe, et la seconde (Maximisation) qui consiste à estimer les paramètres inconnus. Ces deux étapes sont répétées jusqu'à la convergence de l'algorithme. Celui-ci fournit une solution qui est un optimum local, l'ensemble des solutions ne pouvant être exploré. Il arrive parfois que la convergence amène une solution qui est sur un point selle. Pour pallier cet inconvénient, des versions stochastiques de EM ont vu le jour, comme **SEM** (Broniatowski *et al.*, 1983; Celeux et Diebolt, 1985), **SAEM** (Celeux et Diebolt, 1992) et **MCEM** (Wei et Tanner, 1990; Tanner, 1991).

Et pour maximiser la log-vraisemblance complétée, un algorithme dérivé de EM est utilisé, l'algorithme **CEM** (Celeux et Govaert, 1992). Entre les deux étapes Estimation et Maximisation, une autre étape de Classification a été rajoutée, qui va allouer chaque objet à la classe la plus proche (par la méthode du maximum a posteriori -MAP-), passant ainsi d'une partition floue à une partition dure. Une version de type recuit simulé a été développé, **CAEM** (Celeux et Govaert, 1992).

Lorsqu'on applique un algorithme de classification simple, il est nécessaire de connaître le nombre de classes dans la population. Or, dans la majeure partie des applications de la classification, cette information n'est pas connue. Il faut donc pouvoir remédier à cet inconvénient. Une des premières idées est de calculer un critère pour plusieurs nombres de classes candidats, ne retenant que celui qui optimise ce critère. Une deuxième idée serait d'utiliser une méthode de classification hiérarchique. En effet, celle-ci ne requiert pas la connaissance du nombre de classes.

Malheureusement, les avantages d'une telle méthode ne peuvent pas rattraper son principal inconvénient, sa complexité. De par le nombre de calculs de distances à effectuer à chaque étape, un algorithme de ce type prend un temps de plus en plus conséquent avec un nombre d'objets croissant, ce temps devenant rapidement ingérable. Donc, il nous est impossible de lancer de tels algorithmes sur des tableaux de grande taille. Une solution est donc de combiner les deux types d'algorithmes (simple et hiérarchique), comme l'a proposé Wong (1977).

Lors de la récolte des informations, il arrive souvent qu'une partie de celle-ci soit manquantes ou erronées. On en arrive donc à devoir analyser des tableaux où certaines cases manquent. Il faut donc pouvoir pallier ce problème en proposant une solution. Une première idée consiste à *remplir* ces cases vides en leur affectant par exemple la moyenne de la variable ou la modalité la plus choisie, ou bien à considérer une donnée manquante, dans le cas de données qualitatives, comme une nouvelle réponse possible, comme dans AutoClass (Cheeseman et Stutz, 1996). Cette dernière possibilité part du principe qu'une non-réponse équivaut peut-être à une réponse particulière de la personne (ce qui peut être vrai dans des cas comme le salaire d'une personne ou bien la quantité d'alcool bu par jour). Une deuxième idée, largement utilisée dans la Fouille de Données, au vue du nombre faramineux de données disponibles parfois, est de supprimer les objets ou les variables (selon le cas) qui comportent des données manquantes. Cette approche a le mérite de nettoyer la base de données, mais peut-être trop radicalement. En effet, ne perdons-nous pas ici un certain nombre d'informations? Certains répondent qu'il est possible d'analyser a posteriori les objets présentant des données manquantes. Une troisième idée est de gérer statistiquement ces données manquantes. Ceci peut se faire sous une approche statistique de la classification, et donc avec un modèle de mélange.

Ensuite, il est connu que l'algorithme EM, qui possède beaucoup de qualité, peut avoir une lente convergence. Ainsi, dans le cadre de la Fouille de Données, son application peut ainsi être inadaptée. C'est pourquoi il est nécessaire de s'intéresser à l'accélération de cette méthode, par différents moyens. Mais il est absolument nécessaire que ces techniques d'accélération nous permettent d'obtenir des résultats similaires ou approchant (en terme d'estimation de paramètres et de partition) de ceux obtenus avec EM.

Un autre aspect de la classification est la recherche simultanée d'une partition en lignes et d'une partition en colonnes. Celle-ci se fait parfois en lançant un algorithme de classification simple, comme $k$-means, sur les objets puis sur les variables. Or, une telle méthode ne permet pas d'expliquer la relation spécifique pouvant exister entre un groupe d'objets et un groupe de variables. Ainsi, il est préférable d'appliquer des algorithmes de classification croisée, ou de classification directe.

En conclusion, selon (Han et Kamber, 2001), une bonne méthode de classification automatique, dans le domaine de la Fouille de Données, doit :

- traiter de grandes bases de données, aussi bien au niveau du nombre d'objets (plusieurs milliers, voire plusieurs millions) que du nombre de variables (plusieurs centaines),
- traiter des données de tout type (numériques, binaires, catégorielles, ordinales, ...),

- découvrir des classes de profil arbitraire (sans a priori sur leurs formes),
- requérir très peu de connaissance du domaine d'application,
- gérer des données bruitées (outliers, données manquantes, données erronées, ...),
- être paramètrable pour pouvoir satisfaire certaines contraintes,
- produire des résultats simples, clairs, et utilisables directement.

Dans un premier temps, nous allons présenter la classification automatique simple, ainsi que certains algorithmes associés. Nous développerons les prémices de cette méthode, avec des méthodes comme celle des nuées dynamiques (Diday, 1971; Diday, 1975), et nous verrons son évolution jusqu'à la généralisation avec des modèles de mélange Gaussien.

Ensuite, nous aborderons dans le deuxième chapitre l'analyse de données qualitatives, domaine que nous privilégions. Nous appliquerons ici les principes des modèles de mélange pour présenter les versions de EM et de CEM adaptées à ce type de données. Nous verrons leur application sur quelques données réelles. Ensuite, nous aborderons le problème de la classification avec données manquantes.

Afin de proposer une solution au choix du nombre de classes, nous présentons dans le chapitre 3 un ensemble de critères, dont certains sont développés dans un cadre bayésien. Après une étude détaillé de ceux-ci, nous proposons d'étendre la méthode hybride de Wong (1977) au modèle de mélange. Nous proposons ainsi une application sur des données simulées et réelles.

Dans le chapitre 4, nous nous intéressons à l'application de la classification automatique dans le cadre de la Fouille de Données. Le principal challenge ici est d'accélérer la convergence de l'algorithme EM. Pour ceci, nous verrons tout d'abord différentes stratégies d'initialisation. Ensuite, nous analyserons diverses techniques d'accélération, se basant sur une étape d'Estimation des probabilités a posteriori, optimisée en terme de coût de calcul.

Le cinquième chapitre présente la classification croisée et la classification directe. La première méthode cherche simultanément une partition en lignes et une partition en colonnes sur un tableau de données. Nous verrons quelques algorithmes de ce type, optimisant des critères métriques (Govaert, 1983; Govaert, 1995). La seconde méthode, introduite par Hartigan (1972, 1975), cherche une structure des données plus fine, avec des blocs de données homogènes, et de toutes tailles. Nous étudierons en détail cette méthode sur plusieurs tableaux.

Dans le chapitre 6, nous abordons l'extension de deux algorithmes précédemment présentés au modèle de mélange croisé. En agissant d'une manière similaire au cas de la classification simple, nous détaillons les modèles utilisées et les algorithmes ainsi obtenus. Ensuite, nous présentons une méthode de classification croisée hiérarchique. Puis, nous présentons deux méthodes hybrides, combinant des algorithmes de classification croisée, afin de pouvoir travailler sur des données de grandes tailles.

# Chapitre 1

# Principales approches de Classification

La classification ou le regroupement en classes homogènes consistent à réduire un nuage des points d'un espace quelconque en un ensemble de représentants moins nombreux permettant une représentation simplifiée des données initiales. Ainsi, comme les méthodes d'analyse factorielle, la classification automatique est une méthode de réduction des données. Il s'agit d'une démarche très courante qui permet de mieux comprendre l'ensemble analysé. Ces applications sont nombreuses, en statistique, traitement d'image, intelligence artificielle, reconnaissance des formes ou encore la compression de données. Pour toutes ces raisons, elle constitue une étape importante dans le processus de Fouille des Données.

On peut distinguer deux grandes familles de classification : par partitionnement et par hiérarchie. Dans ce chapitre, nous allons d'abord rappeler les méthodes de partitionnement couramment utilisées et disponibles dans la plupart des logiciels, puis les méthodes hiérarchiques. Comme nous nous inscrivons, le long de ce travail, dans un cadre probabiliste, nous allons rappeler l'approche mélange et ses avantages. En effet, cette approche est un outil puissant qui permet entre autres de mieux comprendre la plupart des méthodes des deux grandes familles citées.

Dans tous les chapitres, nous utiliserons les notations suivantes. Soit $\mathbf{x}$ une matrice de données $n \times d$ définie par $\mathbf{x} = \{x_i^j ; i \in I ; j \in J\}$, où $I$ est un ensemble de $n$ objets (lignes, observations, instances, individus) et $J$ est un ensemble de $d$ variables (colonnes, attributs).

## 1.1 Méthodes de partitionnement classiques

**Définition d'une partition.** Une partition de $I$ en $s$ classes ($s$ est supposé connu) est un ensemble de parties non vides de $I$, $z_1, \ldots, z_s$ vérifiant

- $\forall\ k,k' = 1, \ldots, s,\ k \neq k',\ z_k \cap z'_k = \emptyset$,
- $\cup_{k=1}^{s}\ z_k = I$.

Le nombre de partitions possibles est très important même lorsque la cardinalité de $I$ n'est pas grande. Si l'on considère le partitionnement de cet ensemble en $s$ classes, le nombre total de partitions possibles est égal à :

$$\frac{1}{s!}\sum_{k=0}^{s}(-1)^{k-1}\times\left(\begin{array}{c}s\\k\end{array}\right)\times k^n$$

Par exemple, il existe 1701 partitions possibles de 8 objets répartis en 4 classes. Donc, plutôt que chercher la meilleure partition, on se contente d'en proposer une qualifiée de *profitable* qui optimise un critère donné. Celui-ci bien entendu, respectera l'objectif d'homogénéité des classes. Les méthodes utilisées sont souvent conçus d'un point de vue heuristique et utilisent des critères métriques.

### 1.1.1 La méthode des centres mobiles

Ce type de méthodes repose généralement sur des algorithmes simples, et permet de traiter rapidement des ensembles d'effectif assez élevé en optimisant localement un critère. Le plus célèbre de ces algorithmes est incontestablement l'algorithme des **centres mobiles (k-means)** (Forgy, 1965; MacQueen, 1967). Il consiste à minimiser le critère suivant:

$$W(\mathbf{z},\mathbf{g}) = \sum_{k=1}^{s}\sum_{i\in z_k} d^2(\mathbf{x}_i,\mathbf{g}_k), \tag{1.1}$$

où $\mathbf{g} = (\mathbf{g}_1,\ldots,\mathbf{g}_s)$, $\mathbf{g}_k$ est le centre de la classe $z_k$. Rappelons que le critère $W(\mathbf{z},\mathbf{g})$, qui est simplement la somme des inerties des $s$ classes, est appelé inertie intraclasse. La méthode des centres mobiles consiste à chercher la partition telle que le $W$ soit minimal pour avoir en moyenne des classes bien homogènes, ce qui revient à chercher le maximum de l'inertie interclasse,

$$B(\mathbf{z}) = \sum_{k=1}^{s} p_k d^2(\mathbf{g}_k,\bar{\mathbf{g}}), \tag{1.2}$$

avec $\bar{\mathbf{g}}$ le centre de gravité de l'ensemble $I$ et $p_k$ est le poids de la classe $z_k$. Ce résultat est dû à la relation liant l'inertie totale qui ne dépend pas de la partition et les inerties interclasse et intraclasse

$$I = W(\mathbf{z},\mathbf{g}) + B(\mathbf{z}).$$

Cet algorithme se déroule de la façon suivante:

- Initialisation: $s$ points tirés au hasard pour les centres de gravité de chaque classe,
- Affectation: On affecte les points à la classe la plus proche,
- Représentation: On recalcule les nouveaux centres de gravité,
- On répète les étapes d'affectation et de représentation jusqu'à la convergence de l'algorithme (i. e. plus de changement de le partition).

MacQueen et Forgy ont deux approches différentes. L'algorithme de MacQueen recalcule les nouveaux centres dès qu'un objet a été transféré d'une classe à une autre. Alors que dans l'algorithme de Forgy, le calcul des centres de gravité intervient une fois que tous les objets ont été réaffectées à la classe la plus proche.

Les expériences montrent que le nombre d'itérations nécessaires à l'algorithme pour converger est assez faible (généralement inférieur à 20). Cet algorithme est donc adapté à des tableaux de grands tailles, sa complexité étant linéaire.

Une méthode de classification plus générale existe, ce sont les **nuées dynamiques** (Diday, 1971). La différence primordiale est que dans cette méthode, les classes peuvent avoir plusieurs représentations possibles: un point de la classe, une loi de probabilité, un axe factoriel, un groupe de points, ... Dans le cas où la représentation d'une classe est son centre de gravité, on retombe donc sur la méthode des centres mobiles.

Malgré sa popularité, un des inconvénients de l'algorithme $k$-means est qu'il tend à trouver des classes sphériques de même taille. En présence d'une classe de très petite taille, ou d'une classe prédominante, cette méthode va donc avoir tendance à vider une classe et la partition ainsi obtenue ne reflétera donc pas correctement la structure des données en classes. Tous ces aspects seront discuter grâce à l'approche mélange.

### 1.1.2 Données binaires

Lorsque l'ensemble des objets est décrit par un ensemble de variables binaires, souvent on est amené à utiliser l'algorithme des $k$-means pour la classification des objets. Autrement dit, le critère utilisé est celui de l'inertie intra-classe qui repose sur une distance euclidienne. Les centres des classes sont par conséquent de nature différente des données à classifiés 0,1. Afin de respecter le principe d'homogénéité, on impose aux centres d'avoir la même structure que les données et de cette façon, il suffit de chercher à minimiser le critère suivant à l'aide de l'algorithme des nuées dynamiques

$$W(\mathbf{z},\mathbf{g}) = \sum_{k=1}^{s} \sum_{i \in z_k} \mathbf{d}(\mathbf{x}_i,\mathbf{g}_k), \tag{1.3}$$

où $d(\mathbf{x}_i,\mathbf{g}_k) = \sum_{j=1}^{d} |x_i^j - g_k^j|$ avec $g_k^j \in \{0,1\}$.

### 1.1.3 Données qualitatives

Pour classifier l'ensemble des objets, il est habituel d'utiliser $k$-means avec la métrique du $\chi^2$ (voir par exemple (Benzecri, 1973; Ralambondrainy, 1988)). Dans ce cas, le critère, dit critère du $\chi^2$, s'écrit:

$$W(\mathbf{z},\mathbf{g}) = \sum_{k=1}^{s} \sum_{i \in z_k} \mathbf{d}_{\chi^2}^2(\mathbf{f}_i,\mathbf{g}_k), \tag{1.4}$$

où $\mathbf{f}_i = (\frac{f_{i11}}{f_{i.}}, \ldots, \frac{f_{ije}}{f_{i.}})$ est le profil de l'objet $i$ avec $f_{ije} = \frac{x_i^{je}}{nm}$ ($m$ représente le nombre totale de modalité) et $f_{i.} = \sum_{j=1}^{d} \sum_{e=1}^{c_j} f_{ije} = \frac{1}{n}$. Notons que ce critère peut s'écrire sous la forme suivante:

$$W(\mathbf{z},\mathbf{g}) = nm \sum_{k=1}^{s} \sum_{i \in z_k} \sum_{j=1}^{d} \sum_{e=1}^{c_j} \frac{1}{n^{je}} \left( \frac{x_i^{je}}{m} - g_k^{je} \right)^2,$$

où $n^{je} = \sum_{k=1}^{s} \sum_{i \in z_k} x_i^{je}$ est le nombre d'objets ayant choisi la modalité $e$ pour la variable $j$. La minimisation de ce critère est équivalente en fait à la maximisation du critère du $\chi^2(\mathbf{z},J)$ qui s'écrit

$$\chi^2(\mathbf{z},J) = \sum_{k=1}^{s} \sum_{j=1}^{d} \sum_{e=1}^{c_j} \frac{(nmx_k^{je} - x_k n^{je})^2}{nmx_k n^{je}}$$

où $x_k^{je} = \sum_{i \in z_k} x_i^{je}$ et $x_k = \sum_{j=1}^{d} \sum_{e=1}^{c_j} x_k^{je}$. Rappelons qu'en pratique, ce critère donne les mêmes résultats que le critère l'information mutuelle (Benzecri, 1973; Govaert, 1983) qui exprime l'information conservée par la partition $\mathbf{z}$, et qui s'écrit :

$$H(\mathbf{z}) = nm \sum_{k=1}^{s} \sum_{j=1}^{d} \sum_{e=1}^{c_j} \frac{x_k^{je}}{nm} \log_2 \left( \frac{x_k^{je} nd}{x_k n^{je}} \right), \tag{1.5}$$

Notons que la maximisation de ce critère est en fait équivalente à la maximisation de

$$H(\mathbf{z}) = \sum_{k=1}^{s} \sum_{j=1}^{d} \sum_{e=1}^{c_j} x_k^{je} \log x_k^{je} - d \sum_{k=1}^{s} \#z_k \log \#z_k \tag{1.6}$$

En fait le premier est largement plus répandu pour deux raisons : la première est qu'on a longtemps cru que ce critère était plus simple à optimiser alors qu'on verra ultérieurement que celui de l'information mutuelle l'est aussi (Celeux, 1988). La deuxième raison est qu'il est naturellement lié à l'analyse des correspondances. Notons que cette méthode fournit des centres qui ne sont pas de la même forme que les éléments à classifier. Comme dans le cas binaire, en imposant à ces centres d'avoir la même structure que les données initiales, c'est à dire des modalités, Marchetti (1989) et Huang (1997) ont proposé de minimiser le critère suivant :

$$E = \sum_{k=1}^{s} \sum_{i \in z_k} \mathbf{d}(\mathbf{x}_i, m_k). \tag{1.7}$$

où $m_k$ est le centre de la $k$ème classe $z_k$ et $d$ est mesure de dissimilarité. Les auteurs proposent d'utiliser la mesure $d$ définie par :

$$d(\mathbf{x}_i, m_k) = \sum_{j=1}^{d} \Delta(x_i^j, m_k^j) \text{ où } \Delta(x,y) = \begin{cases} 0 & \text{si } x = y \\ 1 & \text{sinon} \end{cases} \tag{1.8}$$

Comme $\mathbf{d}(\mathbf{x}_i, \mathbf{x}_{i'})$ exprime simplement le nombre total de différence entre deux objets $\mathbf{x}_i$ et $\mathbf{x}_{i'}$, ce critère est donc simplement le nombre total des différences entres les objets de chaque classe $z_k$ et son centre $m_k$. Le déroulement de l'algorithme est le même que $k$-means, nous ne le détaillons pas ici. Il est à noter par contre qu'il est inutile de procéder au codage du tableau de modalités pour pouvoir appliquer cet algorithme. En utilisant le critère $E$, la partition obtenue est aisée à commenter. Chaque classe $z_k$ est ainsi caractérisée par un centre $m_k = (m_k^1, \ldots, m_k^d)$ où $m_k^j$ est la modalité la plus choisie de la variable $\mathbf{x}^j$ dans la classe $z_k$. Par simplicité, les vecteurs $m_k$ sont appelés *modes*.

Comme l'algorithme $k$-means, cet algorithme connu maintenant sous le nom de $k$-modes (Huang, 1997) produit aussi des solutions optimales localement, qui sont

influencés par l'étape d'initialisation. Huang (1997) suggère deux méthodes d'initialisation. La première est basique et consiste à prendre les $s$ premiers objets distincts. La seconde est basée sur les fréquences des modalités des variables (pour plus de détail, voir (Huang, 1997)). Même si la propriété majeure de cet algorithme est sa simplicité et une convergence rapide, nous donnerons une interprétation probabiliste du critère $E$ et étudierons son comportement à travers des simulations dans le prochain chapitre.

### 1.1.4 Classification floue (*fuzzy clustering*)

La classification floue, et particulièrement fuzzy $c$-means (**FCM**), est une méthode de classification non-supervisée, dérivée de l'algorithme $c$-means (Ball et Hall, 1967), identique à l'algorithme $k$-means décrit précédemment. Cet algorithme est donc basé sur la minimisation de la somme aux carrés des distances euclidiennes entre les objets et les centres des classes, qui minimise indirectement la variance suivante :

$$J_1(U,V) = \sum_{k=1}^{s}\sum_{i=1}^{n}(u_{ik})^2\mathbf{d}_M^2(\mathbf{x}_i,v_k) \tag{1.9}$$

Ici, $U = [u_{ik}]$ représente une matrice de partition *dure* ($u_{ik} \in \{0,1\}$ et $\sum_{k=1}^{s} u_{ik} = 1$) et $V = \{v_k\}$ le vecteur des centres des classes. Chaque objet $\mathbf{x}_i$ appartient à une seule classe, et chaque classe $k$ contient au moins un objet. Dunn (1974) a étendu en premier l'algorithme $c$-means dur afin d'autoriser une partition floue avec la fonction suivante :

$$J_2(U,V) = \sum_{k=1}^{s}\sum_{i=1}^{n}(\mu_{ik})^2\mathbf{d}_M^2(x_i,v_k) \tag{1.10}$$

Cette fois-ci, $U = [\mu_{ik}]$ correspond à une partition floue en $s$ classes ($\mu_{ik} \in [0,1]\ \forall i,k$ et $\sum_{k=1}^{s}\mu_{ik} = 1\ \forall i$), Donc, chaque objet $\mathbf{x}_i$ peut appartenir à plus d'une classe avec un degré d'appartenance prenant une valeur entre 0 et 1. Bezdek (1981) généralisa $J_2(U,V)$ en un nombre infini de fonctions $J_m(U,V)$, où $1 \leq m \leq \infty$. La nouvelle fonction, sujet aux mêmes contraintes, est maintenant :

$$J_m(U,V) = \sum_{k=1}^{c}\sum_{i=1}^{n}(\mu_{ik})^m\mathbf{d}_M^2(x_i,v_k) \tag{1.11}$$

Pour la minimisation de ces trois fonctions, on utilise l'algorithme suivant :

1. $c$ centres choisis au hasard,
2. Calcul des $u_{ik}$ (ou $\mu_{ik}$) : degré d'appartenance aux classes,
3. Calcul des $v_k$ : centre des classes.
4. On réitère ces deux étapes (2 et 3) jusqu'à la convergence de $J$.

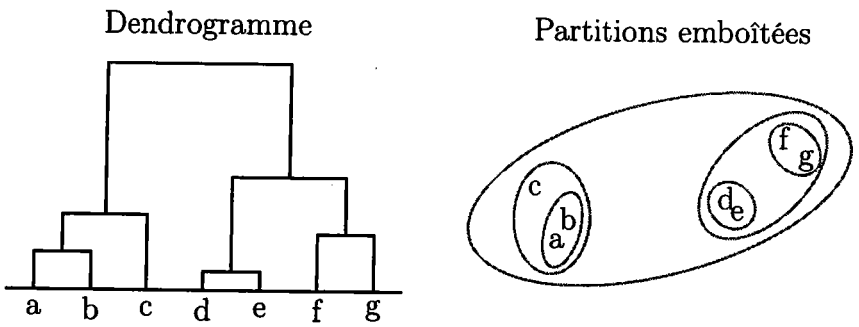
On retrouve dans cet algorithme les mêmes inconvénients que pour $k$-means au sujet des proportions et les formes des classes. Par contre, la complexité de cet algorithme est aussi linéaire, donc il est adapté à des données de grande taille.

### 1.1.5 Classification ascendante hiérarchique, CAH

**Définition d'une hiérarchie.** Un ensemble $H$ de parties non vides de $I$ est une hiérarchie sur $I$ si

- $I \in H$
- $\forall i \in I \ \{i\} \in H$
- $\forall h,h' \in H$, $h \cap h' = \emptyset$ ou $h \subset h'$ ou $h' \subset h$.

Une hiérarchie peut être vue comme un ensemble de partitions emboîtées. Graphiquement, une hiérarchie est souvent représentée par une structure arborescente représentée par un arbre hiérarchique dit aussi dendrogramme.




Il existe deux types de familles de méthodes: une descendante, dite divisive, et une ascendante, dite agglomérative. La première, moins utilisée, consiste à partir d'une seule classe regroupant tous les objets, à partager celle-ci en deux. Cette opération est répétée à chaque itération jusqu'à ce que toutes les classes soient réduites à des singletons. La seconde qui est la plus couramment utilisée consiste, à partir des objets (chacun est dans sa propre classe), à agglomérer les classes les plus proches, afin de n'en obtenir plus qu'une seule contenant tous les objets. S'il est assez aisé de calculer une distance entre deux points, il est moins évident de calculer une distance entre une classe et un point, ou encore entre deux classes. Plusieurs distances classiques $\delta$ dites critères d'agrégation existent. Les plus couramment utilisés sont: critère du lien minimum, critère du lien maximum, critère du lien moyen et le critère de Ward qui résulte de la perte d'inertie en regroupant deux classes $z_1$ et $z_2$ et qui s'écrit:

$$\delta_{ward}(z_1,z_2) = \frac{\#z_1 \times \#z_2}{\#z_1 + \#z_2}\mathbf{d}^2(z_1,z_2)$$

À une hiérarchie est associé un indice qui est une fonction strictement croissante et tel que pour toute classe singleton son indice est nul. Ainsi, pour les classes du bas de la hiérarchie l'indice vaut 0 et pour les autres classes, cet indice est défini en associant à chacune des classes construites au cours de la méthode la distance $\delta$ qui séparaient les deux classes fusionnées pour former cette nouvelle classe. Les critères d'agrégation cités précédemment conduisent bien à un indice, d'autres critères par contre présentent quelques difficultés. Ci-dessous, nous décrivons les principales étapes de l'algorithme de classification ascendante hiérarchique (**CAH**):

1. Au départ, chaque objet est dans sa propre classe,

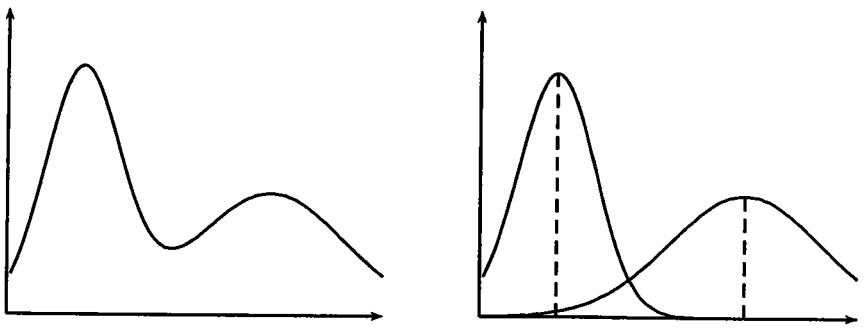


FIG. 1.1 – *Décomposition d'une variable continue en deux variables avec des moyennes et des écarts-type différents.*

2. On fusionne les deux classes les plus proches (selon le critère choisi),
3. On répète l'étape 2 jusqu'à n'avoir plus qu'une seule classe.

La complexité d'un tel algorithme est quadratique. Ceci nous restreint donc à l'application de cette méthode sur des tableaux de taille raisonnable. Dans un contexte de fouille de données, un tel algorithme est assez peu utilisé, on se contente souvent de l'appliquer sur des échantillons de l'ensemble des données ou encore sur des résumés des données obtenus précédemment avec une autre méthode comme nous le verrons ultérieurement.

Par contre, en analysant l'évolution du critère, nous sommes capables de déterminer un nombre de classes approprié. Et donc, à l'inverse des méthodes de classification directe (comme $k$-means et FCM), nous n'avons donc pas besoin ici de la connaissance a priori du nombre de classes.

## 1.2 Modèle de mélange

### 1.2.1 Cas général

Dans la méthode des mélanges, on suppose que l'ensemble de la population est représenté par une distribution de probabilité qui est un mélange de $s$ distributions de probabilités associées aux classes. L'objectif de cette démarche est d'identifier les $s$ distributions en estimant leurs paramètres. Ce problème du mélange a été depuis longtemps étudié dans le cadre univarié et multivarié (Day, 1969; Wolfe, 1970).

Pour expliquer la notion de modèle de mélange, nous allons regarder le cas d'une variable continue, où deux classes sont présentes (voir Figure 1.1). La figure de gauche représente la distribution de la variable. On remarque qu'il y a deux maximums locaux. Nous pouvons décomposer dès lors cette variable en deux variables distinctes, ayant chacune une moyenne et une variance propre, comme indiqué sur la figure de droite. On voit bien ici l'intérêt du modèle de mélange pour décomposer une population en plusieurs classes.

Le tableau de données $\mathbf{x}$ est considéré comme un échantillon $(\mathbf{x}_1, \ldots, \mathbf{x}_n)$ i.i.d. de taille $n$ d'une variable aléatoire à valeurs dans $\mathbb{R}$ dont la loi de probabilité admet la densité $\varphi(\mathbf{x},\theta)$, qui correspond à une probabilité dans le cas discret, définie par :

$$\forall \mathbf{x}_i \ \varphi(\mathbf{x}_i;\theta) = \sum_{k=1}^{s} p_k \varphi_k(\mathbf{x}_i;\alpha_k). \tag{1.12}$$

avec :

$$\forall k = 1,\ldots,s,\ p_k \in ]0,1[ \text{ et } \sum_{k=1}^{s} p_k = 1$$

où

- $\varphi(\mathbf{x}_i,\alpha_k)$ est une densité de probabilité appartenant à une famille de densités de probabilités (dépendant du paramètre $\alpha \in \mathbb{R}^s$, $s \geq 1$).
- $p_k$ est la probabilité qu'un élément de l'échantillon suive la loi $\varphi$. On appellera ces $p_k$ les proportions du mélange.
- $\theta = (p_1,\ldots,p_s;\alpha_1,\ldots,\alpha_s)$ désigne le paramètre du modèle de mélange, qui est inconnu.

Le problème statistique consiste à estimer les proportions des composants $(p_1,\ldots,p_s)$ et les paramètres $(\alpha_1,\ldots,\alpha_s)$. La vraisemblance offre une approche générale à l'estimation de ces paramètres à l'aide des données de l'échantillon. La fonction de vraisemblance notée $V$ s'écrit dans notre cas :

$$V(\mathbf{x}_1,\ldots,\mathbf{x}_n;\theta) = \prod_{i=1}^{n} \varphi(\mathbf{x}_i;\theta) = \prod_{i=1}^{n} \sum_{k=1}^{s} p_k \varphi_k(\mathbf{x}_i;\alpha_k) \tag{1.13}$$

Notons que ce problème ne peut se résoudre dans toute sa généralité que si le mélange de lois appartient à une famille de mélanges identifiables. Tous les mélanges utilisées par la suite le seront. Pour des raisons de simplification de traitement, il est d'usage de travailler sur le logarithme de cette fonction $V$, que l'on nommera la log-vraisemblance $L$ et qui s'écrit

$$L(\mathbf{x}_1,\ldots,\mathbf{x}_n;\theta) = \sum_{i=1}^{n} \log\left(\sum_{k=1}^{s} p_k \varphi_k(\mathbf{x}_i;\alpha_k)\right) \tag{1.14}$$

Dans un contexte de classification, chaque $\mathbf{x}_i$ est supposé provenir d'une seule des $s$ classes du mélange. Pour chaque objet, nous aurons donc un vecteur $\mathbf{z}_i$ de dimension $s$, avec $z_{ik} = 1$ ou 0 selon que l'objet appartient à la classe $z_k$ ou non. Les données *complétées* seront représentées par $(\mathbf{x}_1,\mathbf{z}_1),\ldots,(\mathbf{x}_n,\mathbf{z}_n)$. A partir de cette formulation, nous obtenons la log-vraisemblance des données complétées ou log-vraisemblance complétée qui s'écrit :

$$L_c(\mathbf{x}_1,\ldots,\mathbf{x}_n;\theta) = \sum_{i=1}^{n} \sum_{k=1}^{s} z_{ik} \log\left(p_k \varphi_k(\mathbf{x}_i;\alpha_k)\right) \tag{1.15}$$

### 1.2.2 Cas gaussien

Dans le cas de données continues, on prend habituellement le modèle Gaussien. Dans ce cas, chaque objet $\mathbf{x}_i$ est supposé provenir indépendamment des autres, d'un mélange de densités

$$\varphi(\mathbf{x}_i;\theta) = \sum_{k=1}^{s} p_k \varphi_k(\mathbf{x}_i;\mathbf{g}_k,\Sigma_k)$$

où $\varphi_k$ représente la densité Gaussienne de dimension $d$, de moyenne $\mathbf{g}_k$ et de variance $\Sigma_k$,

$$\varphi_k(\mathbf{x}_i;\mathbf{g}_k,\Sigma_k) = (2\pi)^{-d/2}|\Sigma_k|^{-1/2}\exp\left(-\frac{1}{2}(\mathbf{x}_i-\mathbf{g}_k)'\Sigma_k^{-1}(\mathbf{x}_i-\mathbf{g}_k)\right)$$

et $\theta = (p_1,\ldots,p_{s-1},\mathbf{g}_1,\ldots,\mathbf{g}_s,\Sigma_1,\ldots,\Sigma_s)$ est le vecteur des paramètres du mélange. Ainsi, les classes associées aux composants du mélange sont ellipsoïdales, centrées sur la moyenne $\mathbf{g}_k$ et la matrice de variance $\Sigma_k$ détermine les caractéristiques géométriques.

### Décomposition de la matrice des variance

A partir de (Banfield et Raftery, 1993; Celeux et Govaert, 1995), nous considérons une paramétrisation de la matrice des variance des composants du mélange consistant à exprimer $\Sigma_k$ en fonction de sa décomposition en valeurs propres

$$\Sigma_k = \lambda_k D_k A_k D_k' \tag{1.16}$$

où $\lambda_k = |\Sigma_k|^{1/d}$, $D_k$ est la matrice des vecteurs propres de $\Sigma_k$ et $A_k$ est une matrice diagonale telle que $|A_k| = 1$, avec les valeurs propres normalisées de $\Sigma_k$ sur la diagonale, de façon décroissante. Le paramètre $\lambda_k$ détermine le *volume* de la classe $k$, $D_k$ son *orientation* et $A_k$ sa *forme*. En imposant certaines contraintes sur le volumes, les orientations ou les formes nous obtenons des modèles parcimonieux plus simples à interpréter qui peuvent être appropriés pour décrire diverses situations.

**La famille générale.** Dans un premier temps, nous autorisons les volumes, les formes et les orientations à varier entre les classes. Les variations sur les paramètres $\lambda_k$, $D_k$ et $A_k$ $(1 \leq k \leq s)$ conduisent à 8 modèles génériques. Par exemple, nous pouvons supposer des volumes $\lambda_k$ différents et garder les formes et les orientations égales en posant $A_k = A$ (non connu) et $D_k = D$ (non connu aussi), pour $k = 1,\ldots,s$. Nous notons ce modèle $\lfloor \lambda_k DAD' \rfloor$. Avec cette notation, utiliser le modèle $\lfloor \lambda D_k A D_k' \rfloor$ signifie considérer un modèle de mélange avec des volumes égaux, des formes égales, et des orientations différentes.

**La famille diagonale.** Une autre famille intéressante est celle où les matrices $\Sigma_k$ sont diagonales. Dans la paramétrisation (1.16), cela signifie que les matrices d'orientation sont des matrices de permutation. Nous écrivons alors $\Sigma_k = \lambda_k B_k$, où $B_k$ est une matrice diagonale. Cette paramétrisation particulière nous donne quatre modèles: $\lfloor \lambda B \rfloor$, $\lfloor \lambda_k B \rfloor$, $\lfloor \lambda B_k \rfloor$ et $\lfloor \lambda_k B_k \rfloor$.

**La famille sphérique.** La dernière famille de modèles consistent à supposer des formes sphériques, où l'on a $A_k = I$, $I$ étant la matrice Identité. Dans de tel cas, deux modèles parcimonieux sont en compétition: $\lfloor \lambda I \rfloor$ et $\lfloor \lambda_k I \rfloor$.

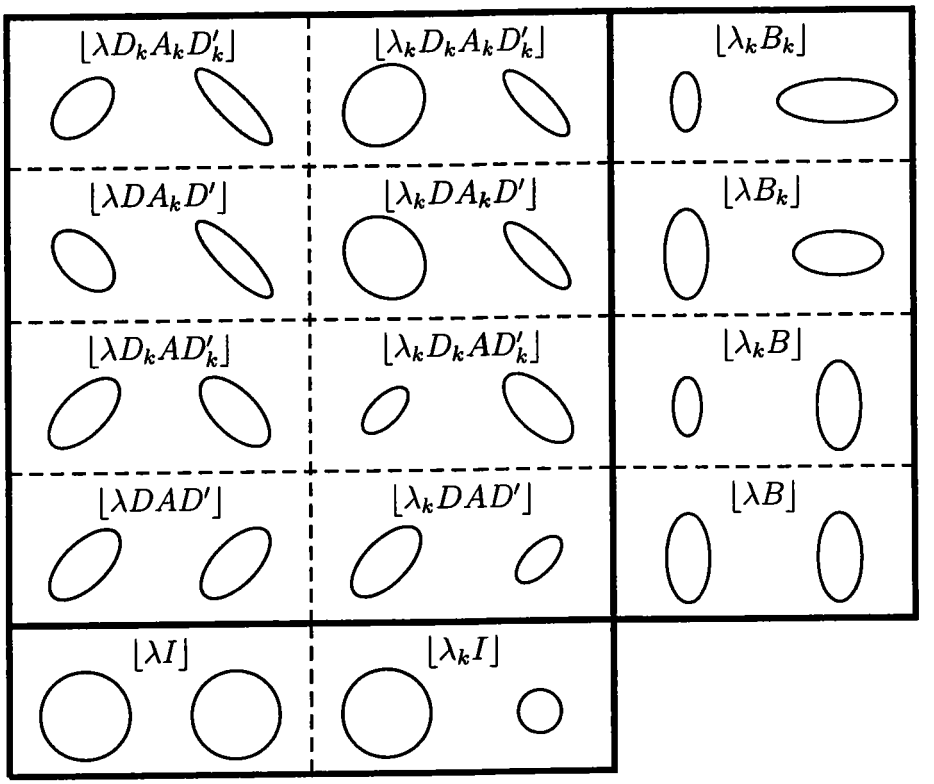



FIG. 1.2 – *Voici les 14 modèles, et les formes obtenues avec 2 classes sur 2 dimension.*

Finalement, nous avons donc 14 modèles différents. Le principal avantage de la décomposition en valeurs propres de la matrice de variance est l'interprétation simplifiée des modèles. Pour illustrer ce propos, la figure 1.2 montre les dessins des contours pour chaque modèle, pour $s = 2$ classes en dimension $d = 2$.

## 1.3 Différentes approches

Dans le cadre de la classification, la problème à résoudre consiste à retrouver pour chaque objet sa population d'origine la plus probable en fonction du vecteur d'observations qui le caractérise. Pour cela, il faut estimer préalablement les paramètres qui caractérisent la distribution des variables dans chacune des $s$ populations. Les principales techniques d'estimation utilisées pour estimer les paramètres d'un modèle de mélange sont celles des moments (Pearson, 1894) et celles du maximum de vraisemblance (Day, 1969; Wolfe, 1970). La méthode du maximum de vraisemblance est la technique la plus largement utilisée pour les problèmes d'estimation de densité de mélange, grâce notamment à l'arrivée de calculateurs à grandes vitesses et des méthodes numériques de plus en plus sophistiquées. Dans cette approche, le problème consiste à maximiser la log-vraisemblance $L$ (1.14). Elle consiste à résoudre itérativement les équations de vraisemblance et les algorithmes les plus efficaces sont de type **EM** (Estimation - Maximisation) de Dempster (Dempster *et al.*, 1977).

### 1.3.1 Approche estimation

En abordant le problème de la classification sous l'approche estimation, les paramètres sont d'abord estimés, puis la partition en est déduite par la méthode du maximum a posteriori (MAP). L'estimation des paramètres du modèle passe par la maximisation de $L(\mathbf{x},\theta)$. Une solution itérative pour la résolution de ce problème est l'algorithme EM composé de deux étapes Estimation et Maximisation (Dempster *et al.*, 1977). Basé sur l'information manquante, ici c'est la partition $\mathbf{z}$, le principe de cet algorithme est de remplacer la maximisation de la vraisemblance par la maximisation de manière itérative de l'espérance de la log-vraisemblance complétée conditionnellement aux données $\mathbf{x}$ et la valeur du paramètre courant $\theta^{(n)}$

$$Q(\theta|\theta^{(q)}) = \mathbf{E}[\log\varphi(\mathbf{x},\mathbf{z};\theta)|\mathbf{x},\theta^{(q)}] \tag{1.17}$$

En effet, cette quantité peut s'écrire

$$\begin{aligned} Q(\theta|\theta^{(q)}) &= \log\varphi(\mathbf{x};\theta) + \mathbf{E}[\log\varphi(\mathbf{z};\theta)|\mathbf{x},\theta^{(q)}] \\ &= L(\mathbf{x},\theta) + H(\theta|\theta^{(q)}) \end{aligned} \tag{1.18}$$

où $H(\theta,\theta^{(q)}) = \mathbf{E}[\log\varphi(\mathbf{z};\theta)|\mathbf{x},\theta^{(q)}]$. Or d'après l'inégalité de Jensen (Dempster *et al.*, 1977) $H(\theta^{(q)}|\theta^{(q)}) \geq H(\theta|\theta^{(q)})$ quel que soit $\theta$ . Ainsi, soit $\theta^{(q+1)}$ qui maximise $Q(\theta|\theta^{(q)})$, cela nous permet d'écrire

$$\begin{aligned} L(\mathbf{x},\theta^{(q+1)}) &= Q(\theta^{(q+1)}|\theta^{(q)}) - H(\theta^{(q+1)}|\theta^{(q)}) \\ &\geq Q(\theta^{(q)}|\theta^{(q)}) - H(\theta^{(q)}|\theta^{(q)}) \\ &= L(\mathbf{x},\theta^{(q)}) \end{aligned} \tag{1.19}$$

L'algorithme EM est composé de deux étapes. Dans la première étape dite Estimation, on calcule la quantité $Q(\theta|\theta^{(q)})$ et dans la seconde dite Maximisation, on maximise cette quantité. On répète ce procédé jusqu'à la convergence qu'on décrira ultérieurement.

**Étape Estimation :** Dans cette étape, l'algorithme EM est appliqué au problème de la classification en traitant les $z_{ik}$ comme données manquantes. A partir de l'expression 1.17, nous avons

$$\begin{aligned} Q(\theta|\theta^{(q)}) &= \mathbf{E}[\sum_{i=1}^{n}\sum_{k=1}^{s} z_{ik}\left(\log(p_k\varphi_k(\mathbf{x_i};\alpha_k))\right)|\mathbf{x},\theta^{(q)}] \\ &= \sum_{i=1}^{n}\sum_{k=1}^{s}\mathbf{E}[z_{ik}|\mathbf{x},\theta^{(q)}]\left(\log(p_k\varphi_k(\mathbf{x_i};\alpha_k))\right) \\ &= \sum_{i=1}^{n}\sum_{k=1}^{s}[p(z_{ik}=1|\mathbf{x},\theta^{(q)}]\left(\log(p_k\varphi_k(\mathbf{x_i};\alpha_k))\right) \\ &= \sum_{i=1}^{n}\sum_{k=1}^{s} t_k(\mathbf{x}_i,\theta^{(q)})\left(\log(p_k) + \log\varphi_k(\mathbf{x}_i,\alpha_k)\right) \end{aligned}$$

où par le théorème de Bayes $t_k(\mathbf{x}_i,\theta^{(q)}) = p(z_{ik} = 1|\mathbf{x},\theta^{(q)})$ et noté $t_{ik}^{(q)}$ s'écrit

$$t_{ik}^{(q)} = \frac{p_k^{(q)}\varphi_k(\mathbf{x}_i,\alpha_k^{(q)})}{\sum_{k'=1}^{s} p_{k'}^{(q)}\varphi_{k'}(\mathbf{x}_i,\alpha_{k'}^{(q)})}$$

Les $t_{ik}^{(q)} \propto p_k^{(q)}\varphi_k(\mathbf{x}_i,\alpha_k^{(q)})$ sont les probabilités conditionnelles a posteriori. Notons que dans le contexte modèle de mélange, cette étape se réduit en fait aux calculs des $t_{ik}^{(q)}$.

**Étape Maximisation :** Dans cette étape on cherche le paramètre $\theta^{(q+1)}$ qui maximise $Q(\theta|\theta^{(q)})$. Ici les estimations $p_k^{(q+1)}$ des proportions $p_k$ sont calculées indépendamment des estimations $\alpha_k^{(q+1)}$. Si les $z_{ik}$ étaient observées, ces proportions seraient estimées simplement par $\sum_{i=1}^{n} \frac{z_{ik}}{n}$ et comme l'étape estimation implique le remplacement de chaque $z_{ik}$ par $t_{ik}^{(q)}$, la mise à jour de ces proportions est donnée simplement par

$$p_k^{(q+1)} = \frac{\sum_{i=1}^{n} t_{ik}^{(q)}}{n}$$

Par ailleurs, les paramètres $\alpha_k^{(q+1)}$ qui dépendent de la paramétrisation du modèle de mélange sont obtenus en résolvant l'équation suivante :

$$\frac{\sum_{i=1}^{n}\sum_{k=1}^{s} t_{ik}^{(q)} \log \varphi_k(\mathbf{x}_i,\alpha_k)}{\partial\theta} = 0$$

Comme ces étapes Estimation et Maximisation ne font pas décroître la vraisemblance, elles sont alternativement répétées jusqu'à ce que la différence

$$L(\mathbf{x},\theta^{(q+1)}) - L(\mathbf{x},\theta^{(q)})$$

soit négligeable. Généralement, le critère d'arrêt est définit par une condition de type

$$\left|\frac{L(\mathbf{x},\theta^{(q+1)}) - L(\mathbf{x},\theta^{(q)})}{L(\mathbf{x},\theta^{(q)})}\right| < \varepsilon$$

avec un seuil arbitrairement choisi, on prend habituellement $\varepsilon = 10^{-6}$.

La convergence de cet algorithme n'est pas démontrée dans un cadre général. En effet, celle proposée par Dempster *et al.* (1977) était fausse. Par contre, Redner et Walker (1984) ont démontré la convergence locale dans le cadre de l'estimation des paramètres d'un mélange fini. Les auteurs ont prouvé que cet algorithme dépend de la position initiale (si celle-ci est très éloignée de la vraie valeur des paramètres, l'algorithme EM risque de converger vers une solution singulière). Afin de pallier cet inconvénient, il est d'usage de lancer plusieurs fois l'algorithme (20 fois par exemple) sur différentes configurations initiales obtenus au hasard ou par une autre méthode.

Avant de clore cette section, et pour illustration, nous considérons ci-dessous, les étapes Estimation et Maximisation correspondant aux mélanges gaussiens :

**Étape Estimation :** Calcul des probabilités $t_{ik}^{(q)}$ en utilisant $\theta^{(q)}$.

**Étape Maximisation :** Calcul de $\theta^{(q+1)}$ qui maximise $Q(\theta|\theta^{(q)})$. Les estimateurs du maximum de vraisemblance s'écrivent alors :

$$p_k^{(q+1)} = \frac{\sum_{i=1}^{n} t_{ik}^{(q)}}{n}$$

$$g_k^{(q+1)} = \frac{\sum_{i=1}^{n} t_{ik}^{(q)} \times \mathbf{x}_i}{\sum_{i=1}^{n} t_{ik}^{(q)}}$$

$$\Sigma_k^{(q+1)} = \frac{\sum_{i=1}^{n} t_{ik}^{(q)} \times (\mathbf{x}_i - g_k^{(q+1)})(\mathbf{x}_i - g_k^{(q+1)})'}{\sum_{i=1}^{n} t_{ik}^{(q)}}$$

Même si l'objectif principal de l'algorithme EM est la recherche des estimateurs de maximum de vraisemblance du modèle de mélange, une partition de l'ensemble des objets peut être obtenue en affectant chaque objet $\mathbf{x}_i$ à la composante du mélange (la classe) la plus probable c'est à dire

$$z_{ik} = \text{argmax } t_k(\mathbf{x}_i, \hat{\theta})$$

où $\hat{\theta}$ est l'estimation au centre du maximum de vraisemblance de $\theta$.

### Lien avec la classification floue (fuzzy $c$-means)

Hathaway (1986) a montré qu'en appliquant l'algorithme EM à un modèle de mélange, on effectue les mêmes calculs que si l'on maximisait de façon alternée le critère de classification floue suivant :

$$F(c,\theta) = \underbrace{\sum_{k=1}^{s}\sum_{i=1}^{n} c_{ik} \log(\pi_k \varphi(x_i,\alpha_k))}_{L_c(\mathbf{c},\theta)} - \underbrace{\sum_{k=1}^{s}\sum_{i=1}^{n} c_{ik} \log(c_{ik})}_{E(\mathbf{c})} \tag{1.20}$$

où $\mathbf{c}$ représente une matrice de classification floue et $\theta$ dénote les paramètres du mélange. La fonction $L_c(\mathbf{z},\theta)$ peut être vue comme l'extension au domaine flou de la vraisemblance complétée. La fonction $E(\mathbf{c})$ s'interprète comme l'entropie de la classification floue $\mathbf{c}$. L'entropie est d'autant plus grande que la classification est floue ($c_{ik}$ très différents de 0 et 1), et réciproquement nulle pour une classification dure (tous les $c_{ik}$ à 0 ou 1). Ainsi, le terme $E(\mathbf{c})$ exprime un degré de confusion des classes. Dans la suite, nous appellerons le critère $L$ la *vraisemblance classifiante floue* associée au modèle de mélange. Cette appellation est due au fait que si la classification est restreinte au domaine des classifications dures, la vraisemblance classifiante floue $L(\mathbf{c},\theta)$ devient exactement égale à la vraisemblance des données complétées $L_c(\mathbf{z},\theta)$.

Pour montrer l'équivalence entre EM et la maximisation de $F(\mathbf{c},\theta)$, Hathaway considère le schéma itératif suivant. On part d'une valeur initiale des paramètres $\theta^{(0)}$. L'itération $q+1$ se décompose en deux étapes :

1. Mise à jour de la classification floue :

$$\mathbf{c}^{(q+1)} = \arg \max L(\mathbf{c},\theta^{(q)})$$

avec comme contraintes

$$\forall\, i,\ \sum_{k=1}^{s} c_{ik} = 1$$
$$\forall\, i,k,\ 0 \leq c_{ik} \leq 1$$
$$\forall\, k,\ 0 < \sum_{i=1}^{n} c_{ik} < n$$

La solution unique est ainsi :

$$\forall i,k\ c_{ik}^{(q+1)} = \frac{\pi_k^{(q)} \varphi(\mathbf{x}_i|\theta_k^{(q)})}{\sum_{h=1}^{s} \pi_h^{(q)} \varphi(\mathbf{x}_i|\theta_h^{(q)})}$$

Cette formule nous montre donc que les degrés d'appartenance sont égaux aux probabilités d'appartenance calculées à l'étape Estimation de EM.

2. Mise à jour des paramètres

$$\theta^{(q+1)} = \text{argmax}\ L(\mathbf{c}^{(q+1)},\theta)$$

Or, puisque l'entropie $E(\mathbf{c}^{(q+1)})$ est indépendante de $\theta$, ceci est équivalent à l'étape de Maximisation de l'algorithme EM.

On peut donc en conclure que EM pour un mélange, effectue les mêmes calculs que l'optimisation alternée de $F(\mathbf{c},\theta)$. Dans le cas d'un mélange gaussien dont les composantes ont la même matrice de variance et des proportions égales, on peut voir que EM et FCM sont équivalents.

### 1.3.2 Approche classification

Nous avons vu que sous l'approche estimation, la recherche des classes est un sous problème de l'estimation des paramètres. A partir du modèle de mélange, on peut traiter ce problème directement sous l'approche classification (Friedman et Rubin, 1967; Scott et Symons, 1971; Diday *et al.*, 1974; Schroeder, 1976). Sous cette approche, le critère considéré n'est pas la vraisemblance de l'échantillon mais la vraisemblance classifiante, soit le produit des vraisemblances sur les classes. La log-vraisemblance dite restreinte dans ce cas s'écrit :

$$L_{cr}(\mathbf{x}_1,\ldots,\mathbf{x}_n;\theta) = \sum_{i=1}^{n}\sum_{k=1}^{s} z_{ik} \log\left(\varphi_k(\mathbf{x}_i;\alpha_k)\right) \tag{1.21}$$

Notons que cette expression ne fait pas apparaître explicitement la notion de proportions entre les différentes sous-populations. La maximisation de ce critère tend en pratique à proposer des partitions où les classes sont de taille comparable. Autrement dit contrairement à l'expression de la log-vraisemblance complétée, ce critère suppose implicitement que toutes les sous-populations sont de même taille. Symons (1981) suggère l'utilisation de la log-vraisemblance complétée (1.15) qui apparaît en fait comme la log-vraisemblance classifiante pénalisée par un terme prenant en compte les proportions des différentes classes

$$L_c(\mathbf{x}_1,\ldots,\mathbf{x}_n;\theta) = L_{cr}(\mathbf{x}_1,\ldots,\mathbf{x}_n;\theta) + \sum_{k=1}^{s} \#z_k \log(p_k). \tag{1.22}$$

La maximisation de ces deux critères peut se faire par l'algorithme classification EM (**CEM**) proposé par (Celeux et Govaert, 1992) et qui est en fait dérivé de EM. On y introduit entre les deux étapes d'estimation et de maximisation, une étape de classification, qui va nous permettre d'affecter chaque objet à une classe. Alors que dans EM, nous travaillons sur une partition floue, ici nous transformons celle-ci en partition dure dans CEM. Une itération de cet algorithme se décompose ainsi :

**E-step** L'étape Estimation calcule les probabilités a posteriori $t_{ik}^{(q)}$ que chaque $\mathbf{x}_i$ appartienne à la $k$ième classe.

**C-step** L'étape Classification affecte chaque objet à une classe, selon la méthode du MAP :

$$z_k = \{i | t_{ik} = max_{h=1,\ldots,s}\ t_{ih}\} \tag{1.23}$$

**M-step** L'étape Maximisation consiste à calculer $\theta^{(q+1)} = (\mathbf{p}^{(q+1)},\alpha^{(q+1)})$ maximisant l'espérance conditionnelle à la connaissance des données complétées et $\theta^{(q)}$. Pour les proportions, cela revient à

$$p_k = \frac{\sharp z_k}{n}$$

De même que pour EM, ces étapes sont répétées jusqu'à l'obtention de la convergence. Et l'initialisation peut se faire de la même manière. Et de même que EM, il produit une solution d'optimum local ; il faut donc le lancer plusieurs fois à partir de différentes initialisations.

### Lien avec $k$-means

Dans le cas de données continues, en utilisant le modèle Gaussien, la log-vraisemblance classifiante $L_c(\mathbf{z};\theta)$ prend la forme suivante :

$$L_c(\mathbf{z};\theta) = -\frac{1}{2}\sum_{k=1}^{s}\sum_{i\in z_k}(\mathbf{x}_i - \mathbf{g}_k)'\Sigma_k^{-1}(\mathbf{x}_i - \mathbf{g}_k) - \frac{nd}{2}\log(2\pi) - \frac{1}{2}\sum_{k=1}^{s}\#z_k \log(|\Sigma_k|).$$

La maximisation de cette expression est équivalente à la minimisation de :

$$\sum_{k=1}^{s} trace(W_k\Sigma_k^{-1}) + \sum_{k=1}^{s}\#z_k\log(|\Sigma_k|) - 2\sum_{k=1}^{s}\#z_k\log(p_k)$$

où $W_k = \sum_{i\in z_k}(\mathbf{x}_i - \mathbf{g}_k)(\mathbf{x}_i - \mathbf{g}_k)'$. Si on considère le modèle le plus simple c'est à dire les formes des classes sont sphériques et les proportions et les volumes identiques, la maximisation de $L_c(\mathbf{z};\theta)$ se réduit à la minimisation de

$$trace\left(\sum_{k=1}^{s}(\mathbf{x}_i - \mathbf{g}_k)(\mathbf{x}_i - \mathbf{g}_k)'\right) = \sum_{k=1}^{s}\sum_{i\in z_k} d^2(\mathbf{x}_i,\mathbf{g}_k)$$

Ainsi, l'algorithme CEM est dans ce cas simplement l'algorithme des centres mobiles.

Comme nous venons de le voir, l'approche mélange permet de placer les méthodes existantes dans un cadre précis, aussi comme nous le verrons dans le prochain chapitre, elle permet d'offrir des outils de choix de modèle voire du choix du nombre de classes qui a été supposé connu jusqu'à présent. Dans le paragraphe suivant, nous allons nous intéresser justement à ce problème en combinant les avantages de l'approche mélange et l'approche hiérarchique.

### 1.3.3 Approche hiérarchique

Le majeur problème des algorithmes de classification simple est la connaissance requise du nombre de classes dans la population. Pour s'affranchir de cette hypothèse, nous proposons d'utiliser un algorithme de type classification ascendante hiérarchique. Comme nous l'avons vu auparavant, celui-ci requiert la définition d'une distance entre classes, ou tout du moins, d'une méthode pour définir la proximité ou non de deux classes. Puisque nous sommes dans le cadre des modèles de mélange, nous pouvons définir une distance entre deux classes à partir de $L_c(\mathbf{z},\theta)$ qui peut s'écrire :

$$L_c(\mathbf{z},\theta) = \sum_{k=1}^{s} L_c(z_k,\theta).$$

De cette manière, nous avons la distance suivante :

$$d(z_k,z_{k'}) = L_c(z_k,\theta) + L_c(z_{k'},\theta) - L_c(z_k \cup z_{k'},\theta) \tag{1.24}$$

Cette distance correspond à l'évolution de la log-vraisemblance, lors de la fusion de deux classes. Soit $k$ et $k'$ les deux classes fusionnées, on a ainsi l'égalité suivante :

$$L_c(\mathbf{z}',\theta^{(q+1)}) = L_c(\mathbf{z},\theta^{(q)}) - d(z_k,z_{k'}) \tag{1.25}$$

où $\mathbf{z}'$ représente la nouvelle partition obtenue après regroupement de $z_k$ et $z_{k'}$. Il faut donc que cette distance soit minimale, afin de maximiser le critère. Nous fusionnons donc les deux classes pour laquelle $d(z_k,z_{k'})$ est la plus petite possible. Et nous utilisons l'algorithme **CAH**, précédemment décrit.

#### Cas des données continues

Il est possible d'écrire la log-vraisemblance classifiante pour le modèle Gaussien de cette manière suivante :

$$\begin{aligned} L_c(\mathbf{z},\theta) &= -\frac{1}{2}\sum_{k=1}^{s}\sum_{i\in z_k}(\mathbf{x}_i - g_k)\Sigma_k^{-1}(x_i - g_k) \\ &- \frac{nd}{2}\log(2\pi) - \frac{1}{2}\sum_{k=1}^{s}\#z_k \log|\Sigma_k| \end{aligned}$$

Dans ce qui suit, et dans un souci de clareté, on ne considérera que le modèle simple *Sphérique* où la matrice de variance $\sum_k = \sigma^2 I$ ($\sigma$ inconnu), la log-vraisemblance classifiante devient

$$L_c(\mathbf{z},\theta) = -\frac{1}{2\sigma^2}\sum_{k=1}^{s}\sum_{i\in z_k}\mathbf{d}^2(\mathbf{x}_i,g_k) - \frac{nd}{2}\log(\sigma^2 2\pi) \tag{1.26}$$

Cette log-vraisemblance classifiante croît avec le nombre de classes, on peut définir un critère d'agrégation à l'aide de $L_c(\mathbf{z},\theta) - L_c(\mathbf{z}',\theta)$ sachant que $\mathbf{z}'$ résulte de la fusion des classes $z_1$ et $z_2$ cette différence se réduit en fait à:

$$\begin{aligned} L_c(\mathbf{z},\theta) - L_c(\mathbf{z}',\theta) &= L_c(z_1,\theta_1) + L_c(z_2,\theta_2) - L_c(z_{1,2},\theta_{1,2}) \\ &= \frac{n}{2\sigma^2}\left(\sum_{i\in z_{1,2}}\frac{1}{n}\mathbf{d}^2(\mathbf{x}_i,g_{1,2}) - \sum_{i\in z_1}\frac{1}{n}\mathbf{d}^2(\mathbf{x}_i,g_1) - \sum_{i\in z_2}\frac{1}{n}\mathbf{d}^2(\mathbf{x}_i,g_2)\right) \\ &= \frac{n}{2\sigma^2}\delta_{ward}(z_1,z_2) \end{aligned}$$

Notons que cette expression est toujours positive et on peut définir par conséquent une mesure de dissimilarité entre deux classes par:

$$\Delta(z_1,z_2) = \frac{n}{2\sigma^2}\delta_{ward}(z_1,z_2) \tag{1.27}$$

On en déduit que

$$L_c(\theta;\mathbf{z}') = L_c(\theta;\mathbf{z}) - \Delta(z_1,z_2) \tag{1.28}$$

Notons que lorsque $\sigma^2$ est fixé, le critère entre deux classes se réduit à une mesure de dissimilarité entre les classes correspondant au critère de Ward.

$$\Delta_1(z_1,z_2) = \delta_{ward}(z_1,z_2)$$

Le calcul des distances entre les classes est essentiel pour la mise en place pratique de la CAH. Sans une formule de récurrence ce calcul serait prohibitif en temps de calcul. Celle-ci dû à Lance et Williams (1967) s'écrit, par exemple pour $\Delta(z_1 \cup z_2, z_3)$ comme suit:

$$\frac{n}{2\sigma^2}\frac{(\#z_1+\#z_3)\delta_{ward}(z_1,z_3) + (\#z_2+\#z_3)\delta_{ward}(z_2,z_3) - \#z_3\delta_{ward}(z_1,z_2)}{\#z_1+\#z_2+\#z_3}$$

### 1.3.4 Classification mixte

Contrairement aux méthodes hiérarchiques, les méthodes de partitionnement requièrent la connaissance du nombre de classes. En combinant les avantages des deux types de méthodes. Wong (1977) a proposé un procédé simple et efficace bien adapté lorsque les données sont de grande taille en combinant $k$-means et la CAH. Cette démarche est décrite dans la figure 1.3 et peut être étendu sans difficulté en restant dans un cadre modèle de mélange.

- on cherche une partition à l'aide de EM ou CEM (à la place de $k$-means) en prenant un nombre très grand de classes,

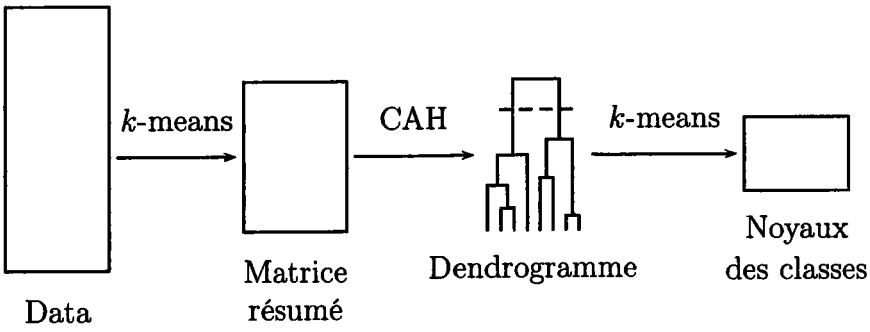



FIG. 1.3 – *La méthode de Wong (notez que k-means peut être remplacé par* **EM** *ou* **CEM**, *soit les deux fois, soit une des deux)*

- à partir des centres des classes, on applique la CAH, en utilisant un critère déduit de la log-vraisemblance classifiante,
- ensuite, on propose un nombre de classes approprié au vu du dendrogramme ou basé sur la décroissance de l'inertie,
- soit on relance EM ou CEM avec ce nombre de classes sur le tableau initial, soit simplement ajouter une étape de réaffectation aux centres les plus proches pour améliorer la partition.

Comme nous le verrons plus tard, ce procédé peut être appliqué à tous types de données (qualitative, binaire, ...).

# Chapitre 2

# Application aux données qualitatives

Dans ce chapitre, nous nous intéressons aux données qualitatives. Malgré l'intérêt grandissant pour ce type de données dans différents domaines tels que l'analyse textuelle et les questionnaires, peu de travaux ont été réalisés comparativement à ceux appliqués sur les données continues. Comme nous considérons l'approche modèle de mélange, nous allons donc rappeler ici la définition des modèles des classes latentes habituellement utilisés pour les données qualitatives. Nous étudierons ensuite différentes variantes de ce modèle et nous verrons comment l'un d'entre eux est associé au critère métrique énoncé déjà dans le chapitre 1 :

$$E = \sum_{k=1}^{s} \sum_{i \in z_k} \mathbf{d}(\mathbf{x}_i, m_k). \tag{2.1}$$

où $m_k$ est le centre de la $k$ème classe $z_k$ et $\mathbf{d}$ est une mesure de dissimilarité définie par :

$$\mathbf{d}(\mathbf{x}_i, m_k) = \sum_{j=1}^{q} \delta(x_i^j, m_k^j) \text{ où } \delta(x,y) = \begin{cases} 0 & \text{si } x = y \\ 1 & \text{sinon} \end{cases} \tag{2.2}$$

A partir des données simulées, nous comparerons l'algorithme CEM et l'algorithme $k$-modes associé à ce critère décrit également dans le chapitre 1. Nous montrerons ainsi l'intérêt de l'approche probabiliste. L'approche estimation est également étudiée pour ce modèle. Enfin, Une étude comparative entre les approches estimation et classification sera réalisée à l'aide de données simulées et réelles.

## 2.1 Modèles

### 2.1.1 Classes latentes

Comme nous nous intéressons aux données de type qualitatif (couleur des cheveux, marque de voiture, ...), nous présentons tout d'abord le modèle général, le modèle des classes latentes (Lazarfeld et Henry, 1968; Everitt et Hand, 1981; Everitt, 1984), noté $\lfloor p_k, \alpha_k^{je} \rfloor$. Celui-ci dépendra des proportions des classes et des probabilités associées aux modalités dans chaque classe.

Le principe de base de ce modèle est la supposition de l'existence d'une variable qualitative latente à $s$ modalités ($s$ étant le nombre de classes) telle qu'à la

connaissance de l'une d'entre elles, les variables sont supposées indépendantes. Pour sa simplicité et sa performance, cette hypothèse, même si elle paraît trop forte, est souvent retenue lorsque les données sont de type qualitatif ou binaire, comme par exemple dans le programme Autoclass (Cheeseman et Stutz, 1996). Dans ce cas, la densité de probabilité d'un objet $\mathbf{x}_i$ peut s'écrire :

$$\varphi(\mathbf{x}_i,\theta) = \sum_{k=1}^{s} p_k \varphi_k(\mathbf{x}_i;\alpha_k) = \sum_{k=1}^{s} p_k \prod_{j=1}^{d} \prod_{e=1}^{c_j} (a_k^{je})^{x_i^{je}}$$

où chaque variable $j = 1,\ldots,d$ a un nombre fini de modalités $c_j$, et $x_i^{je} = 1$ si l'objet $i$ a la modalité $e$ pour la variable $j$, et 0 sinon. Les valeurs $a_k^{je}$ donnent la probabilité de la modalité $e$ de la variable $j$ pour la classe $z_k$. Le paramètre $\theta$ indique l'ensemble de tous les paramètres inconnus du modèle. Dans ce cas, la log-vraisemblance et la log-vraisemblance des données complétées s'écrivent respectivement :

$$L(\mathbf{x}_1,\ldots,\mathbf{x}_n;\theta) = \sum_{i=1}^{n} \log \left( \sum_{k=1}^{s} p_k \prod_{j=1}^{d} \prod_{e=1}^{c_j} (a_k^{je})^{x_i^{je}} \right)$$

et

$$L_c(\mathbf{x}_1,\ldots,\mathbf{x}_n;\theta) = \sum_{i=1}^{n} \sum_{k=1}^{s} z_{ik} \left( \log p_k + \sum_{j=1}^{d} \sum_{e=1}^{c_j} x_i^{je} \log a_k^{je} \right)$$

Ci-après, nous allons expliciter les différentes de l'algorithme EM :

- **Étape Estimation** : Calcul des probabilités $t_{ik}$ en utilisant $\theta^{(q)}$.
- **Étape Maximisation** : Calcul de $\theta^{(q+1)}$ qui maximise $Q(\theta|\theta^{(q)})$. Les estimateurs du maximum de vraisemblance s'écrivent alors :

$$p_k^{(q+1)} = \frac{\sum_{i=1}^{n} t_{ik}^{(q)}}{n}$$

$$(a_k^{je})^{(q+1)} = \frac{\sum_{i=1}^{n} t_{ik}^{(q)} \times x_i^{je}}{\sum_{i=1}^{n} t_{ik}^{(q)}}$$

Rappelons qu'une partition de l'ensemble des objets peut être obtenue en affectant chaque objet $\mathbf{x}_i$ à la composante du mélange (la classe) la plus probable c'est à dire

$$z_{ik} = \text{argmax}_{h=1,\ldots,s}\ t_h(\mathbf{x}_i,\hat{\theta})$$

où $\hat{\theta}$ est l'estimation au centre du maximum de vraisemblance de $\theta$. Notons que si l'on considère l'approche classification, la probabilité $(a_k^{je})$ à l'étape $(q+1)$ est définie par

$$\frac{\sum_{i=1}^{n} z_{ik}^{(q)} \times x_i^{je}}{\#z_k}$$

qui correspond à la proportion d'objets dans la classe $z_k$ ayant choisi la modalité $e$ de la variable $j$. Il s'ensuit que lorsque les proportions sont supposées égales, le critère de log-vraisemblance classifiante peut s'écrire sous la forme suivante :

$$\sum_{k=1}^{s}\sum_{j=1}^{d}\sum_{e=1}^{c_j} x_k^{je}\log x_k^{je} - d\sum_{k=1}^{s}\#z_k\log\#z_k$$

Comme nous l'avons vu dans le premier chapitre, la maximisation de ce terme est équivalent à la maximisation de l'information mutuelle (1.6). Donc, rechercher une partition $\mathbf{z}$ en $s$ classes maximisant l'information mutuelle à partir d'un tableau disjonctif complet revient à identifier un modèle de $s$ classes latentes.

Le nombre de paramètres à estimer que ce modèle nécessite est égal à $s(\sum_{j=1}^{d} c_j - d + 1) - 1$. Une condition nécessaire et triviale pour que le modèle soit identifiable est que le nombre de valeurs posssibles soit plus grand que le nombre de paramètres. Une bonne qualité d'estimation nécessitera une taille raisonnable des données. Pour ces raisons, nous présentons ci-dessous des modèles restreints plus parcimonieux que le modèle des classes latentes.

### 2.1.2 Modèles restreints

A partir du modèle précédent, qui est le plus général, on peut en déduire plusieurs autres, pour lesquels nous pouvons imposer des contraintes (proportions égales, degrés d'homogénéité égaux pour chaque classe, pour chaque variable, etc.). Dans un premier temps, nous allons considérons le modèle $\lfloor p_k,\varepsilon_k^j \rfloor$ (Nadif et Marchetti, 1993). Dans ce modèle, nous supposons l'indépendance des $d$ variables pour chaque classe $z_k$ du mélange, et que chacune d'entre elles est distribuée selon une densité de probabilité définie par :

$$\varphi(\mathbf{x}_i,\alpha_k) = \prod_{j=1}^{d}(1-\varepsilon_k^j)^{1-\delta(x_i^j,m_k^j)}\left(\frac{\varepsilon_k^j}{c_j-1}\right)^{\delta(x_i^j,m_k^j)} \tag{2.3}$$

$$\text{et } \delta(x,y) = \begin{cases} 0 & \text{si } x = y \\ 1 & \text{sinon} \end{cases}$$

Ainsi, le paramètre $\alpha_k = (m_k,\varepsilon_k)$ avec $m_k$ est le mode de la $k$ième classe (modalités les plus choisies pour chaque variable) et $\varepsilon_k$ est un vecteur de probabilités de dimension $d$. Cette distribution indique que pour la classe $k$, la variable $j$ prend la modalité $m_k^j$ avec la plus grande probabilité $1-\varepsilon_k^j$ et prend les autres modalités avec la même probabilité $\frac{\varepsilon_k^j}{c_j-1}$. Le nombre de paramètres à estimer est dans ce cas $s(2d+1)$ qui est bien entendu inférieur à celui du modèle des classes latentes. Dans l'approche Estimation, avec l'algorithme EM, les paramètres sont définies de la manière suivante :

$$m_k^j = \arg\min_{e=1,\ldots,c_j}\sum_{i=1}^{n} t_{ik}\delta(x_i^j,e)\forall k,j$$

et

$$\varepsilon_k^j = \frac{\sum_{i=1}^{n} t_{ik}\delta(x_i^j, m_k^j)}{\sum_{i=1}^{n} t_{ik}}$$

Quant aux proportions, elles sont toujours définies de la même manière quelque soit le modèle. Dans le cadre de l'approche Classification, avec CEM, nous utilisons la formule suivante pour calculer les valeurs majoritaires (ainsi que pour tous les autres modèles) :

$$m_k^j = \arg\min_{e=1,\ldots,c_j} \sum_{i \in z_k} \delta(x_i^j, e) \forall k, j$$

et

$$\varepsilon_k^j = \frac{\sum_{i \in z_k} \delta(x_i^j, m_k^j)}{\sharp z_k}$$

Pour illustrer l'algorithme EM à partir de ce modèle, nous proposons un exemple. Soit x une table de données de 10 objets $\{i1,i2,i3,i4,i5,i6,i7,i8,i9,\ i10\}$ décrits par 5 variables $\{v1,v2,v3,v4,v5\}$ qualitatives nominales. La meilleure partition obtenue est constituée de :

- $z_1 = \{i1,i5,i6,i8\}$
- $z_2 = \{i9,i10\}$
- $z_3 = \{i2,i3,i4,i7\}$

TAB. 2.1 – *Données* $10 \times 5$, *initiales et réordonnées selon la partition obtenue avec EM.*

| objets | v1 | v2 | v3 | v4 | v5 |
|---|---|---|---|---|---|
| i1 | 1 | 2 | 2 | 3 | 2 |
| i2 | 1 | 1 | 1 | 1 | 1 |
| i3 | 1 | 1 | 1 | 2 | 1 |
| i4 | 2 | 1 | 1 | 1 | 1 |
| i5 | 2 | 2 | 1 | 3 | 3 |
| i6 | 1 | 2 | 1 | 3 | 2 |
| i7 | 1 | 1 | 1 | 1 | 3 |
| i8 | 1 | 2 | 1 | 3 | 2 |
| i9 | 3 | 3 | 3 | 1 | 1 |
| i10 | 3 | 3 | 3 | 2 | 1 |

| objets | v1 | v2 | v3 | v4 | v5 |
|---|---|---|---|---|---|
| i1 | 1 | 2 | 2 | 3 | 2 |
| i5 | 2 | 2 | 1 | 3 | 3 |
| i6 | 1 | 2 | 1 | 3 | 2 |
| i8 | 1 | 2 | 1 | 3 | 2 |
| i9 | 3 | 3 | 3 | 1 | 1 |
| i10 | 3 | 3 | 3 | 2 | 1 |
| i2 | 1 | 1 | 1 | 1 | 1 |
| i3 | 1 | 1 | 1 | 2 | 1 |
| i4 | 2 | 1 | 1 | 1 | 1 |
| i7 | 1 | 1 | 1 | 1 | 3 |

Dans le tableau 2.1, nous voyons les données initiales et réordonnées selon la meilleure partition avec EM. De plus, le tableau 2.2 présente les modes des classes. Nous voyons avec ce petit exemple que l'interprétation des classes est très aisée.

Comme pour les mélanges gaussiens, nous pouvons proposer différentes variantes de ce modèle en imposant des contraintes sur les proportions et les paramètres $\varepsilon_k^j$. Par exemple, lorsqu'on suppose que les proportions sont égales, nous définissons le

TAB. 2.2 – *Les centres des classes et les valeurs de* $\varepsilon_k^j$

| m | v1 | v2 | v3 | v4 | v5 |
|---|---|---|---|---|---|
| $m_1$ | 1 | 2 | 1 | 3 | 2 |
| $m_2$ | 3 | 3 | 3 | 1 | 1 |
| $m_3$ | 1 | 1 | 1 | 1 | 1 |

| $\varepsilon_k^j$ | v1 | v2 | v3 | v4 | v5 |
|---|---|---|---|---|---|
| $\varepsilon_1$ | 0.25 | 0.00 | 0.25 | 0.00 | 0.25 |
| $\varepsilon_2$ | 0.00 | 0.00 | 0.00 | 0.50 | 0.00 |
| $\varepsilon_3$ | 0.25 | 0.00 | 0.00 | 0.25 | 0.25 |

TAB. 2.3 – *Les autres variantes du modèle* $\lfloor p_k,\varepsilon_k^j \rfloor$

| Modèles | Signification |
|---|---|
| $\lfloor p_k,\varepsilon \rfloor$ | les $\varepsilon_k^j$ sont supposées égales pour toutes les classes et toutes les variables |
| $\lfloor p,\varepsilon \rfloor$ | c'est le modèle $\lfloor p_k,\varepsilon \rfloor$lorsque les proportions sont supposées égales |
| $\lfloor p_k,\varepsilon^j \rfloor$ | les $\varepsilon_k^j$ dépendent seulement des variables |
| $\lfloor p,\varepsilon^j \rfloor$ | c'est le modèle $\lfloor p_k,\varepsilon^j \rfloor$lorsque les proportions sont supposées égales |
| $\lfloor p_k,\varepsilon_k \rfloor$ | les $\varepsilon_k^j$ dépendent seulement des classes |
| $\lfloor p,\varepsilon_k \rfloor$ | c'est le modèle $\lfloor p_k,\varepsilon_k \rfloor$lorsque les proportions sont supposées égales |

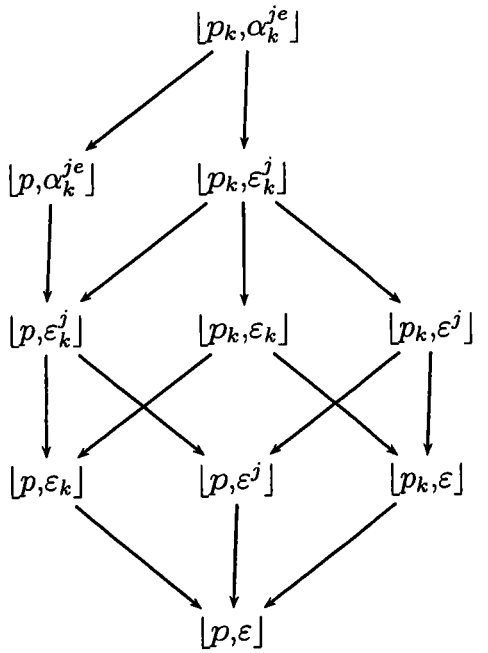



FIG. 2.1 – *Les différents modèles restreints possibles, à partir du modèle des classes latentes le plus complexe,* $\lfloor p_k,\alpha_k^{je} \rfloor$*, au plus simple,* $\lfloor p,\varepsilon \rfloor$.

modèle $\lfloor p,\varepsilon_k^j \rfloor$. Les autres modèles sont définies dans le tableau 2.3 et schématisés dans la figure 2.1.

Nous présentons dans le tableau 2.4, pour chacun des modèles et pour chaque approche (Estimation et Classification) les expressions des paramètres $\alpha$ dans le cas du modèles des classes latentes, et des paramètres $\varepsilon$ pour les autres modèles (le calcul des paramètres $m_k$ étant commun et décrit plus haut).

Le choix du modèle est primordial lorsque la taille des données n'est pas très grande (voir les expériences numériques qui ont été réalisées sur des données simulées par Nadif et Marchetti (1993) et Nadif et Govaert (1993)).

TAB. 2.4 – *Calcul des paramètres pour le modèle des classes latentes, et les modèles dérivés.*

| Modèles | EM | CEM |
|---|---|---|
| $\lfloor p_k,\alpha_k^{je}\rfloor$, $\lfloor p,\alpha_k^{je}\rfloor$ | $a_k^{je} = \frac{\sum_{i=1}^{n} t_{ik} x_i^{je}}{\sum_{i=1}^{n} t_{ik}}$ | $a_k^{je} = \frac{\sum_{i\in z_k} x_i^{je}}{\sharp z_k}$ |
| $\lfloor p_k,\varepsilon_k^{j}\rfloor$, $\lfloor p,\varepsilon_k^{j}\rfloor$ | $\varepsilon_k^{j} = \frac{\sum_{i=1}^{n} t_{ik}\delta(x_i^j,m_k^j)}{\sum_{i=1}^{n} t_{ik}}$ | $\varepsilon_k^{j} = \frac{\sum_{i\in z_k} \delta(x_i^j,m_k^j)}{\sharp z_k}$ |
| $\lfloor p_k,\varepsilon_k\rfloor$, $\lfloor p,\varepsilon_k\rfloor$ | $\varepsilon_k = \frac{\sum_{i=1}^{n} t_{ik} \sum_{j=1}^{d} \delta(x_i^j,m_k^j)}{d\sum_{i=1}^{n} t_{ik}}$ | $\varepsilon_k = \frac{\sum_{i\in z_k} \sum_{j=1}^{d} \delta(x_i^j,m_k^j)}{d\,\sharp z_k}$ |
| $\lfloor p_k,\varepsilon^{j}\rfloor$, $\lfloor p,\varepsilon^{j}\rfloor$ | $\varepsilon^{j} = \frac{\sum_{i=1}^{n}\sum_{k=1}^{s} t_{ik}\delta(x_i^j,m_k^j)}{n}$ | $\varepsilon^{j} = \frac{\sum_{k=1}^{s}\sum_{i\in z_k} \delta(x_i^j,m_k^j)}{n}$ |
| $\lfloor p_k,\varepsilon\rfloor$, $\lfloor p,\varepsilon\rfloor$ | $\varepsilon = \frac{\sum_{i=1}^{n}\sum_{k=1}^{s} t_{ik} \sum_{j=1}^{d} \delta(x_i^j,m_k^j)}{nd}$ | $\varepsilon = \frac{\sum_{k=1}^{s}\sum_{i\in z_k}\sum_{j=1}^{d} \delta(x_i^j,m_k^j)}{nd}$ |

## 2.2 Les algorithmes CEM et $k$-modes

### 2.2.1 Lien entre CEM et $k$-modes

Nous établissons ici le lien qui existe entre CEM et $k$-modes (Huang, 1997), déjà décrit dans le premier chapitre. Nous montrerons que $k$-modes est juste une version de CEM, avec le modèle le plus simple, $\lfloor p,\varepsilon\rfloor$. En effet, en partant de ce modèle, la log-vraisemblance classifiante s'écrit :

$$L_c(\mathbf{z},\theta) = \sum_{k=1}^{s}\sum_{i\in z_k}\sum_{j=1}^{d} \log\left(\frac{\varepsilon}{(1-\varepsilon)(c_j-1)}\right)\delta(x_i^j,m_k^j).$$

Il est évident que la maximisation cette log-vraisemblance classifiante est équivalent à la minimisation de :

$$W_2 = \sum_{k=1}^{s} \sum_{i \in z_k} \mathbf{d}_\varepsilon(\mathbf{x}_i, m_k).$$

où

$$\mathbf{d}_\varepsilon(\mathbf{x}_i, m_k) = \sum_{j=1}^{d} \log \frac{(1-\varepsilon)(c_j - 1)}{\varepsilon} \delta(\mathbf{x}_i^j, m_k^j)$$

$W_2$ dépend seulement de $\mathbf{d}_\varepsilon(\mathbf{x}_i, m_k)$ qui est une mesure de dissimilarité pondérée. Maintenant, quand toutes les variables ont le même nombre de modalités $c$, ce critère peut s'écrire :

$$W_3 = \log \left( \frac{(1-\varepsilon)(c-1)}{\varepsilon} \right) \sum_{k=1}^{s} \sum_{i \in z_k} \mathbf{d}(\mathbf{x}_i, m_k),$$

où $\mathbf{d}(\mathbf{x}_i, m_k) = \sum_{j=1}^{d} \delta(x_i^j, m_k^j)$. Ainsi, minimiser le critère $W_3$ revient à minimiser le critère de $k$-modes 2.1, et notre modèle probabiliste nous permet d'interpréter le critère $E$. Celui-ci est associé au modèle le plus simple $\lfloor p, \varepsilon \rfloor$ avec les hypothèses suivantes :

- toutes les variables ont le même nombre de modalités,
- les proportions des classes sont supposées égales,
- le paramètre $\varepsilon$ est un réel identique pour chaque classe et chaque variable.

Il apparaît clairement que toutes ces hypothèses implicites dans la méthode $k$-modes sont trop fortes. Nous verrons justement comment elles peuvent pénaliser la structure classificatoire.

Comme nous l'avons mentionné, ce critère a été proposé par Nadif et Marchetti (1993) et Huang (1997), mais les deux algorithmes utilisés sont sensiblement différents. Nous allons étudié les différences entre ces méthodes et CEM. Pour minimiser le critère $W_3$, l'algorithme CEM se réduit en fait à une version qu'on appellera $k^*$-modes. Avec cette version, les proportions n'affectent pas les étapes **Estimation** et **Classification**. Il est évident que ces deux étapes peuvent se réduire à une seule étape d'**Affectation**, où chaque objet est alloué à la classe la plus proche, selon la dissimilarité $\mathbf{d}$ entre un objet et un mode d'une classe.

Contrairement à $k$-modes, $k^*$-modes est initialisé par une partition tirée au hasard, et est basé sur le processus de mise à jour des modes après que tous les objets soient assignés à une classe. Ce processus a été introduit avec l'algorithme $k$-means de Forgy (1965) et étendu aux méthodes des nuées dynamiques (Diday, 1980; Celeux *et al.*, 1989).

### 2.2.2 Simulation

A travers des données simulées selon le modèle $\lfloor p_k, \varepsilon \rfloor$, nous avons noté qu'à partir de la position initiale au hasard, $k$-modes et $k^*$-modes sont équivalents. Pour illustrer les effets des paramètres du modèle (et donc, la nécessité de recourir à un modèle complexe), nous avons effectué quelques simulations pour comparer $k^*$-modes et CEM (Jollois et Nadif, 2002). Le processus de simulation est le suivant.

Nous générons 30 échantillons, avec 2 classes, avec les mêmes paramètres (i. e. $n = 1000$, $d = 10$, $m_1$, $m_2$, $p_1 = 1 - p_2$, $\varepsilon_1$ and $\varepsilon_2$). Pour chaque tableau simulé, nous lançons 20 fois les deux algorithmes, à partir de positions initiales au hasard, et nous choisissons la meilleure solution sur ces 20 essais. Nous évaluons les performances de ces algorithmes avec les proportions d'objets mal-classés en comparant la partition obtenue avec celle simulée.

### Effet des proportions

Ici, nous avons choisi de prendre dans le modèle $\lfloor p_k, \varepsilon \rfloor$ avec $\varepsilon = 0.2$. Puisque le degré de séparation des deux classes dépend de $\Delta = \mathbf{d}(m_1, m_2) = \sum_{j=1}^{q} \delta(m_1^j, m_2^j)$ (Govaert et Nadif, 1996), nous prenons :

- $m_1 = (1, 1, 1, 1, 1, 1, 1, 1, 1, 1)$,
- $m_2 = (1, 1, 1, 2, 2, 2, 2, 2, 2, 2)$ pour des classes peu mélangées ($\Delta = 7$),
- $m_2 = (1, 1, 1, 1, 1, 1, 1, 2, 2, 2)$ pour des classes très mélangées (avec $\Delta = 3$).

De plus, nous prenons pour chaque variable, $c_j = 4$ (donc 4 modalités). Les résultats sont présentés dans la figure 2.2.

Il à noter est que, contrairement à CEM, $k^*$-modes est incapable de fournir une partition en 2 classes lorsque $p_1$ est proche de 0.25, où les classes sont très mélangées. Pour chaque essai, il met tous les objets dans une seule et même classe, privilégiant ainsi la plus grosse. Ceci est donc une des limitations majeures de cet algorithme. Par contre, CEM est assez constant et performant, même quand $\Delta = 3$.

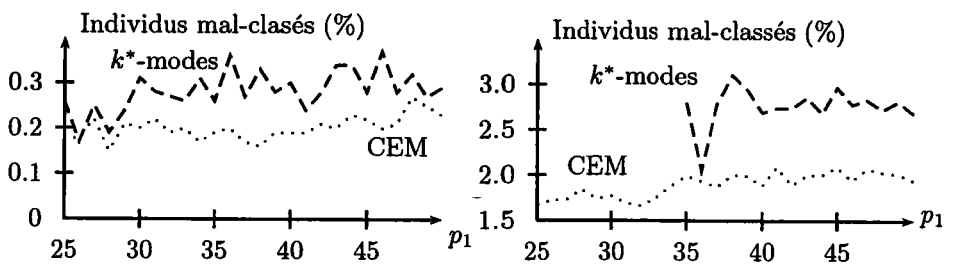



FIG. 2.2 – *$k^*$-modes contre CEM : effet des proportions sur la qualité de la partition obtenue ($\Delta = 7$ à gauche, $\Delta = 3$ à droite). On fait varier la proportion $p_1$ afin d'obtenir une partition déséquilibrée.*

### Effet des paramètres $\varepsilon_k$

Afin d'étudier l'effet des valeurs $\varepsilon_k$, nous prenons des proportions égales (i. e. $p_1 = p_2 = 0.5$), nous fixons $\varepsilon_1^j = 0.1$ pour $j = 1, \ldots, d$ et nous varions pour la deuxième classe $\varepsilon_2^j$ (tous égaux pour $j = 1, \ldots, d$). Ces valeurs vont de 0.10 à 0.40, par un pas de 0.01. Avec le même schéma de simulation que précédemment, les résultats sont présentés dans la figure 2.3.

Pour chaque situation, CEM est nettement meilleur que $k^*$-modes. Quand les données simulées sont proches du modèles $\lfloor p, \varepsilon \rfloor$(c'est-à-dire que les valeurs de $\varepsilon_2$ sont proches de 0.10), CEM et $k^*$-modes ont le même comportement, et donnent les mêmes résultats. Mais, quand les données sont loin de ce modèle (le plus simple est celui correspondant à $k^*$-modes), CEM fonctionne beaucoup mieux que $k^*$-modes.

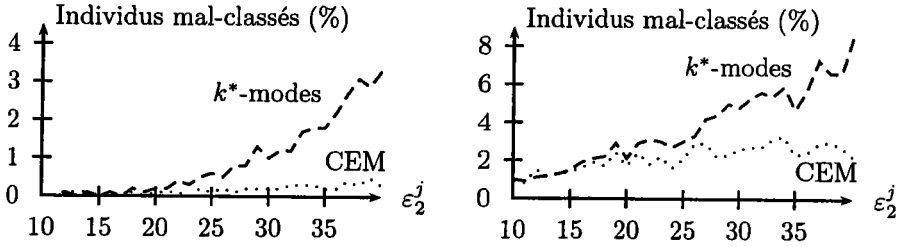



FIG. 2.3 – *$k^*$-modes contre CEM : effet des paramètres $\varepsilon_k^j$ sur la qualité de la partition obtenue ($\Delta = 7$ à gauche, $\Delta = 3$ à droite). On fait varier les paramètres d'une classe afin d'obtenir des classes plus ou moins mélangées.*

### Linéarité de l'algorithme CEM

Afin de montrer la linéarité de la complexité de CEM, nous avons lancé celui-ci sur des données simulées, dont la taille variait de $n = 50000$ à $n = 500000$, avec $d = 50$, $s = 5$ et des classes bien séparées. Nous prenons le temps moyen d'exécution de 5 essais de CEM pour chaque $n$. A partir des résultats obtenus (voir Fig. 2.4), nous voyons que le temps d'exécution de CEM croît linéairement quand la taille des données augmente. Ainsi, cet algorithme est adapté aux données de grandes tailles.

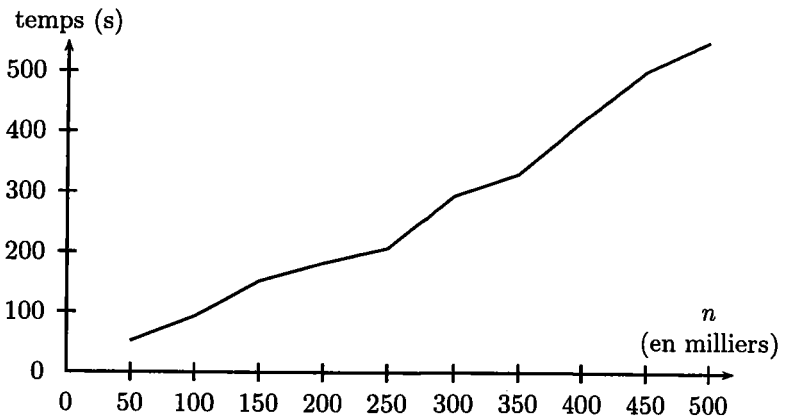



FIG. 2.4 – *Linéarité du temps d'exécution de l'algorithme CEM, quand n augmente.*

## 2.3 Comparaison EM - CEM

Pour comparer ces deux algorithmes, il nous faut bien sûr disposer de données sur lesquelles une classification est possible. De plus, pour valider les résultats, il est nécessaire de connaître ces données. C'est pourquoi nous avons eu recours à des données simulées.

Nous avons simulés trois tableaux de 300 objets et de 15 variables, tous composés de deux classes de tailles différentes (50 et 250). Les deux classes ont été construites à partir de noyaux $m_k$ pour $k = 1$ et 2, et d'une probabilité d'erreur $\varepsilon$, identiques pour toutes les variables et pour toutes les classes. Pour le premier tableau ES1, nous avons des classes assez bien séparées, avec $\varepsilon = 0{,}15$. Le deuxième, ES2, est composé de classes moyennement séparées, avec $\varepsilon = 0{,}30$. Enfin, pour le dernier

tableau ES3, les classes sont peu séparées, avec $\varepsilon = 0{,}45$. Les résultats obtenus sont présentés dans les tableaux 2.5, 2.6 et 2.7.

TAB. 2.5 – *Paramètres $(p_k, \varepsilon_k^j)$ obtenus par les algorithmes EM et CEM pour les données peu mélangées (ES1 : $\varepsilon = 0.15$).*

| | EM | | CEM | |
|---|---|---|---|---|
| Classe ($k$) | 1 | 2 | 1 | 2 |
| Proportions | 0.17 | 0.83 | 0.17 | 0.83 |
| $j$ | $\varepsilon_1^j$ | $\varepsilon_2^j$ | $\varepsilon_1^j$ | $\varepsilon_2^j$ |
| 1 | 0.16 | 0.22 | 0.16 | 0.22 |
| 2 | 0.14 | 0.20 | 0.14 | 0.20 |
| 3 | 0.16 | 0.12 | 0.16 | 0.12 |
| 4 | 0.14 | 0.20 | 0.14 | 0.20 |
| 5 | 0.16 | 0.08 | 0.16 | 0.08 |
| 6 | 0.17 | 0.20 | 0.17 | 0.20 |
| 7 | 0.15 | 0.16 | 0.15 | 0.16 |
| 8 | 0.14 | 0.18 | 0.14 | 0.18 |
| 9 | 0.13 | 0.18 | 0.13 | 0.18 |
| 10 | 0.16 | 0.10 | 0.16 | 0.10 |
| 11 | 0.14 | 0.08 | 0.14 | 0.08 |
| 12 | 0.16 | 0.18 | 0.16 | 0.18 |
| 13 | 0.17 | 0.10 | 0.17 | 0.10 |
| 14 | 0.18 | 0.12 | 0.18 | 0.12 |
| 15 | 0.13 | 0.18 | 0.13 | 0.18 |

TAB. 2.6 – *Paramètres $(p_k, \varepsilon_k^j)$ obtenus par les algorithmes EM et CEM pour les données moyennement mélangées (ES2 : $\varepsilon = 0.30$).*

| | EM | | CEM | |
|---|---|---|---|---|
| Classe ($k$) | 1 | 2 | 1 | 2 |
| Proportions | 0.16 | 0.84 | 0.16 | 0.84 |
| $j$ | $\varepsilon_1^j$ | $\varepsilon_2^j$ | $\varepsilon_1^j$ | $\varepsilon_2^j$ |
| 1 | 0.31 | 0.31 | 0.31 | 0.31 |
| 2 | 0.26 | 0.35 | 0.26 | 0.35 |
| 3 | 0.25 | 0.30 | 0.25 | 0.31 |
| 4 | 0.35 | 0.27 | 0.35 | 0.27 |
| 5 | 0.31 | 0.23 | 0.31 | 0.22 |
| 6 | 0.33 | 0.33 | 0.33 | 0.33 |
| 7 | 0.27 | 0.23 | 0.27 | 0.22 |
| 8 | 0.28 | 0.39 | 0.28 | 0.39 |
| 9 | 0.29 | 0.36 | 0.29 | 0.37 |
| 10 | 0.29 | 0.39 | 0.29 | 0.39 |
| 11 | 0.30 | 0.17 | 0.30 | 0.16 |
| 12 | 0.31 | 0.20 | 0.31 | 0.20 |
| 13 | 0.32 | 0.39 | 0.32 | 0.39 |
| 14 | 0.33 | 0.37 | 0.33 | 0.37 |
| 15 | 0.31 | 0.31 | 0.31 | 0.31 |

### Convergence

Comme nous l'avons précisé auparavant, EM converge généralement plus lentement que CEM. Ceci se retrouve vérifié dans notre cas. En effet, il faut en moyenne deux fois plus d'itérations à EM pour converger. Nous trouvons que, pour le premier tableau, CEM converge en moyenne en 4 itérations, alors qu'il en faut 8 pour EM. Pour les deux autres cas, c'est assez similaire (pour ES2, on a 12 itérations contre 7, et pour ES3, 24 contre 9).

TAB. 2.7 – *Paramètres $(p_k,\varepsilon_k^j)$ obtenus par les algorithmes EM et CEM pour les données très mélangées (ES3 : $\varepsilon = 0.45$).*

| | EM | | CEM | |
|---|---|---|---|---|
| Classe ($k$) | 1 | 2 | 1 | 2 |
| Proportions | 0.15 | 0.85 | 0.14 | 0.86 |
| $j$ | $\varepsilon_1^j$ | $\varepsilon_2^j$ | $\varepsilon_1^j$ | $\varepsilon_2^j$ |
| 1 | 0.42 | 0.35 | 0.42 | 0.36 |
| 2 | 0.43 | 0.30 | 0.43 | 0.29 |
| 3 | 0.42 | 0.41 | 0.41 | 0.43 |
| 4 | 0.51 | 0.48 | 0.50 | 0.52 |
| 5 | 0.49 | 0.39 | 0.48 | 0.40 |
| 6 | 0.51 | 0.44 | 0.52 | 0.40 |
| 7 | 0.44 | 0.47 | 0.45 | 0.43 |
| 8 | 0.43 | 0.45 | 0.43 | 0.45 |
| 9 | 0.45 | 0.54 | 0.45 | 0.52 |
| 10 | 0.48 | 0.36 | 0.48 | 0.33 |
| 11 | 0.44 | 0.44 | 0.45 | 0.40 |
| 12 | 0.48 | 0.37 | 0.48 | 0.33 |
| 13 | 0.45 | 0.52 | 0.46 | 0.50 |
| 14 | 0.42 | 0.44 | 0.42 | 0.40 |
| 15 | 0.46 | 0.44 | 0.46 | 0.38 |

### Partitionnement

Pour les deux premiers tableaux (classes assez séparées et moyennement séparées), les deux algorithmes trouvent les mêmes partition. Dans le troisième cas des classes peu séparées, l'algorithme CEM classe quelques objets dans la mauvaise classe. En fait, il vide la classe la plus petite pour mettre dans la plus grande. C'est un défaut connu de cet algorithme.

### Estimation des paramètres

Ici aussi, pour les deux premiers cas, EM et CEM trouvent les mêmes estimations des paramètres. Il n'y a pas de différences significatives entre les deux à ce niveau. Par contre, dans le troisième cas, avec ES3, on remarque que EM fournit des meilleures estimations que celles obtenues par CEM, et qui sont biaisées. En effet, il est en moyenne plus proche de la valeur théorique de 0,45. Nous montrons les résultats obtenus sur l'estimation des paramètres $(p_k,\varepsilon_k^j)$ pour les trois tableaux de données, dans les trois tableaux (2.5, 2.6, 2.7). Nous n'indiquons pas les noyaux obtenus, puisqu'ils sont à chaque fois identiques pour EM et CEM, et égaux aux centres simulés.

### Conclusion

On retrouve les phénomènes habituels et connus de ces deux algorithmes. On voit que pour des données assez séparées (comme c'était le cas pour ES1, et dans une moindre mesure pour ES2), ils trouvent de bons résultats, généralement les mêmes. Par contre, quand les données sont un peu moins séparées, et qu'il est plus difficile de les dissocier, l'algorithme EM est plus performant que CEM. Il permet d'avoir une estimation des paramètres beaucoup plus juste.

## 2.4 Application sur des données réelles

### 2.4.1 Congressional Votes

Ce tableau comprend les votes pour chacun des représentants du congrès américain[1], pour 16 votes clés, sur différents sujets (handicap, religion, immigration, armée, éducation, ...). Pour chaque vote, trois réponses ont été prises en comptes : pour, contre, et abstention. Les objets sont séparés en deux classes distinctes :

- Démocrates (267)
- Républicains (168)

Le tableau 2.8 montre les croisements entre la partition initiale et celles obtenues avec EM et CEM, ainsi que le croisement des partitions de EM et de CEM. Nous voyons ici que ces deux algorithmes trouvent la même partition, qui est assez proche de la partition initiale (13.1 % de mal-classés).

TAB. 2.8 – *Votes : croisement des partitions réelle et obtenues avec EM et CEM*

| | Dem. | Rep. |
|---|---|---|
| EM | 219 | 9 |
| | 48 | 159 |

| | Dem. | Rep. |
|---|---|---|
| CEM | 219 | 9 |
| | 48 | 159 |

| | CEM | |
|---|---|---|
| EM | 228 | 0 |
| | 0 | 207 |

### 2.4.2 ADN

Ce tableau de données a été obtenu à partir d'exemples tirés de Genbank 64.1[2], et a été plusieurs fois utilisé dans des articles d'apprentissage automatique. Il contient 3186 gènes, décrits par 60 variables, représentant des nucléotides, toutes avec 4 modalités possibles (A, C, G ou T). Ces objets sont répartis en 3 classes :

- 'intron → exon' ou *ie* (parfois nommé donneurs, 767 objets),
- 'exon → intron' ou *ei* (parfois nommé receveurs, 765 objets),
- ni l'un, ni l'autre, noté $n$ (1654 objets).

A partir de ce tableau, et selon les indications apportées par les créateurs des données, nous avons choisi de ne retenir que les variables 21 à 40, qui représentent les nucléotides les plus proches de la jonction du gène.

Dans le tableau 2.9, nous présentons le croisement de la partition initiale avec les partitions obtenues avec EM et CEM, puis le croisement de ces deux partitions obtenues. Nous remarquons que les deux algorithmes n'obtiennent pas la même partition, mais elles sont assez proches. EM parvient à obtenir un taux de mal-classés de 4.8 % et CEM un taux de 5.0 %. La différence n'est pas très significative. Il s'agit de 16 gènes mal classés.

1. Congressional Quarterly Almanac, 98th Congress, 2nd session 1984, Volume XL : Congressional Quarterly Inc. Washington, D.C., 1985

2. ftp://genbank.bio.net

TAB. 2.9 – *ADN : croisement des partitions réelle et obtenues avec EM et CEM*

| EM | *ie* | *ei* | *n* |
|---|---|---|---|
| | 36 | 741 | 52 |
| | 721 | 12 | 32 |
| | 10 | 12 | 1570 |

| CEM | *ie* | *ei* | *n* |
|---|---|---|---|
| | 722 | 15 | 38 |
| | 10 | 14 | 1568 |
| | 35 | 736 | 48 |

| | CEM | | |
|---|---|---|---|
| EM | 4 | 6 | 819 |
| | 765 | 0 | 0 |
| | 6 | 1586 | 0 |

## 2.4.3 Mushroom

Ce tableau de données a été obtenu sur le site de l'UCI Machine Learning Repositery[3]. Il contient les descriptions de 8124 champignons, grâce à 22 variables nominales (couleur, forme, taille, habitat, ...). Ils sont répartis en deux classes :

- Comestible (4208 champignons),
- Vénéneux ou inconnu et donc considéré comme potentiellement dangereux (3916).

Les croisements entre la partition initiale et les partitions obtenues avec les algorithmes EM et CEM sont présentées dans le tableau 2.10. Le croisement des classes obtenues avec EM et CEM est aussi présentées. Pour ces données, nous voyons que EM place 816 champignons vénéneux dans la classe des comestibles, et CEM en place 860. Le taux de mal-classés est ici de 10.0 % pour EM et 10.6 % pour CEM.

TAB. 2.10 – *Mushroom : croisement des partitions réelle et obtenues avec EM et CEM*

| EM | Com. | Ven. |
|---|---|---|
| | 4208 | 816 |
| | 0 | 3100 |

| CEM | Com. | Ven. |
|---|---|---|
| | 0 | 3056 |
| | 4208 | 860 |

| | CEM | |
|---|---|---|
| EM | 0 | 5024 |
| | 3056 | 44 |

## 2.4.4 Titanic

Ce tableau décrit l'âge (adulte ou enfant), le sexe et la classe (première, deuxième, troisième ou équipage) des 2201 personnes présentes sur le Titanic lors de son naufrage en pleine mer, et s'ils ont été naufragés (1490) ou rescapés (711) de cet accident[4].

Les croisement des partitions initiales et obtenues avec EM et CEM, sont présentés dans le tableau 2.11. Ici, EM parvient à obtenir un taux de mauvais classement de 22.4 %, contre 22.7 % pour CEM. En croisant les deux partitions de EM et CEM, on remarque qu'elles ne différent que de 64 personnes.

3. http://www.ics.uci.edu/~mlearn/MLRepository.html
4. http://www2.ncsu.edu/ncsu/pams/stat/info/jse/homepage.html

TAB. 2.11 – *Titanic : croisement des partitions réelle et obtenues avec EM et CEM*

| | Nau. | Sur. |
|---|---|---|
| EM | 126 | 344 |
| | 1364 | 367 |

| | Nau. | Sur. |
|---|---|---|
| CEM | 161 | 373 |
| | 1329 | 338 |

| | CEM | |
|---|---|---|
| EM | 470 | 0 |
| | 64 | 1667 |

## 2.5 Données manquantes

Un problème majeur des données réelles est l'absence possible de certaines valeurs. Et la plupart des méthodes de classification automatique font l'hypothèse que toutes les données sont présentes. Ainsi, il est courant que les objets présentant des données manquantes soient ignorés, mais cette solution peut introduire un biais à l'analyse. Une autre solution consiste à reconstruire les valeurs manquantes par différentes techniques, tel que l'imputation multiple (Little et Rubin, 1987; Schafer, 1997), avant de procéder à la classification.

L'approche probabiliste a permis de proposer une solution au problème de la classification de données binaires avec des données manquantes (Nadif et Govaert, 1993). Nous nous intéressons ici aux données qualitatives , où des valeurs sont absentes. Nous supposons que les données sont manquantes au hasard (*missing at random* ou MAR) : cela signifie que l'absence ne dépend pas des données manquantes, mais peut dépendre des valeurs observées dans le tableau.

### 2.5.1 Expression de $Q(\theta,\theta^{(q)})$

Notons $\mathbf{x}$ les données partiellement observées, avec $\mathbf{x} = (\mathbf{x}^o,\mathbf{x}^m)$ où $\mathbf{x}^o$ représente des valeurs observées alors que $\mathbf{x}^m$ représente les valeurs manquantes. Pour chaque objet $\mathbf{x}_i$, nous pouvons écrire $\mathbf{x}_i = (\mathbf{x}_i^o,\mathbf{x}_i^m)$ où $\mathbf{x}_i^o$ est l'ensemble des valeurs observées pour $i$ et $\mathbf{x}_i^m$ celles absentes. L'estimation conditionnelle de la log-vraisemblance selon un estimateur précédemment obtenue $\theta^{(q)}$ et $\mathbf{x}$ est donnée par

$$\begin{aligned} Q(\theta,\theta^{(q)}) &= \sum_{i=1}^{n}\sum_{k=1}^{s} t_{ik}^{(q)} \left(\log(p_k) + \log \varphi_k(\mathbf{x}_i^o;\alpha_k)\right) \\ &+ \sum_{i=1}^{n}\sum_{k=1}^{s} t_{ik}^{(q)} \mathbf{E}\left[\log \varphi_k(\mathbf{x}_i^m|\mathbf{x}_i^o;\alpha_k)|\mathbf{x}^o,\theta^{(q)}\right] \end{aligned}$$

où

$$t_{ik}^{(q)} = pr\{z_{ik} = 1|\mathbf{x}^o,\theta^{(q)}\} = \frac{p_k^{(q)}\varphi_k(\mathbf{x}_i^o;\alpha_k^{(q)})}{\sum_{k'=1}^{s} p_{k'}^{(q)}\varphi_{k'}(\mathbf{x}_i^o;\alpha_{k'}^{(q)})}$$

Notons que si toutes les valeurs sont observées, le second terme de $Q(\theta,\theta^{(q)})$ est nul. Ensuite, nous étudions modèle de mélange résultant pour les données qualitatives.

### 2.5.2 L'algorithme EM

Nous posons $O_i$ l'ensembles des variables $j$ où $x_i^j$ est observé, et $M_i$ l'ensemble des variables $j$ où $x_i^j$ est absent. Comme chaque composant du mélange est supposé

indépendant, nous avons

$$Q(\theta,\theta^{(q)}) = \sum_{i=1}^{n}\sum_{k=1}^{s} t_{ik}^{(q)}\left(\sum_{j\in O_i}\log\varphi_k(x_{ij}^o;\alpha_k) + \sum_{j\in M_i}\mathbf{E}\left[\log\varphi_k(x_{ij}^m|\mathbf{x}_i^o;\alpha_k)|\mathbf{x}^o,\theta^{(q)}\right]\right) + \sum_{i=1}^{n}\sum_{k=1}^{s} t_{ik}^{(q)}\log(p_k)$$

Ici, nous considérons le modèle le plus simple, les calculs sont identiques pour les autres variantes du modèle. Après de rapides calculs, la log-vraisemblance complète peut être écrite de la manière suivante

$$\begin{aligned} L(\mathbf{x},\theta) &= \sum_{i=1}^{n}\sum_{k=1}^{s} t_{ik}^{(q)}\sum_{j\in O_i}\log\left(\frac{\varepsilon}{(1-\varepsilon)(c_j-1)}\right)\delta(x_{ij}^o,m_k^j) \\ &+ \sum_{i=1}^{n}\sum_{k=1}^{s} t_{ik}^{(q)}\sum_{j\in M_i}\log\left(\frac{\varepsilon}{(1-\varepsilon)(c_j-1)}\right)\beta_k^{j(q)} \\ &+ \sum_{i=1}^{n}\sum_{k=1}^{s} t_{ik}^{(q)}\log(p_k) + nd\log(1-\varepsilon) \end{aligned}$$

où $\beta_k^{j(c)} = (1-2\varepsilon^{(q)})\delta(m_k^j,m_k^{j(q)}+\varepsilon^{(q)})$ qui peut être défini par

$$\beta_k^{j(q)} = \begin{cases} \varepsilon^{(q)} & \text{si } m_k^j = m_k^{j(q)} \\ 1-\varepsilon^{(q)} & \text{sinon} \end{cases}$$

Ainsi, les deux étapes de l'algorithme EM deviendront

**Étape Estimation** : Pour $k = 1,\ldots,s$ et pour $i = 1,\ldots,n$, on calcule les probabilités a posteriori $t_{ik}^{(q)}$

**Étape Maximisation** : Pour $k = 1,\ldots,s$, on calcule les paramètres $(p_k^{(q+1)},\alpha_k^{(q+1)})$. Pour la classe $k$, cela revient à faire

$$p_k^{(q+1)} = \frac{\sum_{i=1}^{n} t_{ik}^{(q)}}{n},$$

et pour $j = 1,\ldots,d$, $m_k^{j(q+1)}$ est la modalité qui minimise

$$e_k^{j(q+1)} = \sum_{\text{observé}} t_{ik}^{(q)}\left(\delta\left(x_{ij}^o,m_k^{j(q+1)}\right)\right) + \sum_{\text{manquant}} t_{ik}^{(q)}\beta_k^{j(q)}$$

et

$$\varepsilon^{(q+1)} = \frac{\sum_{k=1}^{s}\sum_{j=1}^{d} e_k^{j(q+1)}}{nd}.$$

Il apparaît clairement que le processus est tel qu'à chaque itération, les données manquantes ont été reconstituées par le paramètre $\varepsilon$ calculée à l'étape précédente.

### 2.5.3 Résultats

#### Données simulées

Pour étudier le comportement de l'algorithme EM, nous avons effectué des simulations de Monte-Carlo, en générant 30 échantillons pour chaque situation considérée, dépendant seulement des paramètres du modèle (ici, nous avons pris le modèle $\lfloor p_k,\varepsilon \rfloor$) et des proportions de données manquantes. Nous évaluons la performance de l'algorithme avec le pourcentage d'objets mal classés, en comparant la partition obtenue avec celle simulée. Pour illustrer ceci, dans la figure 2.5, nous reportons ces pourcentages pour les paramètres suivants

$s = 2$, $n = 1000$, $d = 10$,
$a_1 = (1,1,1,1,1,1,1,1,1,1)$,
$a_2 = (1,1,1,1,1,2,2,2,2,2)$,
$c_j \in \{3,4,5\}\ \forall\ j = 1,\ldots,d$.
$p_1 = 0.3, p_2 = 0.7$,
$\varepsilon$ variant entre 0.05 et 0.35.

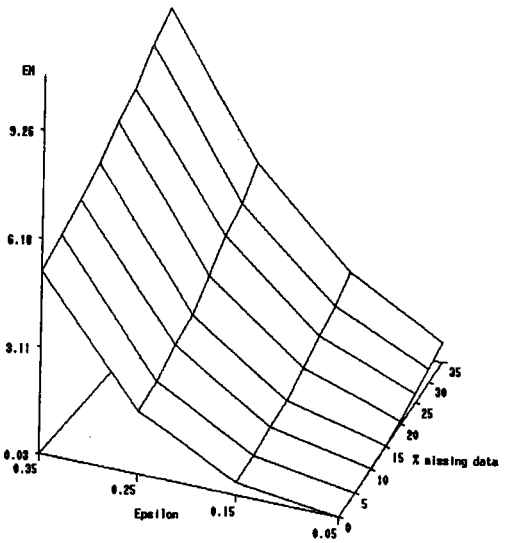



FIG. 2.5 – *Évolution des pourcentages moyens d'objets mal-classés.*

Il est très clair que le pourcentage d'objets mal-classés croit avec le pourcentage de données manquantes. Mais les résultats sont très encourageants, même lorsque le degré de mélange des données, conditionné par le paramètre $\varepsilon$, augmente.

#### Données réelles

Nous avons appliqué cette méthode sur deux tableaux réels, ADN et Votes. Nous avons supprimés aléatoirement plusieurs pourcentages de données (dans {5%, 10%, 15%, 20%, 25%, 30%, 35%}), et 20 fois pour chaque taux de données. Pour résumer le comportement de EM, nous utilisons la proportion d'objets mal-classés obtenue

pour chaque essai. Dans le tableau 2.12, nous reportons la moyenne et l'écart-type du taux d'erreur, calculés sur les 20 essais. La principale remarque à partir de ces expériences est que les résultats de EM sont très encourageants, même lorsque le taux de données manquantes est important.

TAB. 2.12 – *Données réels : moyenne et écart-type du taux de données mal-classés.*

| Pourcentage de données manquantes | Votes | DNA |
|---|---|---|
| 0 | 0.1310 | 0.048 |
| 5% | 0.1287 (0.0083) | 0.0555 (0.0051) |
| 10% | 0.1239 (0.0082) | 0.0625 (0.0044) |
| 15% | 0.1280 (0.0066) | 0.0735 (0.0049) |
| 20% | 0.1285 (0.0104) | 0.0800 (0.0000) |
| 25% | 0.1214 (0.0104) | 0.0940 (0.0050) |
| 30% | 0.1294 (0.0066) | 0.0999 (0.0055) |
| 35% | 0.1280 (0.0108) | 0.1110 (0.0055) |

## 2.6 Conclusion

Suite à ces variantes de modèles, dans la communauté statistique, il est de coutume d'analyser le comportement des méthodes en utilisant plusieurs modèles différents et de comparer leur adéquation aux données. Dans le contexte Fouille de Données, il est surtout important que le modèle choisi soit interprétable et surtout que son choix ne ralentisse pas le processus de Fouille de Données.

Dans le cas des données qualitatives, nous avons vu tous les modèles possibles, du plus simple, $\lfloor p,\varepsilon \rfloor$, au plus complexe, $\lfloor p_k,\alpha_k^{je} \rfloor$. Et forcément, plus le modèle est complexe, plus son nombre de paramètres à estimer est important, et plus le nombre d'objets doit être conséquent afin de bien estimer toutes les valeurs inconnues. Force est de constater que dans une optique de Fouille des Données, on travaillera exclusivement sur des tableaux, si ce n'est de grande taille, tout du moins de taille assez importante. Cette remarque nous permet de nous affranchir du problème du manque d'objets pour estimer les paramètres d'un modèle. De plus, à la suite de nombreux essais, nous avons pu remarquer que le modèle $\lfloor p_k,\alpha_k^{je} \rfloor$ est plus performant que tous les autres modèles, dès que l'on atteint des tailles de bases de données supérieures à 1000 objets. En dessous, ce modèle peut être pris en défaut lors de certaines configurations de données et pêche par manque de précision, contrairement à d'autres modèles plus simples.

A la suite de cette réflexion, et car le modèle des classes latentes nous permet de garder un maximum d'informations, nous avons décidé de n'utiliser par la suite que ce modèle, $\lfloor p_k,\alpha_k^{je} \rfloor$, le modèle des classes latentes.

Suite aux différents résultats obtenus lors de la comparaison des algorithmes, il en ressort que EM est meilleur que CEM, tant en terme d'estimation des paramètres qu'en terme de classification. L'inconvénient majeur de cet algorithme réside dans sa lenteur, dû à un nombre d'itération très important. Au contraire, CEM tire

son avantage principal dans sa rapidité à obtenir une partition. Il lui faut généralement moins d'une vingtaine d'itérations pour converger. Malheureusement, celui-ci a tendance à vider les petites classes, principalement lorsque les données sont très mélangées. Dans la suite, nous allons donc présenter différentes méthodes d'accélération de l'algorithme EM. Mais, avant ceci, nous étudierons le problème du choix du nombre de classes.

# Chapitre 3

# Choix du nombre de classes

Une des premières critiques que présentent les méthodes de classification directe est la connaissance requise du nombre de classes dans la population. Or, rarement dans le cas de données réelles, ce nombre de classes est connu, mais il est absolument nécessaire d'en avoir une idée.

Il est bien évident qu'il n'est pas possible de se baser uniquement sur la vraisemblance, ou la vraisemblance complétée. En effet, celles-ci augmentent avec le nombre de classes. Plus il y a de classes, plus la partition est ajustée aux données, et donc plus la vraisemblance est grande. Ainsi, chercher à maximiser la vraisemblance en faisant varier le nombre de classes $s$ nous conduit irrémédiablement à une partition en $n$ classes, où chaque objet serait dans sa propre classe, ce qui ne nous apporterait aucune information.

C'est ainsi que nous allons, dans un premier temps, nous baser sur des critères de choix du modèle, nous permettant aussi bien de choisir le modèle en tant que tel, ou bien de choisir le nombre de classes. Le but ici n'est pas forcément de retrouver le nombre exact de classes contenues dans la population, mais d'avoir la partition la plus profitable, en ayant un nombre ni trop grand ni trop petit de classes, aisément interprétables. Nous présentons dans la suite plusieurs de ces critères. Il en existe d'autres, mais nous avons préféré concentrer nos efforts sur ceux-ci. Une étude comparative est effectuée afin d'évaluer les performances et défaillances de chaque critère.

Enfin, nous considérerons ce problème sous la méthode mixte. Cette méthode qui combine les deux aspects de la classification non-supervisée : simple et hiérarchique a été déjà décrite dans le chapitre 1 pour les données continues. Nous avons montré comment on peut l'intégrer dans un cadre probabiliste en utilisant des mélanges Gaussiens. Ici, nous étendons son utilisation aux données qualitatives en nous basant sur le modèle des classes latentes.

## 3.1 Critères

### 3.1.1 Vraisemblance et vraisemblance complétée pénalisées

Les critères d'information présentés ici partagent un principe similaire. Ils se basent sur le maximum de vraisemblance du modèle à $s$ classes, qu'ils pénalisent

par le nombre de paramètres à estimer dans ce modèle. Leur formulation générale est

$$C(s) = -2\Big(L_{max}(s) + \gamma_C \times \nu(s)\Big) \tag{3.1}$$

où

- $L_{max}(s)$ est le maximum du critère de vraisemblance dans le domaine des paramètres du modèle à $s$ classes,
- $\gamma_C$ est un coefficient de pénalisation de la complexité spécifique à chaque critère (voir après),
- $\nu(s)$ est le nombre de paramètres libres dans le modèle à $s$ classes.

On observe que minimiser l'expression (3.1) revient à réaliser un compromis entre maximiser la vraisemblance et minimiser la complexité du modèle. En d'autres termes, les critères de ce type détectent les modèles qui fournissent un maximum de vraisemblance élevé en utilisant aussi peu de classes que possible.

Le *critère d'information Akaike* (AIC), initialement proposé par Akaike (1973), a pour expression

$$AIC(s) = -2L(s) + 2\nu(s), \text{ avec } \gamma_{AIC} = 1.$$

Bozdogan (1981, 1983) a suggéré un critère AIC modifié, appelé AIC3, défini comme suit

$$AIC3(s) = -2L(s) + 3\nu(s), \text{ avec } \gamma_{AIC3} = \frac{3}{2}.$$

Dans Banfield et Raftery (1993), en effectuant une approximation de la solution exacte au problème de la sélection d'un modèle approprié, on abouti à un critère appelé AWE (*approximate weight of evidence*), qui s'écrit :

$$AWE(s) = -2L_c(s) + \nu(s)\left(\frac{3}{2} + \log n\right), \text{ avec } \gamma_{AWE} = \frac{1}{2}\left(\frac{3}{2} + \log n\right).$$

On notera que pour une taille d'échantillon $n > 20$, les degrés de pénalisation de la complexité avec les différents critères respectent l'ordre suivant, du moins pénalisant au plus pénalisant :

$$0 < \gamma_{AIC} < \gamma_{AIC3} < \gamma_{AWE}$$

Pour traiter le problème de l'évaluation du nombre de classes $s \in \{1,...,S\}$, nous présentons ici quatre critères proposés dans un cadre bayésien (BIC, CS, ICL et ICL-BIC). Ces critères, qui ont été étudiés et comparés dans le cadre des mélanges Gaussiens, sont soit basés sur la vraisemblance intégrée, soit basés sur la vraisemblance complétée intégrée. Le nombre de classes approprié est choisi en maximisant ces critères.

### 3.1.2 Critère BIC

Le premier d'entre-eux est le critère BIC, qui peut aussi être vu comme un critère du type (3.1), avec $\gamma_{BIC} = \frac{\log n}{2}$. C'est ce calcul que nous utiliserons lorsque nous le comparerons aux critères précédemment présentés. À partir des données observées, la vraisemblance intégrée est obtenue par :

$$\varphi(\mathbf{x}|s) = \int \varphi(\mathbf{x}|s;\theta)p(\theta|s)d\theta \tag{3.2}$$

où $\varphi(\mathbf{x}|s;\theta) = \prod_{i=1}^{n} \varphi(\mathbf{x}_i|s;\theta)$ et $p(\theta|s)$ est la densité a priori de $\theta$. Une façon classique d'estimer cette fonction (3.2) est d'utiliser le critère BIC :

$$BIC(s) = L(\mathbf{x}|s;\hat{\theta}) - \frac{\nu(s)}{2}\log n$$

où $\hat{\theta}$ est l'estimateur au sens du maximum de vraisemblance de $\theta$ et $\nu(s)$ est le nombre de paramètres libres. Notons que cette approximation ne fait pas intervenir l'information a priori. De plus, elle est intuitive, car le second terme pénalise la log-vraisemblance par la complexité du modèle. D'autres approximations existent en tenant compte de l'information a priori, on peut citer le critère CS utilisé dans Autoclass, que nous détaillons ci-après.

### 3.1.3 Critère CS

Dans le logiciel AutoClass (Cheeseman et Stutz, 1996), une approximation de la vraisemblance intégrée est utilisé comme critère, noté CS (voir aussi (Chickering et Heckerman, 1997)). Celui-ci est donné par

$$CS(s) = L(\mathbf{x}|s;\hat{\theta}) - n\sum_{k=1}^{s}\hat{p}_k\log\hat{p}_k - \frac{\nu(s)}{2}\log n + K(n\hat{p}_1,\ldots,n\hat{p}_s)$$

où

$$K(n\hat{p}_1,\ldots,n\hat{p}_s) = \sum_{k=1}^{s}\log\Gamma\left(n\hat{p}_k+\frac{1}{2}\right) - \log\left(n+\frac{s}{2}\right) - s\log\Gamma\left(\frac{1}{2}\right) + \log\Gamma\left(\frac{s}{2}\right)$$

avec $\Gamma(t)$ représentant la fonction Gamma, et $\nu(s)$ représentant le nombre de paramètres inconnus pour le modèle.

### 3.1.4 Critère ICL

A partir des données complétées, la vraisemblance complétée intégrée s'obtient par :

$$\varphi(\mathbf{x},\mathbf{z}|s) = \int \varphi(\mathbf{x}|\mathbf{z},s;\theta)p(\theta|s)d\theta. \tag{3.3}$$

où $\varphi(\mathbf{x}|s;\theta) = \prod_{i=1}^{n}\prod_{k=1}^{s}\varphi(\mathbf{x}_i|\mathbf{z},s;\theta)^{z_{ik}}$. En supposant que $p(\theta|s) = p(\alpha|s)\,p(\mathbf{p}|s)$, Biernacki *et al.* (1998) ont montré que l'approximation de la log-vraisemblance complétée intégrée notée ICL s'écrit:

$$ICL(s) = L(\mathbf{x},\mathbf{z};\tilde{\theta}) - \sum_{k=1}^{s} n_k \log \tilde{p}_k - \frac{\nu(s)}{2} \log n + S(n_1,\ldots,n_s).$$

avec

$$S(n_1,\ldots,n_s) = \sum_{k=1}^{s} \log \Gamma\left(n_k + \frac{1}{2}\right) - \log\left(n + \frac{s}{2}\right) - s \log \Gamma\left(\frac{1}{2}\right) + \log \Gamma\left(\frac{s}{2}\right)$$

où $\Gamma(t)$ représente la fonction Gamma, $n_k$ est le nombre d'objets dans la classe $k$, et $\nu(s)$ le nombre de paramètres à estimer dans $\alpha$. Notons que $\tilde{\theta}$ ne correspond pas forcément à l'estimateur au sens du maximum de vraisemblance de $\theta$. Par contre, dès lors qu'on considère que la partition est obtenue par l'algorithme EM, alors on peut supposer que $(\hat{\theta} = \tilde{\theta})$ et établir une relation entre le critère CS et ICL.

$$ICL(s) = CS(s) + \sum_{i=1}^{n} \sum_{k=1}^{s} \hat{t}_{ik} \log \hat{t}_{ik}.$$

Aussi, lorsque les tailles des classes sont suffisamment larges, le critère ICL se réduit au critère suivant que l'on note ICL-BIC qui traduit une approximation *à la BIC* d'une vraisemblance complétée intégrée :

$$ICL\text{-}BIC(s) = BIC(s) + \sum_{i=1}^{n} \sum_{k=1}^{s} \hat{t}_{ik} \log \hat{t}_{ik}. \tag{3.4}$$

### 3.1.5 Essais sur des données simulées

Pour illustrer les comportements des quatre derniers critères décrits précédemment, nous avons choisi de présenter ici les résultats à partir de tableaux simulés. Nous avons généré deux tableaux de 5000 objets, répartis en 3 classes avec 20 variables, chacune à 4 modalités, distribuées comme précisé dans le tableau suivant, pour $j = 1,\ldots,d$:

| $e$ | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| $m_1^{je}$ | 0.4 | 0.2 | 0.2 | 0.2 |
| $m_2^{je}$ | 0.2 | 0.4 | 0.2 | 0.2 |
| $m_3^{je}$ | 0.2 | 0.2 | 0.4 | 0.2 |

Pour les proportions, nous avons choisi de prendre :

- Pour Sim1 : $p_1 = 0.33$, $p_2 = 0.33$ et $p_3 = 0.34$
- Pour Sim2 : $p_1 = 0.25$, $p_2 = 0.70$ et $p_3 = 0.05$

Les évolutions des critères sont présentés dans la figure 3.1. Pour le premier tableau simulé, CS obtienne le bon nombre de classes, et deux classes pour le second. ICL trouve aussi trois classes pour Sim1, mais qu'une seule classe pour Sim2. Nous avons choisi ici de ne pas représenter les courbes pour BIC et ICL-BIC car ils ont respectivement les mêmes comportement que CS et ICL.

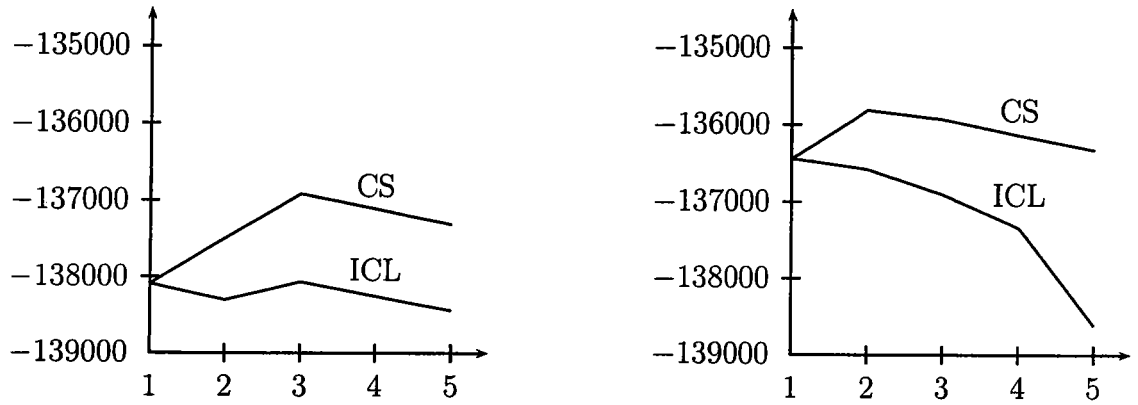



FIG. 3.1 – *Critères CS (identique à BIC) et ICL (identique à ICL-BIC) pour Sim1 (à gauche) et Sim2 (à droite), avec* $s = 1,\ldots,5$.

### 3.1.6 Étude détaillée du comportement de CS, BIC et ICL

Nous proposons ici une étude des critères de choix du nombre de classes CS, BIC et ICL, qui proviennent tous du cadre bayésien (Jollois *et al.*, 2002). Afin de pouvoir comprendre leur différent comportement selon le mélange des classes, il est nécessaire de simuler différents degrés de confusion des données. Malheureusement, il est très difficile d'évaluer le degré de mélange. De plus, le calcul de la probabilité d'erreur théorique est complexe et très difficile à mettre en œuvre. Pour remédier à ce problème, nous avons eu recours à des simulations de Monte Carlo. En générant avec les mêmes paramètres 500 tableaux différents, et en faisant la moyenne du nombre d'objets mal-classés obtenu par CEM initialisé avec les vrais paramètres, nous avons pu définir les différents degrés de mélange correspondant aux pourcentages d'objets mal-classés, définis comme suit :

| Degré de mélange | + | ++ | +++ |
|---|---|---|---|
| Pourcentage d'objets mal-classés | ∼ 5 % | ∼ 15 % | ∼ 25 % |

Ainsi, nous avons analysé le comportement des critères dans différentes situations (tailles de données petites, moyennes et grandes, proportions égales ou différentes, degré de mélange varié). Dans toutes les expériences ci-après, la partition a été dérivée de l'estimateur du maximum de vraisemblance (MLE) de $\theta$, obtenu avec l'algorithme EM initialisé de la manière suivante : premièrement, l'algorithme CEM est lancé $r$ fois (ici, $r = 20$), à partir de centres au hasard. Quand il fournissait une partition sans classe vide, EM est initialisé avec les paramètres dérivés de cette partition. Par contre, quand CEM fournit des partitions avec au moins une classe vide, l'algorithme EM est alors lancé $r$ fois à partir de centres au hasard.

Pour illustrer le comportement des critères CS, ICL et BIC, nous avons étudié leurs performances sur des données simulées, avec $n = 500$, 1000 et 5000, et $d = 20$, suivant le modèle des classes latentes avec $s = 3$, $c^j = 4\ \forall j = 1,\ldots,d$, et en faisant varier les paramètres de la façon suivante :

- les proportions sont supposées égales ($p = \{0.33,0.33,0.34\}$) ou différentes ($p = \{0.10,0.30,0.60\}$),
- les classes sont supposées bien séparées (+), moyennement séparées (++) et peu séparées (+++)

Pour chaque simulations de Monte Carlo, nous avons généré 20 échantillons de chaque situations, et nous calculons la moyenne des valeurs obtenues pour chaque critère CS, ICL et BIC. Dans nos expériences, nous avons remarqué que CS et BIC fournissaient les mêmes résultats. Ainsi, nous avons décidé de ne reporter que les critères CS et ICL dans les figures 3.2, 3.3 et 3.4. Et quand nous commentons le critère CS, nous nous référons au couple (CS, BIC).

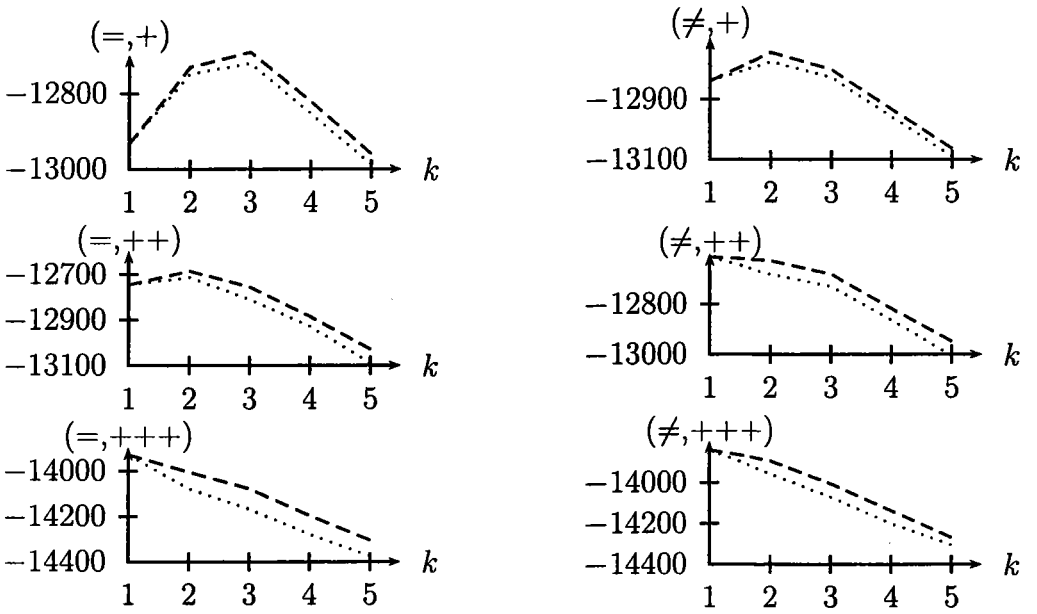



FIG. 3.2 – $n = 500$, $d = 20$ : *comportement de CS (tiret) et ICL (pointillé) pour les proportions égales (=) ou différentes (≠) et les classes très séparées (+), moyennement séparées (++) et peu séparées (+++).*

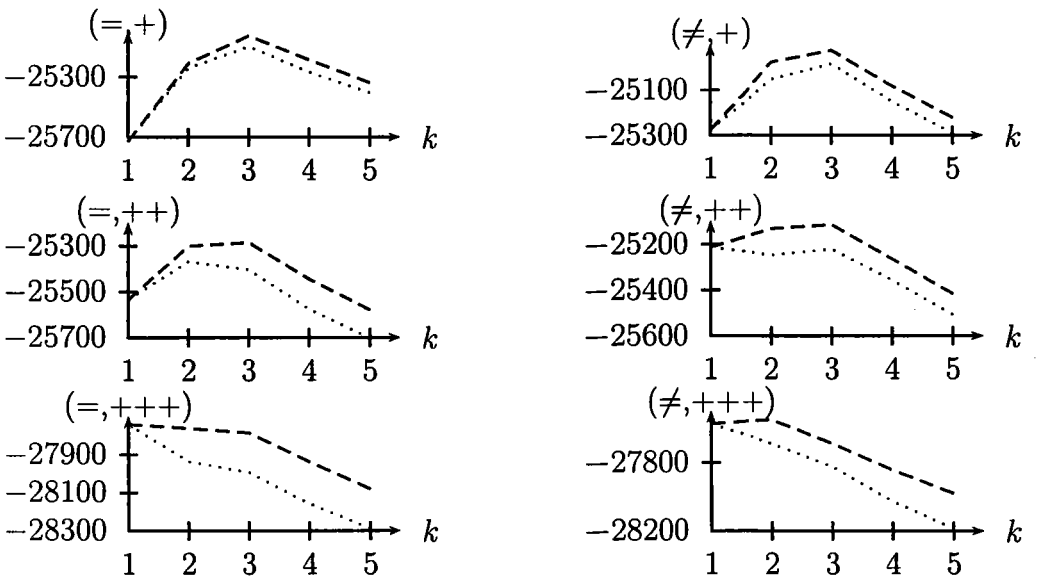



FIG. 3.3 – $n = 1000$, $d = 20$ : *comportement de CS (tiret) et ICL (pointillé).*

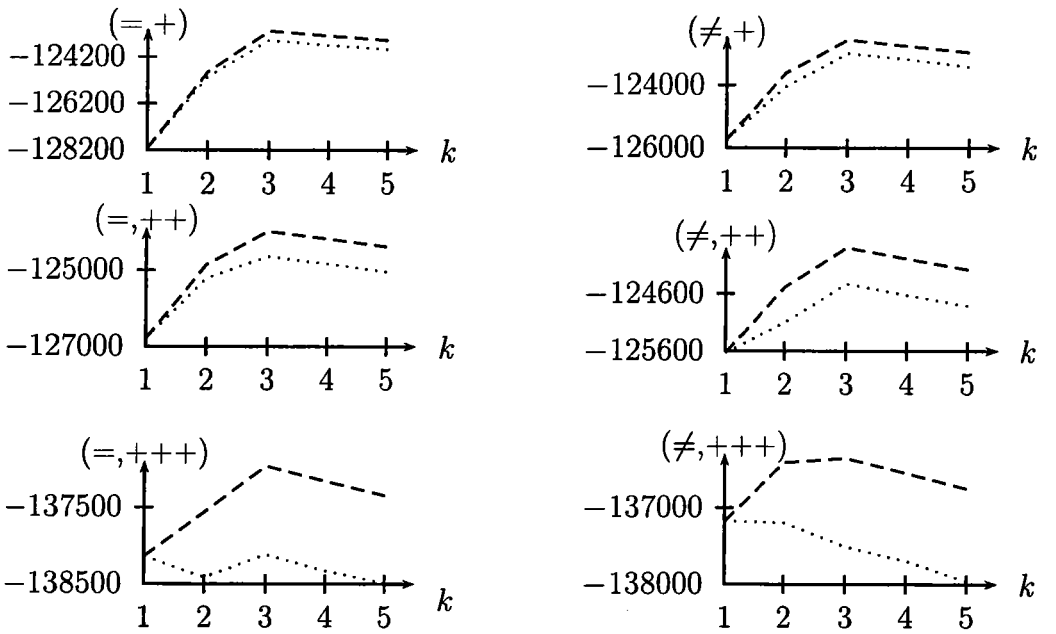



FIG. 3.4 – $n = 5000$, $d = 20$: *comportement de CS (tiret) et de ICL (pointillé).*

Les performances de CS et ICL pour trouver le bon nombre de classes dépendent du nombre d'objets $n$, des proportions $p_k$ et du degré de confusion des classes. A partir des résultats présentés, nous pouvons déduire les différentes remarques suivantes :

- CS et ICL sous-estiment le nombre de classes quand $n = 500$, mais cette faiblesse disparaît lorsque $n$ croit,
- le critère ICL donne de bons résultats particulièrement quand les classes sont supposées peu mélangées.
- même si les proportions sont dramatiquement différentes, CS donne de très bons résultats, principalement pour un grand nombre d'objets.
- plus les classes sont mélangées, plus les deux critères rencontrent des difficultés à trouver le bon nombre de classes $s$.

Nos expériences nous permettent de déduire que CS et BIC semble être plus performant que ICL pour le modèle des classes latentes, contrairement au cas des mélanges Gaussiens. Même dans le dans de classes très mélangées, CS et BIC estiment très bien le nombre de classes.

### Comparaison de BIC et ICL sur des données réelles

Afin de comparer ces deux critères, nous les avons appliqué sur les données ADN, déjà présentées dans le chapitre 2. Nous reportons dans la figure 3.5 l'évolution des critères BIC et ICL. Respectivement, les critères CS et ICL-BIC ont les mêmes comportements. On peut citer qu'AutoClass trouve 8 classes pour le tableau de données réelles ADN, ce qui est beaucoup plus que le vrai nombre de classes (3).

On voit que BIC trouve quatre classes, alors qu'ICL en trouve trois. Mais on remarque que les valeurs pour 3, 4 et 5 classes, pour BIC, sont très proches. Les

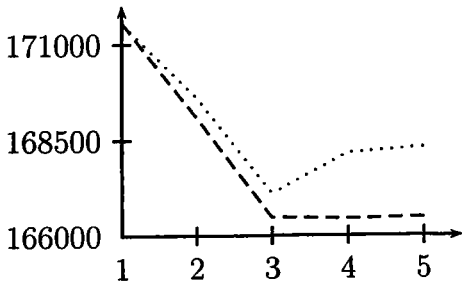



FIG. 3.5 – *Évolution des critères BIC (tiret) et ICL (pointillé) pour les données ADN.*

décompositions en trois et quatre classes se font de la manière suivante :

| | *ie* | *ei* | *n* |
|---|---|---|---|
| EM | 36 | 741 | 52 |
| | 721 | 12 | 32 |
| | 10 | 12 | 1570 |

| | *ie* | *ei* | *n* |
|---|---|---|---|
| EM | 3 | 8 | 575 |
| | 12 | 20 | 1006 |
| | 38 | 721 | 40 |
| | 714 | 16 | 33 |

Pour $s = 4$, la classe la plus importante ($n$) est séparée en deux classes plus petites. Cette décomposition semble donc tout de même appropriée. Même si BIC ne donne pas le nombre de classes réel, il propose une solution en quatre classes intéressante.

### 3.1.7 Simulation de Monte Carlo : comparaison des critères

Ici, nous avons choisi d'étudier 6 différents critères de choix du nombre de classes (BIC, ICL, AIC, AIC3, AWE et NEC) sur des données qualitatives. Dans cette comparaison, nous avons simulées plusieurs situations différentes, en prenant les paramètres suivants :

- $n = 500$ et 1000,
- $d = 30$,
- $c^j = 3$ pour $j = 1, \ldots, 10$, 4 pour $j = 11, \ldots, 20$ et 5 pour $j = 21, \ldots, 30$
- 3 degrés de mélange différents :
  - . classes peu mélangées (+),
  - . moyennement mélangées (++),
  - . et très mélangées (+++)

Pour chaque situation, nous avons choisi de simuler 30 tableaux. Les décomptes du nombre de classes obtenus par chaque critère pour chaque tableau sont reportés dans les tableaux 3.1 et 3.2.

Lorsque le nombre d'objets est petit ($n = 500$), BIC, CS et ICL sont bons lorsque les données sont séparées. Des lors que celle-ci deviennent de plus en plus mélangées, ces trois critères faiblissent, pour sous-estimer finalement le nombre de classes. Le critère AIC a tendance à sur-estimer $s$, et NEC a des comportements très divers, et même s'il semble fournir de bons résultats pour (+++), il n'en est pas de même pour les deux autres situations. Enfin, AIC3 se montre performant pour (+) et (++), mais pas pour (+++).

TAB. 3.1 – *Résultats pour $n = 500$. Le nombre indiqué est le nombre de fois où le critère choisi le nombre de classes $s$, pour chacune des 30 simulations.*

| Degré de mélange | Critères | $s$ | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| + | BIC | 0 | 3 | 27 | 0 | 0 | 0 | 0 |
| | ICL | 0 | 3 | 27 | 0 | 0 | 0 | 0 |
| | AIC | 0 | 0 | 6 | 14 | 9 | 1 | 0 |
| | AIC3 | 0 | 0 | 30 | 0 | 0 | 0 | 0 |
| | AWE | 0 | 20 | 10 | 0 | 0 | 0 | 0 |
| | NEC | 0 | 7 | 11 | 10 | 0 | 1 | 1 |
| ++ | BIC | 17 | 13 | 0 | 0 | 0 | 0 | 0 |
| | ICL | 20 | 10 | 0 | 0 | 0 | 0 | 0 |
| | AIC | 0 | 0 | 1 | 9 | 9 | 8 | 3 |
| | AIC3 | 0 | 3 | 27 | 0 | 0 | 0 | 0 |
| | AWE | 30 | 0 | 0 | 0 | 0 | 0 | 0 |
| | NEC | 0 | 2 | 1 | 0 | 3 | 8 | 16 |
| +++ | BIC | 30 | 0 | 0 | 0 | 0 | 0 | 0 |
| | ICL | 30 | 0 | 0 | 0 | 0 | 0 | 0 |
| | AIC | 0 | 0 | 0 | 7 | 8 | 8 | 7 |
| | AIC3 | 22 | 8 | 0 | 0 | 0 | 0 | 0 |
| | AWE | 30 | 0 | 0 | 0 | 0 | 0 | 0 |
| | NEC | 0 | 13 | 1 | 0 | 1 | 1 | 14 |

Avec un nombre d'objets plus importants ($n = 1000$), on remarque que le critère BIC se montre plus robuste, et parvient à mieux estimer le nombre de classes, mis à part pour (+++). ICL et AWE ont tendance à sous-estimer $s$, dès que le mélange devient trop important. AIC continue à sur-estimer les nombre de classes, alors que NEC trouvent toujours $s = 2$. AIC3 est ici le plus performant, en trouvant pour chaque situation $s = 3$.

Afin de voir si BIC avait un comportement plus robuste lorsque le nombre d'objets augmentait, nous avons choisi simuler 3 tableaux avec $n = 10000$, en prenant les trois degrés de mélange précédent. A partir des figures 3.6, 3.7 et 3.8, nous pouvons faire les remarques suivantes :

- BIC et AIC3 trouvent toujours le bon nombre de classes
- ICL et AWE trouvent $s = 1$ pour (+++), et $s = 3$ pour les deux autres.
- AIC trouve $s = 6$ pour (+) et (+++), et $s = 7$ pour (++).
- Nous avons décidé de ne pas reporter dans les figures le critère NEC. Il trouve pour chaque tableau $s = 2$.

## 3.2 Classification mixte

Comme nous l'avons vu précédemment, la méthode de (Wong, 1977) nous permet de combiner les avantages de la classification directe pour résumer l'information et pour estimer correctement la partition et/ou les paramètres du mélange, avec ceux de la classification hiérarchique pour obtenir un nombre de classes judicieux et une initialisation intéressante pour le premier algorithme (voir Fig. 1.3)

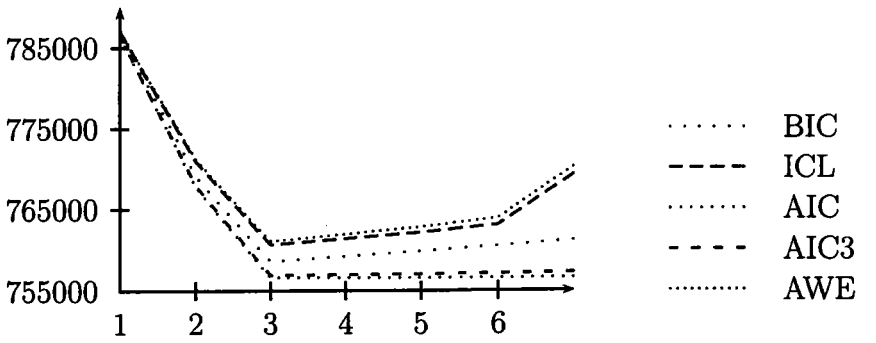



FIG. 3.6 – *Évolution des critères de choix du nombre de classes pour (+) avec* $n = 10000$.

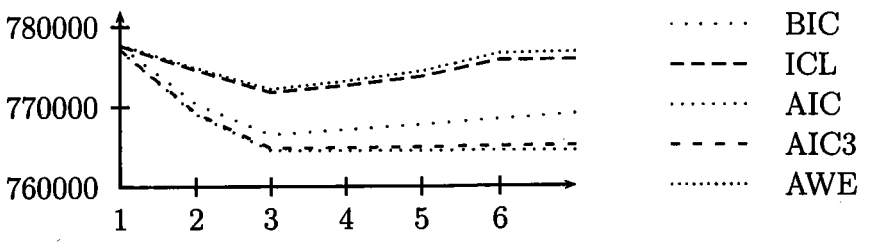



FIG. 3.7 – *Évolution des critères de choix du nombre de classes pour (++) avec* $n = 10000$.

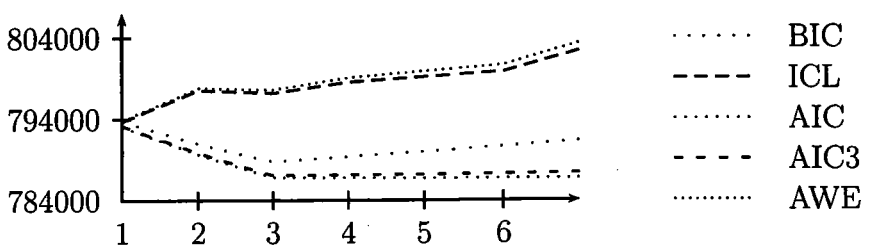



FIG. 3.8 – *Évolution des critères de choix du nombre de classes pour (+++) avec* $n = 10000$.

TAB. 3.2 – *Résultats pour $n = 1000$. Le nombre indiqué est le nombre de fois où le critère choisi le nombre de classes $s$, pour chacune des 30 simulations.*

| Degré de mélange | Critères | $s$ | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| + | BIC | 0 | 0 | 30 | 0 | 0 | 0 | 0 |
| | ICL | 0 | 0 | 30 | 0 | 0 | 0 | 0 |
| | AIC | 0 | 0 | 0 | 5 | 9 | 9 | 7 |
| | AIC3 | 0 | 0 | 30 | 0 | 0 | 0 | 0 |
| | AWE | 0 | 0 | 30 | 0 | 0 | 0 | 0 |
| | NEC | 0 | 21 | 7 | 2 | 0 | 0 | 0 |
| ++ | BIC | 0 | 6 | 24 | 0 | 0 | 0 | 0 |
| | ICL | 0 | 27 | 3 | 0 | 0 | 0 | 0 |
| | AIC | 0 | 0 | 0 | 3 | 4 | 13 | 10 |
| | AIC3 | 0 | 0 | 30 | 0 | 0 | 0 | 0 |
| | AWE | 30 | 0 | 0 | 0 | 0 | 0 | 0 |
| | NEC | 0 | 30 | 0 | 0 | 0 | 0 | 0 |
| +++ | BIC | 0 | 21 | 9 | 0 | 0 | 0 | 0 |
| | ICL | 30 | 0 | 0 | 0 | 0 | 0 | 0 |
| | AIC | 0 | 0 | 0 | 1 | 4 | 7 | 18 |
| | AIC3 | 0 | 0 | 30 | 0 | 0 | 0 | 0 |
| | AWE | 30 | 0 | 0 | 0 | 0 | 0 | 0 |
| | NEC | 0 | 30 | 0 | 0 | 0 | 0 | 0 |

Bien sûr, alors que Wong s'est placé dans un contexte de classification de données gaussiennes, et avec des algorithmes tels que $k$-means, nous allons nous placer dans le cadre des modèles de mélange, sur des données qualitatives.

### 3.2.1 Modèle de mélange associé

A partir de chaque objet dans sa propre classe, le but de la classification hiérarchique est d'assembler à chaque étape les deux classes les plus proches, jusqu'à ce qu'il n'y ait plus qu'une seule classe. Nous sommes donc contraints d'établir une notion de distance entre classes, afin de déterminer quelles sont les deux plus proches. Pour cela, et partant du fait que

$$L_c(\mathbf{x},\mathbf{z};\theta) = \sum_{k=1}^{s} L_c(\mathbf{x},z_k;\theta_k),$$

où

$$L_c(\mathbf{x},z_k;\theta_k) = \sum_{i \in z_k} \log(\varphi(\mathbf{x}_i;\alpha_k))$$

est la contribution d'une classe à la log-vraisemblance classifiante, la distance entre deux classes $z_k$ et $z_{k'}$ peut s'exprimer de la manière suivante :

$$\Delta(z_k,z_{k'}) = L_c(\mathbf{x},z_k;\theta_k) + L_c(\mathbf{x},z_{k'};\theta_{k'}) - L_c(\mathbf{x},z_{kk'};\theta_{kk'}).$$

Dans le cas des données qualitatives, avec le modèles des classes latentes $\lfloor p_k,\alpha_k^{je} \rfloor$, la distance entre deux classes se définit donc ainsi :

$$\Delta(z_k,z_{k'}) = \sum_{j=1}^{d}\sum_{e=1}^{c^j}\left(\#z_k\alpha_k^{je}\log\left(\frac{\alpha_k^{je}}{\alpha_{kk'}^{je}}\right) + \#z_{k'}\alpha_{k'}^{je}\log\left(\frac{\alpha_{k'}^{je}}{\alpha_{kk'}^{je}}\right)\right)$$

La classe $z_{kk'}$ est obtenue en regroupant les deux classes $z_k$ et $z_{k'}$. Il est évident que $\Delta(z_k,z_{k'})$ est défini tel que la log-vraisemblance classifiante est minimisée à chaque étape. En effet, le terme $\Delta(z_k,z_{k'})$ est toujours positif (de plus, on choisit le plus petit), et on a

$$L_c(\mathbf{x},\mathbf{z}';\theta) = L_c(\mathbf{x},\mathbf{z};\theta) - \Delta(z_k,z_{k'})$$

où $\mathbf{z}'$ désigne la nouvelle partition obtenue, après regroupement des classes $z_k$ et $z_{k'}$. Ainsi la log-vraisemblance classifiante $L_c(\mathbf{x},\mathbf{z}';\theta)$ sera ainsi minimisée à chaque étape.

Chaque objet est initialement dans sa propre classe. Les deux classes les plus proches sont fusionnées, selon la distance $\Delta(z_k,z_{k'})$, pour former une nouvelle classe. Ce processus est répété jusqu'à l'obtention d'une seule classe finale regroupant tous les individus.

Puisque la distance dépend seulement des deux classes $z_k$ et $z_{k'}$, les distances entre les autres classes ne sont pas affectées par ce regroupement. De plus, les paramètres $\theta_k$ et les valeurs de $\Delta(z_k,z_{k'})$ dépendent seulement des statistiques exhaustives, qui sont stockées pour chaque classe.

### 3.2.2 Combinaison d'algorithmes

Le but est donc ici d'obtenir une partition d'une population sans avoir a priori une idée sur le nombre de classes, et surtout sans avoir à faire autant de calculs qu'on ne le ferait en comparant un critère. L'objectif d'une telle tâche est de trouver une partition judicieuse des données, interprétable et compréhensible, avec un nombre de classes approprié.

Une méthode simple pour ceci est d'utiliser une méthode de classification hiérarchique. Ainsi, grâce à l'évolution du critère, et par une méthode graphique, il est possible de déterminer un coude, et d'en déduire un nombre de classes $s$ approprié. Cette méthode hiérarchique demande malheureusement un nombre de calcul trop important dès que la taille des données augmente (complexité quadratique). Il est donc impossible de l'appliquer directement sur le tableau d'origine, dans une optique de Fouille de Données, où les données se comptent en milliers, voire millions parfois. Il est donc nécessaire de résumer l'information.

Et pour ce faire, l'utilisation de la classification directe s'impose naturellement, et nous permet de réaliser cette opération. En effet, en utilisant un nombre de classes assez grand pour nous permettre de garder un maximum d'information, il sera ainsi possible d'obtenir une matrice réduite, avec laquelle nous pourrons travailler par la suite.

Une fois le résumé obtenu, une fois l'arbre hiérarchique sur ce résumé obtenu, et donc une fois le nombre de classes choisi, il est nécessaire de relancer un algorithme de classification directe pour obtenir une partition profitable des données. Une initialisation possible est de prendre la partition obtenue avec la classification hiérarchique, et ainsi de l'améliorer jusqu'à la convergence.

### 3.2.3 Application sur des données simulées

Pour illustrer cette méthodologie, nous présentons quelques résultats obtenus à partir de 3 tableaux simulés. Nous avons choisi $s = 3$, $n = 5000$, $d = 10$, $c^j = 3, \forall j = 1, \ldots, d$. Pour les trois tableaux, les classes sont définies comme suit :

1. très séparées (+),
2. moyennement séparées (++),
3. peu séparées (+++).

Nous avons utilisé les deux algorithmes EM et CEM, avec $s = 10$, pour obtenir les centres utilisés ensuite par l'algorithme CAH. Dans la figure 3.9, nous voyons l'évolution du critère $L_c$ et le dendrogramme associé, pour le tableau (+) avec la méthode EM → CAH. Dans le tableau 3.3, nous voyons que cette méthode trouve exactement le bon nombre de classes pour les 3 tableaux simulés, contrairement à CEM → CAH, qui sous-estime $s$, et AutoClass qui a tendance à surestimer ce nombre de classes. On peut préciser que la méthode CEM → CAH ne propose pas de coude dans la courbe de $L_c$ et n'arrive donc pas à déceler un nombre de classes pour (+++).

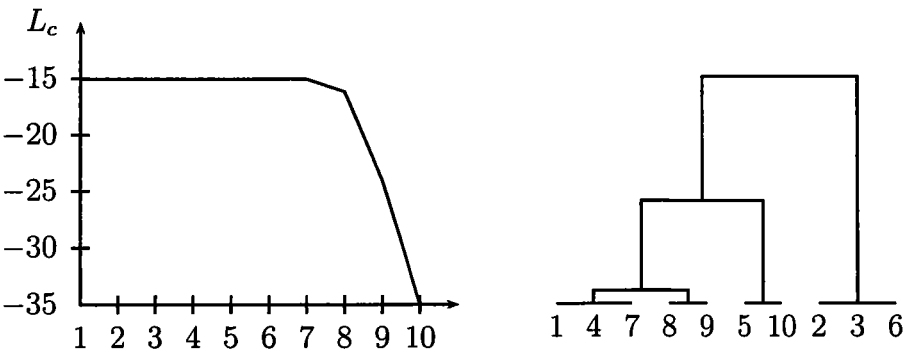



FIG. 3.9 – *Évolution du critère de la CAH ($L_c$) et arbre hiérarchique pour la situation (+) , avec EM comme initialisation*

TAB. 3.3 – *Comparaison du nombre de classes données par notre méthode et par AutoClass. Pour (+++), CEM suivi de CAH ne fournit pas de nombres de classes particulier (i. e. il n'y a pas de coude dans la courbe de $L_c$).*

| | *EM → CAH* | *CEM → CAH* | AutoClass |
|---|---|---|---|
| + | $s = 3$ | $s = 2$ | $s = 5$ |
| ++ | $s = 3$ | $s = 2$ | $s = 4$ |
| +++ | $s = 3$ | NA | $s = 3$ |

### 3.2.4 Application sur des données réelles

Afin de valider cette méthode, nous avons appliqué celle-ci sur le tableau de données réelles déjà présenté dans le chapitre 2, ADN. Nous rappelons que celui-ci comporte 3 classes.

Pour appliquer la méthode mixte, nous avons choisi de lancer tout d'abord EM avec un nombre de classes $s = 7$. Sur la matrice de données résumée que nous avons obtenu, nous avons appliqué la méthode CAH avec modèle de mélange. Nous reportons dans la figure 3.10 l'évolution de la log-vraisemblance classifiante, durant les étapes de l'algorithme. Nous voyons apparaître très nettement un coude à l'itération 4. Celle-ci correspond à couper l'arbre hiérarchique en 3 classes (voir figure 3.11). Ainsi, nous notons que cette méthode permet de choisir correctement le nombre de classes approprié.

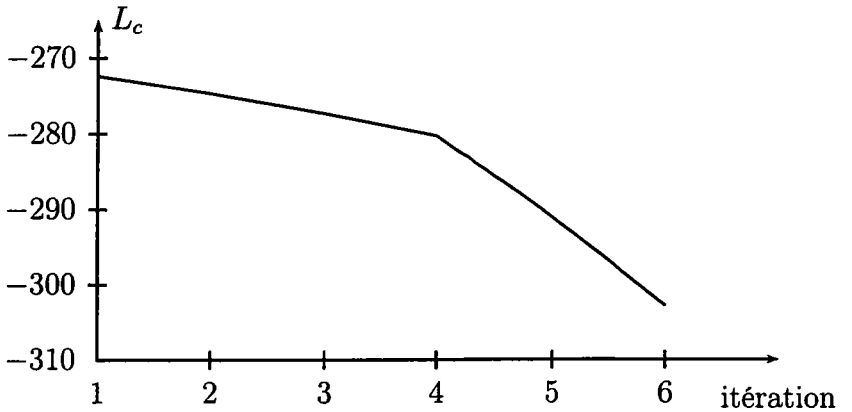



FIG. 3.10 – *Évolution de la log-vraisemblance classifiante $L_c$ en fonction des itérations, pour ADN, avec la CAH.*

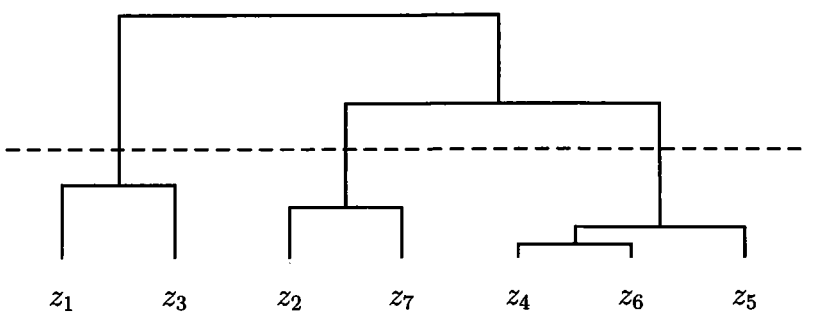



FIG. 3.11 – *Arbre hiérarchique des classes obtenus avec EM, pour ADN.*

## 3.3 Conclusion

Nous avons présenté dans ce chapitre deux méthodes de choix du nombre de classes. La première, très classique, consiste à calculer pour un certain ensemble de nombres de classes possibles un critère, et ensuite de choisir le nombre $s$ qui optimise ce dernier. Dans ce cadre, il est apparu que BIC et AIC3 s'avéraient les plus performants. Cette performance croît avec la taille des données. Au contraire, nous avons vu les limites d'ICL, surtout dans le cas de données très mélangées. Les autres critères ne se montrent guère intéressants dans le cas de données qualitatives.

La seconde méthode est une extension aux modèles de mélanges de la méthode mixte. Nous avons montré que celle-ci se comportait très bien pour des données si-

mulées, principalement la stratégie associant EM à CAH. Celle avec CEM a tendance à sous-estimer le nombre de classes.

Il existe une troisième méthode, que nous n'avons pas détaillé ici. Il s'agit de la version stochastique de l'algorithme EM, SEM (Celeux et Diebolt, 1985). Celle-ci insère une étape de classification stochastique entre les deux étapes d'Estimation et de Maximisation. Cette étape repose sur un *principe d'affectation aléatoire* (*Random Imputation Principle*) et affecte chaque objet à une classe tirée selon une loi multinomiale d'ordre 1 et de paramètres $t_{ik}^{(q)}$ (pour $k = 1, \ldots,$). Lorsqu'une classe se révèle trop petite (effectif inférieur à $d + 1$), on décide de la supprimer, et de continuer l'algorithme avec $s - 1$ classes. Lors de nos évaluations de cet algorithme pour le choix du nombre de classes, il est apparu que cette méthode avait une tendance marquée à sous-estimer celui-ci. C'est pourquoi nous ne l'avons pas reporté dans nos résultats.

# Chapitre 4

# Accélération de l'algorithme EM

Un des objectifs de la classification automatique est de réduire l'information contenue dans les données en une matrice de taille réduite. Dans un contexte Fouille de Données, un tel processus sera souvent utilisé sur des données de grande taille. De ce fait, il est absolument nécessaire que les méthodes utilisées soient capables à la fois de travailler sur des grandes bases de données, dans des temps raisonnables, et qu'ils donnent des résultats très performants, car déterminant la qualité des connaissances acquises par la suite.

Ainsi, nous allons nous concentrer sur l'amélioration possible de l'utilisation de l'algorithme **EM**. Pour ceci, nous allons prendre deux chemins distincts. Le premier consiste à initialiser d'une bonne manière EM, pour la recherche du meilleur optimum. En effet, nous avons déjà précisé que le résultat fourni par cet algorithme est très dépendant du point de départ de celui-ci. Nous allons donc chercher à rendre ce point de départ assez proche de la bonne solution. Pour ceci, nous comparons plusieurs stratégies d'initialisation. Ensuite, nous chercherons à accélérer directement l'algorithme EM. Pour ce faire, et en nous basant sur les résultats de (Neal et Hinton, 1998), nous allons chercher à minimiser le nombre de calculs de probabilités a posteriori.

## 4.1 Stratégies d'initialisation

Comme nous l'avons mentionné précédemment, les algorithmes de type EM donnent des résultats fortement dépendants de l'initialisation. Puisqu'ils convergent vers un optimum local, il est évident que selon le point de départ choisi, le résultat sera plus ou moins bon. Pour appréhender ce problème, la technique généralement utilisée est de lancer plusieurs fois l'algorithme d'une position aléatoire, et de ne retenir que la solution fournissant la log-vraisemblance ou log-vraisemblance complétée la plus grande. Bien que pour CEM, ceci soit totalement réalisable, il n'en est pas de même pour EM, qui peut mettre parfois des milliers d'itérations avant de converger.

L'idée est donc de choisir judicieusement le point de départ, afin d'aider EM à converger rapidement vers la bonne solution. Pour ceci, nous pouvons lancer dans un

premier temps CEM ou un autre algorithme, et prendre la partition obtenue pour démarrer EM à partir de celle-ci.

Parmi de nombreuses méthodes d'initialisation de l'algorithme EM, nous avons retenu CEM et eM (EM avec un critère de convergence modifié), en plus de la méthode traditionnelle, partant d'une partition initiale choisie au hasard. Le schéma d'utilisation de ces stratégies est présenté dans la figure 4.1.

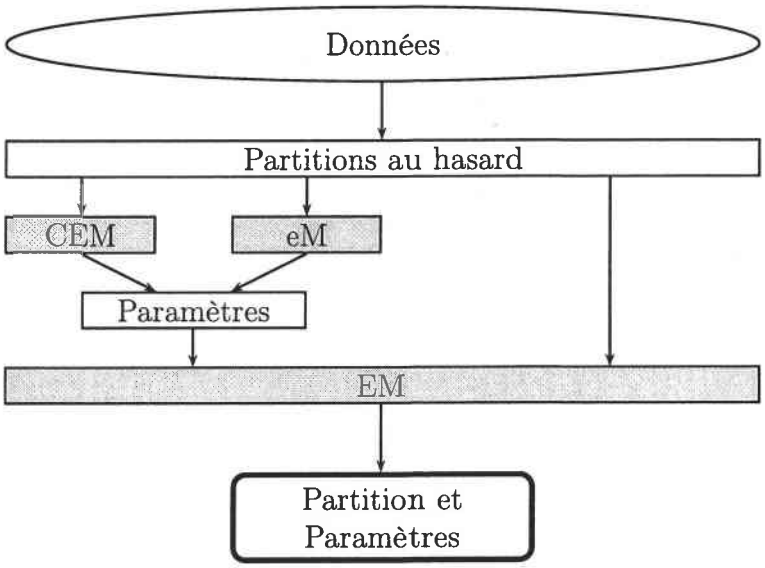



FIG. 4.1 – *Différentes stratégies d'initialisation de EM. On utilise généralement 20 partitions initiales distinctes.*

### 4.1.1 Description

Le gros avantage de CEM pour l'initialisation est le nombre généralement réduit d'itérations qu'il lui faut pour converger. Donc, assez simplement et rapidement, plusieurs essais de CEM peuvent être effectués avant de passer à EM. Mais il est possible de retrouver cette possibilité avec des petits essais de EM, noté eM (Biernacki *et al.*, 2001). Quand on parle de petit essai de EM, on indique le fait que l'on attend pas la convergence de l'algorithme, mais on utilise un autre critère d'arrêt, défini comme suit

$$\frac{L^{(q)} - L^{(q-1)}}{L^{(q)} - L^{(0)}} \leq 10^{-2}$$

$L^{(q)}$ décrivant la vraisemblance à l'itération $q$. Ici, le seuil de $10^{-2}$ a été choisi de façon pragmatique, et qui, sur nos expériences, s'est révélé fournir un nombre d'itérations de eM approximativement égal à celui de CEM. De la même manière que pour CEM, on va donc lancer plusieurs fois eM d'une partition au hasard, suivi par EM, initialisé par la solution donnant la plus grande vraisemblance avec eM.

### 4.1.2 Résultats

Pour illustrer le comportement de ces stratégies, nous les avons appliquées sur 3 tableaux simulés suivant le modèle des classes latentes. Chaque tableau comporte 1000 objets, répartis en 3 classes et décrit par 30 variables (à 3, 4 ou 5 modalités). Les trois tableaux différent de par leur degré de mélange :

| | |
|---|---|
| (+) | classes peu mélangées, |
| (++) | classes moyennement mélangées, |
| (+++) | classes bien mélangées. |

Les résultats sont présentés dans le tableau 4.1. Nous voyons que la stratégie CEM-EM est la plus rapide, tout en fournissant une partition identique à EM (comme eM-EM). Ces stratégies apparaissent intéressantes sur ces exemples, mais nous verrons par la suite leur comportement sur des données réelles.

TAB. 4.1 – *Comparaison de différentes stratégies d'initialisation de l'algorithme EM sur plusieurs tableaux simulés.*

| Données | Stratégie | $L$ | Temps (en s) | Coefficient d'accélération | % de mal-classés |
|---|---|---|---|---|---|
| + | EM | -61646.06 | 219.18 | 1.00 | 4.4 |
| | CEM-EM | -61646.06 | 34.86 | 6.29 | 4.4 |
| | eM-EM | -61646.06 | 110.35 | 1.99 | 4.4 |
| ++ | EM | -62044.00 | 436.97 | 1.00 | 17.7 |
| | CEM-EM | -62044.00 | 64.12 | 6.81 | 17.7 |
| | eM-EM | -62044.00 | 176.56 | 2.47 | 17.7 |
| +++ | EM | -64202.54 | 1342.37 | 1.00 | 24.2 |
| | CEM-EM | -64202.54 | 241.06 | 5.57 | 24.2 |
| | eM-EM | -64202.54 | 389.59 | 3.45 | 24.2 |

## 4.2 Accélération

Nous venons de voir un certain nombre de stratégies pour améliorer l'initialisation de celui-ci, nous allons maintenant étudier différentes manières de rendre directement EM plus rapide. Plusieurs versions ont été faites pour accélérer cet algorithme et beaucoup d'entre elles agissent sur l'étape de maximisation. Ici, comme nous nous intéressons au modèle des classes latentes, l'étape de maximisation ne présente aucune difficulté pour le calcul des paramètres. Nous avons donc choisi d'étudier des versions particulièrement adaptées aux données de grande taille et qui utilisent une étape d'Estimation partielle au lieu d'une étape d'Estimation complète. Ces versions semblent très efficaces pour des mélanges Gaussiens, nous les appliquons ici sur des données qualitatives, et discutons leur comportement. Enfin, nous proposerons une nouvelle variante, plus performante.

### 4.2.1 Méthodes existantes

Il y a plusieurs méthodes alternatives pour accélérer EM dans des situations où la taille des données fait que l'étape E à un coût de calcul trop important (Thiesson *et al.*, 2001). Nous décrivons 2 types de méthodes: celles qui oublient partiellement les statistiques précédentes et celles qui résument ou compressent plusieurs données.

Une version *amnésique* de EM est une méthode qui utilise les statistiques exhaustives calculées comme une moyenne décroissante (exponentiellement) des blocs de données visités récemment. Les variantes amnésiques ou *on-line* de EM ont été proposées par plusieurs auteurs, comme par exemple (Nowlan, 1991) ou (Sato et Ishii, 2001), avec des généralisations dans (Sato, 1999). Ces variantes supposent la plupart du temps que les blocs consistent en des objets seuls, mais peuvent être facilement étendues au cas de blocs de taille arbitraire.

Ces versions amnésiques sont souvent utilisées pour estimer des paramètres variant dans le temps. Cependant, elles peuvent potentiellement être utiles pour des données de très grande taille. Dans de telles situations, il est possible qu'une version amnésique et incrémentale de EM converge plus rapidement qu'une version incrémentale de EM. Cette possibilité provient du fait que les statistiques estimées pour les blocs calculées durant les étapes précédentes contiennent d'assez grandes imprécisions. En oubliant ces statistiques, on peut rapidement améliorer les estimations des paramètres. D'un autre côté, dans leur comparaison limitée des deux versions, les auteurs ont trouvé que ces méthodes ne proposent pas d'amélioration significative par rapport à une version incrémentale de EM dans les cas où plusieurs passages à travers les données sont nécessaires pour la convergence. Cette observation n'est pas surprenante, étant donné que cette version incrémentale est en fait une méthode amnésique qui oublie complètement les statistiques calculées pour un bloc après un passage à travers les données.

Les applications de ce type de méthode à de grandes bases de données requièrent une sélection d'un taux de décroissance. En plus, pour permettre la convergence en moins d'un passage sur toutes les données, un critère de convergence alternatif est requis. Finalement, à l'inverse de IEM, les implémentations naïves ne garantissent pas la convergence à un maximum local de la vraisemblance pour des tableaux de données finis.

Le second type de méthode pour accélérer EM se base sur la compression. Grossièrement, dans une telle méthode, on représente un ensemble de données par une *observation compressée*. Ces méthodes accélèrent EM en mettant à jour (approximativement) les statistiques pour un ensemble de données associées à une observation compressée en calculant les statistiques estimées pour celle-ci. Notons que le choix des objets à condenser n'a pas besoin d'être fixé à l'avance et différents ensembles de telles observations succinctes peuvent être utilisés aux différentes itérations de l'algorithme.

Une telle méthode de compression est celle de Moore (1999), qui montre des résultats exceptionnels d'accélération pour des problèmes de modèle de mélange, avec peu de variables et beaucoup d'objets. Dans l'approche de Moore, on construit d'abord un *multi-resolution kd-tree* (voir figure 4.2) pour les objets dans la table. Avec cette représentation, un cas compressé est un sommet dans le *kd-tree* qui représente tous les objets stockés sous ce sommet. Ainsi, on lance une version de EM dans

laquelle les statistiques exhaustives estimées sont calculées à partir des cas condensés. Le choix des cas compressés (et donc le choix des sommets) est déterminé à nouveau à chaque étape Estimation afin de faire un compromis entre la qualité de l'approximation et le temps de calcul de l'étape E. Malheureusement, les algorithmes *kd-tree* supportent mal un nombre trop important de variables.

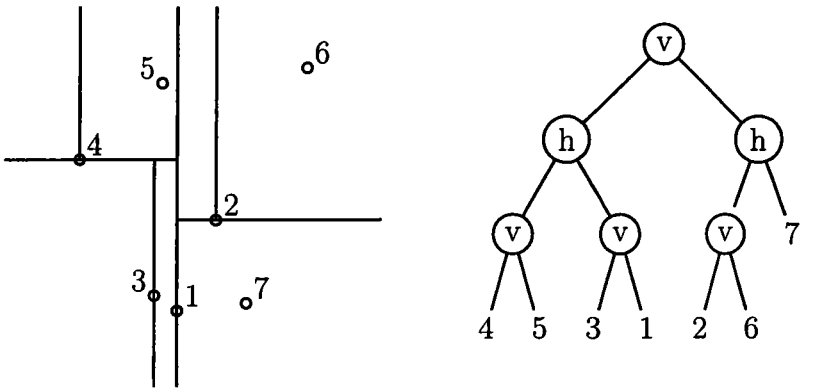



FIG. 4.2 – *Par découpages verticaux (v) et horizontaux (h) successifs, on obtient avec un kd-tree une séparation des données, représentable dans un arbre (comme celui à droite), permettant ainsi de choisir la résolution souhaitée, en fonction de la précision que l'on désire.*

Une méthode relativement proche, qui est adapté à un nombre de variables important, est celle de McCallum *et al.* (2000). Au lieu de partitionner strictement les données dans un *multi-resolution kd-tree*, cette méthode utilisé une mesure de similarité grossière pour premièrement diviser les données en *calottes*, qui sont des sous-ensembles qui peuvent se recouvrir. Durant une itération de l'algorithme EM, les composants du modèle de mélange sont associés à une ou plusieurs calottes se chevauchant, et la mise à jour du modèle de mélange est seulement influencée par les sous-ensembles de cas qui appartiennent aux calottes associées à cette classe. Dans une variante de cette méthode, la mise à jour est de plus influencée par la moyenne de chaque calotte non associée à la classe. L'accélération et la précision de ces méthodes sont très liées au recouvrement des calottes. L'accélération résultante sera significative si les données permettent une séparation très nette des sous-ensembles, et que la méthode identifie une telle division. Bien que non expliqué dans McCallum *et al.* (2000), nous pensons malgré tout que soit l'accélération, soit la précision des estimations finales décroiront quand les objets sont générés à partir de classes qui se recouvrent. Pour des données simulées avec des degrés de recouvrement variants, Moore (1999) indique une baisse de l'accélération quand le recouvrement augmente.

Un type différent de méthode de compression est décrit dans Bradley *et al.* (1998). Leur méthode découpe les données en blocs de cas, et travaille à travers ces blocs, en ajoutant un nouveau bloc dans un *ensemble de travail* à chaque itération. Durant une itération, l'algorithme EM standard est utilisé jusqu'à la convergence, et les données qui sont proches selon la distance de Mahalanobis sont compressées. Une fois les objets condensés, ils ne peuvent plus être séparés. Ainsi, cette approche ne garantit pas la convergence à un maximum local de la fonction de vraisemblance.

Nous allons nous intéresser maintenant à quelques uns de ces algorithmes d'ac-

célération. Le premier sera une version incrémentale de EM, séparant les données en blocs. Ensuite, nous présenterons deux méthodes qui sont des versions amnésiques de EM, LEM et SpEM, ainsi que deux variantes, LEM-D et LSpEM.

### 4.2.2 Incremental EM (IEM)

L'algorithme Incremental EM, ou IEM, (Neal et Hinton, 1998) qui est une variante de EM, est destiné à réduire le temps de calcul en réalisant des étapes Estimation partielles. Soit $y = y_1, \ldots, y_B$ une partition des données en $B$ blocs disjoints ($B$ est proposé par l'utilisateur, ou bien est déduit par rapport à la taille des blocs demandés par l'utilisateur, en pourcentage de la taille initiale), comme présenté dans la figure 4.3. L'algorithme IEM parcourt tous les blocs de façon cyclique. A chaque itération, on met à jour les probabilités a posteriori d'un bloc dans l'étape Estimation. Ci-dessous, on décrit plus en détail la $n$ième itération :

- **Estimation** : Dans cette étape, on retient un bloc $y_b$ et on met à jour les probabilités a posteriori $t_{ik}^{(q)}$ pour tous les objets appartenant aux bloc $y_b$, quant aux autres objets (appartenant aux autres blocs) nous avons $t_{ik}^{(q)} = t_{ik}^{(q-1)}$. L'espérance conditionnelle associée au bloc $b$ notée $Q_b$ est mise à jour, quant à celles associées aux autres blocs elles restent inchangées. Autrement dit la quantité globale qu'on cherchera à maximiser dans l'étape maximisation peut s'écrire :
$$Q(\theta|\theta^{(q)}) = Q(\theta|\theta^{(q-1)}) - Q_b(\theta|\theta^{(q-1)}) + Q_b(\theta|\theta^{(q)}) \quad (4.1)$$
- **Maximisation** : On cherche comme dans l'algorithme EM classique, le paramètre $\theta^{(q+1)}$ qui maximise $Q(\theta|\theta^{(q)})$.

De cette façon, chaque objet $\mathbf{x}_i$ est visité après les $B$ étapes d'estimation partielles. Une approximation de la log-vraisemblance ne décroît pas à chaque itération. La justification théorique de cet algorithme a été faite par Neal et Hinton (Neal et Hinton, 1998). Récemment, nous avons étudié son comportement avec le modèle de mélange de Bernoulli (Jollois *et al.*, 2003). Enfin, notons que lorsque $B = 1$, IEM se réduit à EM.

En appliquant l'algorithme IEM sur des données simulées, nous avons étudié plusieurs aspects dont le choix du nombre de blocs et la taille des données. Pour illustrer ces différents aspects, nous avons choisi de présenter quelques résultats obtenus à partir de 3 tableaux simulés de taille différente ($5000 \times 5$, $10000 \times 10$, $10000 \times 50$), de données binaires.

Dans nos expériences numériques, on initialise les algorithmes EM et IEM à partir des mêmes paramètres choisis au hasard $\theta^{(0)}$. A partir des résultats présentés dans Tab. 4.2, nous constatons pour les deux premiers tableaux de données, qu'un trop petit nombre n'apportera pas forcément une accélération de l'algorithme très importante, alors qu'un trop grand nombre demandera plus de calculs intermédiaires, et finalement n'accélérera pas l'algorithme. Aussi, nous pouvons relever que IEM accélère nettement l'algorithme, au détriment parfois de la qualité des estimations obtenues (voir la log-vraisemblance $L$ arrondie). D'où l'intérêt de proposer un nombre approprié de blocs en fonction de la taille de l'échantillon. Pour le dernier cas, avec $n = 10000$ et $d = 50$, où le rapport entre $n$ et $d$ est égal à 200 alors que dans les cas précédents, le rapport est égal à 1000, IEM apparaît toujours moins

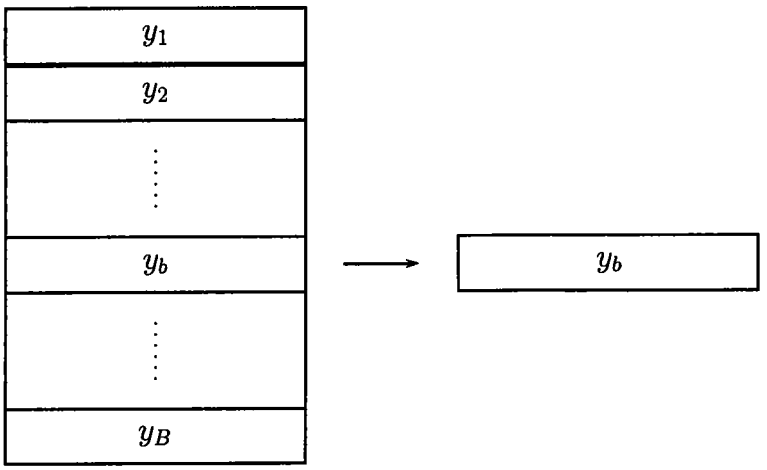



FIG. 4.3 – *Séparation des objets en B blocs pour IEM. A l'étape t, on utilise le bloc correspondant $y_b$, avec $b = t \bmod B$.*

rapide que EM. Ce qui laisse à penser que l'utilisation de IEM n'est profitable que dans certaines situations.

TAB. 4.2 – *Temps mis (en secondes) par IEM sur des données simulées, selon le nombre de blocs demandé (1 seul bloc correspond à EM standard).*

| | 5000x5 | | 10000x10 | | 10000x50 | |
|---|---|---|---|---|---|---|
| $B$ | temps | $L$ | temps | $L$ | temps | $L$ |
| 1 | 307 | -16164 | 706 | -62356 | 989 | -291943 |
| 2 | 227 | -16163 | 619 | -62356 | 1005 | -291943 |
| 4 | 172 | -16163 | 564 | -62356 | 1122 | -291943 |
| 8 | 50 | -16163 | 672 | -62356 | 1388 | -291943 |
| 16 | 29 | -16163 | 599 | -62356 | 2086 | -291943 |
| 32 | 13 | -16473 | 187 | -62357 | 3358 | -291943 |
| 64 | 17 | -16473 | 52 | -65591 | 5216 | -291943 |
| 128 | 23 | -16472 | 69 | -65593 | 10291 | -291943 |
| 256 | 40 | -16474 | 97 | -65587 | 12525 | -291943 |

D'autres études ont largement montrées l'intérêt de IEM (Neal et Hinton, 1998; McLachlan et Peel, 1998) dans le cas de données quantitatives. Malheureusement, les simulations ont souvent été réalisés avec $d = 2$. Nos premiers résultats sur des données qualitatives nous laissent à penser que IEM n'est que peu performant dans ce cas.

### 4.2.3 Sparse EM

L'algorithme Sparse EM, ou SpEM, introduit par (Neal et Hinton, 1998), minimise le coût de l'étape Estimation en choisissant les calculs à effectuer à partir d'un seuil, contrairement à IEM. SpEM cherche donc les probabilités a posteriori très petites (inférieur à un certain seuil), et ne les recalcule plus pendant un certain nombre

d'itérations. L'idée est qu'un objet qui a une faible probabilité d'appartenance à une classe a peu de chance de la voir devenir grande en une seule itération. Et donc, ces probabilités sont bloqués pendant un certain temps, puis recalculer au cours d'une itération standard de EM, comme présenté dans la figure 4.4. L'algorithme peut être défini comme suit

- **Estimation**

  Étape standard : Calculer les probabilités a posteriori. Identifier $y^i_{sparse}$ comme l'ensemble des classes à ignorer durant les étapes *sparse*, pour chaque objet $i$.

  Étape *sparse*: Dans cette étape, on calcule les probabilités a posteriori $t^{(q)}_{ik}$ pour toutes les classes appartenant aux blocs $y^i_{\overline{sparse}}$. Seule l'espérance conditionnelle associée au bloc $y_{\overline{sparse}}$ notée $Q_{\overline{sparse}}$ est mise à jour. Autrement dit la quantité globale qu'on cherchera à maximiser dans l'étape maximisation est celle de l'équation 4.1 (en remplaçant $Q_b$ par $Q_{\overline{sparse}}$).

- **Maximisation** : On cherche comme dans l'algorithme EM classique, le paramètre $\theta^{(q+1)}$ qui maximise $Q(\theta|\theta^{(q)})$.

L'algorithme débute sur une itération standard de EM, puis effectue un nombre $n_{sparse}$ d'itération avec une étape Estimation *sparse*, pour revenir ensuite à une itération standard, et ainsi de suite jusqu'à la convergence.

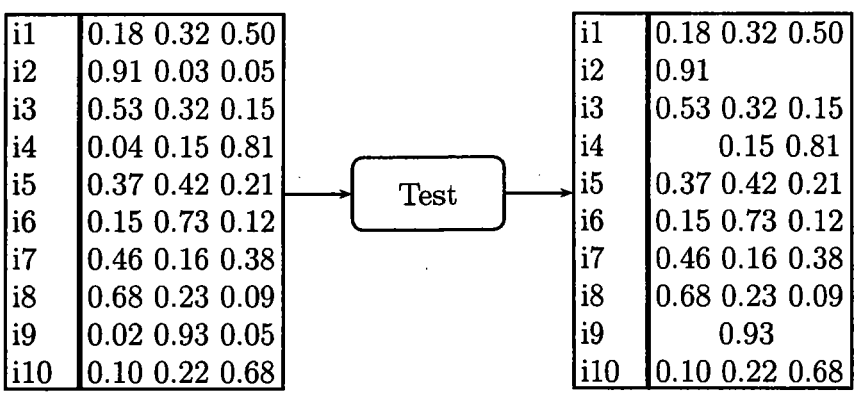



FIG. 4.4 – *Choix des probabilités a posteriori $t_{ik}$ à conservées ou non, dans l'algorithme SpEM (ici avec un seuil à 0.05). Celles ayant une valeur inférieure à ce seuil seront fixées, et on ne les calculera pas pendant un certain nombre d'itérations.*

### 4.2.4 Lazy EM

Ayant le même objectif que IEM et SpEM, l'algorithme Lazy EM, ou LEM, (Thiesson *et al.*, 2001) cherche donc à réduire le temps de l'étape Estimation. Pour ceci, il cherche à identifier régulièrement les objets importants, et à porter son attention sur ceux-ci pour plusieurs itérations. Un objet est considéré comme important si le changement de la probabilité $t_{ik}$ entre deux itérations successives est grande. Notons $y_{\overline{lazy}}$ cet ensemble de cas significatif, et $y_{lazy}$ l'ensemble de cas restants. Suivant un déroulement similaire à SpEM, chaque itération requiert soit une étape

Estimation standard, soit une étape Estimation *lazy*, suivi ensuite par une étape Maximisation standard. L'étape complète calcule pour tous les objets les probabilités a posteriori. De plus, elle établit la liste des objets importants, comme l'indique la figure 4.5. Une étape *lazy* ne met à jour qu'une partie des probabilités a posteriori.

- **Estimation**

  Étape standard : Calculer les probabilités a posteriori. Identifier $y_{lazy}$ comme l'ensemble d'objets à ignorer durant les étapes *lazy*.

  Étape *lazy* : Dans cette étape, on calcule les probabilités a posteriori $t_{ik}^{(q)}$ pour tous les objets appartenant au bloc $y_{\overline{lazy}}$, quant aux autres objets (appartenant à $y_{lazy}$), nous avons $t_{ik}^{(q)} = t_{ik}^{(q-1)}$. Seule l'espérance conditionnelle associée au bloc $y_{\overline{lazy}}$ notée $Q_{\overline{lazy}}$ est mise à jour. Autrement dit la quantité globale qu'on cherchera à maximiser dans l'étape maximisation est celle de l'équation 4.1 (en remplaçant $Q_b$ par $Q_{\overline{lazy}}$).
- **Maximisation** : On cherche comme dans l'algorithme EM classique, le paramètre $\theta^{(q+1)}$ qui maximise $Q(\theta|\theta^{(q)})$.

Le déroulement de LEM est le même que SpeM, avec une itération standard suivie de $n_{lazy}$ itérations dites *lazy*. Ce schéma est répété jusqu'à la convergence de l'algorithme.

La viabilité de l'algorithme LEM réside partiellement dans l'idée que toutes les données ne sont pas d'importance égale. Elle dépend aussi du coût de calcul pour déterminer l'importance de chaque objet et du coût de stockage pour garder cette information. Pour les modèles de mélange, ces deux coûts peuvent être grandement réduits, voire simplement supprimés pour le coût de stockage, grâce à un critère d'importance. L'idée derrière ce critère est la suivante : si un objet a une forte probabilité d'appartenir à une classe, il n'est pas approprié de l'assigner à une autre. Et s'il le fallait, cela ne serait pas soudainement mais plutôt progressivement. Ainsi, nous supposons que les objets qui ne sont pas fortement liés à une classe contribuent le plus à l'évolution des paramètres. Et donc, les objets sont considérés comme importants s'ils ont toutes leur probabilités d'appartenance $t_{ik}$ inférieures à un certain seuil.

Due à (Neal et Hinton, 1998), la convergence de l'algorithme est justifiée théoriquement et est applicable pour chaque découpage arbitraire des objets, du moment qu'on visite régulièrement tous les cas.

### 4.2.5 Variation autour du même thème, LEM modifié

Après réflexion, il nous est apparu que l'on pouvait choisir de différentes manières les objets à garder ou non. Ainsi, nous avons développé cet ensemble de variantes, toutes dérivées de LEM. Après plusieurs essais, nous avons choisi de ne garder que la dernière, qui est la plus intéressante de toutes. Voici présentées les stratégies étudiées :

- La première idée consiste à évincer les objets dont la probabilité a posteriori maximale est supérieure à la proportion de la classe correspondante, multipliée par un certain facteur.

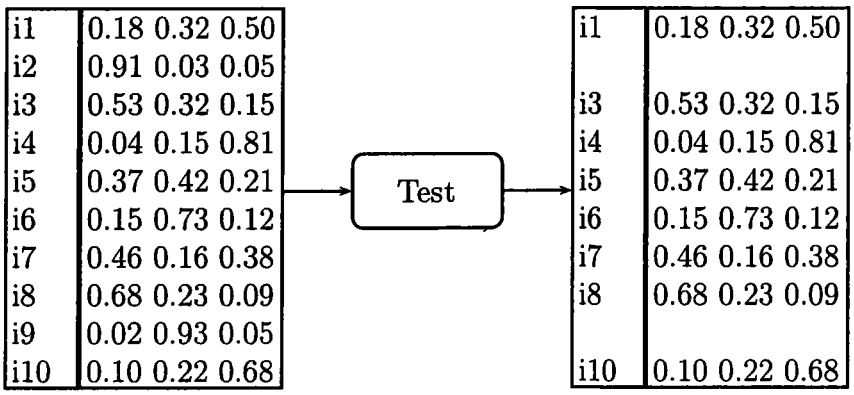



FIG. 4.5 – *Choix des objets en fonction de leur probabilités a posteriori $t_{ik}$, dans l'algorithme LEM (ici avec un seuil à 0.90). Ceux ayant une valeur supérieure à ce seuil seront fixés, et on ne calculera pas leurs probabilités pendant un certain nombre d'itérations.*

- Puis, en se basant sur les tests avec intervalle de confiance, nous avons choisi d'effectuer le test consistant à évaluer si la probabilité maximale d'un objet était significativement supérieure ou non à la proportion de la classe. Si cela est le cas, alors, l'objet est mis de côté.
- Reprenant les travaux de (Thiesson *et al.*, 2001), nous avons choisi de mettre de côté un certain pourcentage $\alpha$ d'objets, ceux qui ont les plus grosses probabilités a posteriori.
- Avec la même idée que précédemment, nous avons choisi de prendre un pourcentage $\alpha$ des objets en fonction de leur différence moyenne des probabilités $t_{ik}$ entre deux itérations. Ceux présentant les plus petites sont ainsi écartés.
- Toujours avec les différences, mais en se basant sur le principe d'un seuil, nous avons choisi de faire le test suivant: si la différence moyenne est inférieur à un certain seuil, alors l'objet est mis à part. C'est cette stratégie qui nous est apparue la meilleure et la plus intéressante au vue de tous les résultats obtenus, non détaillés ici. Nous la détaillons par la suite.

### 4.2.6 Lazy EM, basé sur les différences (LEM-D)

L'idée de base de (Thiesson *et al.*, 2001) est d'écarter un certain nombre d'objets que l'on peut considérer comme peu important dans les calculs. Cette notion d'importance peut être rapportée à l'évolution des probabilités a posteriori. En effet, si un objet ne montre pas d'évolution importante entre deux étapes, c'est qu'il est a priori stable et donc, il a de fortes chances de le rester un long moment. Dans ce cas, on prend la décision de l'écarter des calculs et de ne plus le prendre en compte pendant un certain nombre d'itérations. Au contraire, si son évolution est significative, il est intéressant de le garder dans les calculs.

A partir de là, un nouveau problème se pose à nous. Comment déterminer si un objet a évolué significativement ou non? Pour ceci, nous avons choisi de mesurer les différences entre les probabilités a posteriori avant et après l'étape Estimation

standard, où on remet à jour tous les $t_{ik}$ de tous les objets. Plus précisément, nous comparons la moyenne des valeurs absolues de ces différences pour chaque classe avec un seuil $\alpha$ :

$$\frac{\sum_{k=1}^{s} |t_{ik}^{(q)} - t_{ik}^{(q-1)}|}{s} < \alpha$$

Nous avons testé plusieurs valeurs de ce seuil. Il est apparu que si celui-ci est grand (généralement au-dessus de 0.0250), l'algorithme met de côté tous les objets, et s'arrête donc automatiquement. Pour cette raison, dans nos expériences, les tests seront réalisés avec un seuil inférieur à 0.0250.

### 4.2.7 Lazy Sparse EM, une combinaison de deux méthodes

Nous avons ici combiné les deux idées de LEM et SpEM, dans l'algorithme lazy sparse EM, noté LSpEM. Les probabilités supérieures à un certain seuil *lazy* et celles inférieures à un autre seuil *sparse* sont mises de côté, et figées pendant quelques itérations, de la même manière pour Sparse EM. L'idée ici est de se dire que les fortes probabilités, ainsi que les faibles, n'évolueront sûrement pas de manière significative à chaque itération. Il est donc intéressant de les figer pendant un certain nombre d'itérations, afin de se concentrer sur les probabilités moyennes, et ainsi de pouvoir accélérer l'algorithme, grâce au gain de calcul effectué. Lors de l'étape général, toutes les probabilités a posteriori sont calculées de nouveau, et on choisit par la suite celles qui seront fixées.

- **Estimation**
  - Étape standard : Calculer les probabilités a posteriori. Identifier $y_{lsp}^{i}$ comme l'ensemble des classes à ignoré durant les étapes *lazy sparse*, pour chaque objet $i$, en fonction des deux seuils.
  - Étape *lazy sparse* : Dans cette étape, on calcule les probabilités a posteriori $t_{ik}^{(n)}$ pour toutes les classes appartenant aux blocs $y_{l\bar{s}p}^{i}$. Seule l'espérance conditionnelle associée au bloc $y_{l\bar{s}p}$ notée $Q_{l\bar{s}p}$ est mise à jour. Autrement dit la quantité globale qu'on cherchera à maximiser dans l'étape maximisation est celle de l'équation 4.1 (en remplaçant $Q_b$ par $Q_{l\bar{s}p}$).
- **Maximisation** : On cherche comme dans l'algorithme EM classique, le paramètre $\theta^{(n+1)}$ qui maximise $Q(\theta|\theta^{(n)})$.

Pour cette méthode, le choix des paramètres est complexe, car deux seuils sont à prendre en compte. Nous avons choisi les même valeurs que pour LEM pour le seuil *lazy*, et les mêmes valeurs que SpEM pour le seuil *sparse*.

### 4.2.8 Étude des algorithmes de type *seuil* : LEM, LEM-D, SpEM et LSpEM

Pour avoir une première idée du comportement de ces algorithmes, nous allons observer leur comportement, en les comparant de plus à EM. Pour ceci, nous avons simulé 4 tableaux, avec les paramètres suivants :

- Sim1 : $s = 3$, $n = 10000$, $d = 10$, classes séparées.

- Sim2 : $s = 3$, $n = 10000$, $d = 10$, classes mélangées.
- Sim3 : $s = 7$, $n = 10000$, $d = 10$, classes séparées.
- Sim4 : $s = 7$, $n = 10000$, $d = 10$, classes mélangées.

Pour les méthodes que nous testons ici, nous avons choisi de prendre les différents paramètres suivant :

| Méthode | Itération | Seuils |
|---|---|---|
| LEM | 1 à 8 | 0.50, 0.60, 0.70, 0.80, 0.90, 0.95 et 0.99 |
| LEM-D | 1 à 8 | 0.0001, 0.0005, 0.001, 0.005, 0.01, 0.015 et 0.02 |
| SpEM | 1 à 8 | 0.001, 0.005, 0.010 et 0.050 |
| LSpEM | 1 à 8 | 0.50, 0.60, 0.70, 0.80, 0.90, 0.95 et 0.99 et 0.001, 0.005, 0.010 et 0.050 |

Dans les tableaux 4.3, 4.4, 4.5 et 4.6, nous présentons pour chaque tableau simulé (respectivement Sim1, Sim2, Sim3 et Sim4) le meilleur et le moins bon résultat pour la log-vraisemblance $L$, le coefficient d'accélération et le pourcentage de données mal-classées, avec le paramètrage permettant d'obtenir ces résultats.

TAB. 4.3 – *Le meilleur (+) et le moins bon (-) résultats de différents algorithmes accélérant la convergence de EM, avec le paramètrage correspondant, pour Sim1.*

| | Méthode | + | | - | |
|---|---|---|---|---|---|
| | | Résultat | Paramètres | Résultat | Paramètres |
| $L$ | EM | -42875.69 | - | -42875.69 | - |
| | LEM | -42875.69 | 1, 0.6 | -42927.27 | 2, 0.5 |
| | LEM-D | -42875.69 | 1, 0.0001 | -42880.94 | 7, 0.015 |
| | SpEM | -42875.69 | 1, 0.1 | -42875.71 | 7, 0.1 |
| | LSpEM | -42875.69 | 1, 0.5, 0.1 | -42926.90 | 4, 0.5, 0.1 |
| Coefficient d'accélération | EM | 1.00 | - | 1.00 | - |
| | LEM | 3.62 | 4, 0.5 | 1.16 | 3, 0.9 |
| | LEM-D | 2.23 | 1, 0.015 | 0.76 | 8, 0.02 |
| | SpEM | 1.63 | 7, 0.1 | 0.95 | 3, 0.1 |
| | LSpEM | 2.63 | 4, 0.5, 0.05 | 0.99 | 2, 0.5, 0.001 |
| Pourcentage de mal-classés | EM | 1.17 | - | 1.17 | - |
| | LEM | 1.17 | 1, 0.6 | 2.41 | 2, 0.5 |
| | LEM-D | 1.17 | 1, 0.0001 | 1.20 | 7, 0.015 |
| | SpEM | 1.17 | 1, 0.1 | 1.17 | 1, 0.1 |
| | LSpEM | 1.17 | 1, 0.5, 0.1 | 2.41 | 4, 0.5, 0.1 |

Pour Sim1, SpEM donne toujours les mêmes résultats que EM, mais n'accélère pas toujours l'algorithme, contrairement à LEM, qui lui fournit parfois des résultats mauvais. LSpEM est dans le même cas de figure que LEM, il accélère souvent l'algorithme, mais peut produire des résultats plutôt mauvais. LEM-D, lui, arrive de temps en temps à aller bien plus vite que EM, tout en fournissant toujours une solution plutôt correct.

Pour Sim2, les 4 méthodes fournissent toujours une solution proche de celle de EM. Par contre, on voit que LEM-D accélère toujours l'algorithme, contrairement aux 3 autres, SpEM n'étant que très peu rapide.

TAB. 4.4 – *Le meilleur (+) et le moins bon (-) résultats de différents algorithmes accélérant la convergence de EM, avec le paramètrage correspondant, pour Sim2.*

| | Méthode | + | | - | |
|---|---|---|---|---|---|
| | | Résultat | Paramètres | Résultat | Paramètres |
| $L$ | EM | -63217.91 | - | -63217.91 | - |
| | LEM | -63217.91 | 1,0.5 | -63240.51 | 7, 0.5 |
| | LEM-D | -63217.91 | 1,0.0001 | -63225.97 | 7, 0.005 |
| | SpEM | -63217.91 | 1,0.1 | -63217.91 | 1, 0.1 |
| | LSpEM | -63217.91 | 1,0.5, 0.1 | -63240.33 | 6, 0.5, 0.1 |
| Coefficient | EM | 1.00 | - | 1.00 | - |
| d'accélération | LEM | 7.05 | 7, 0.5 | 0.73 | 8, 0.7 |
| | LEM-D | 7.54 | 1, 0.02 | 1.40 | 4, 0.0001 |
| | SpEM | 1.14 | 1, 0.01 | 0.72 | 8, 0.1 |
| | LSpEM | 5.93 | 8, 0.5, 0.1 | 0.32 | 7, 0.7, 0.01 |
| Pourcentage | EM | 15.33 | - | 15.33 | - |
| de mal-classés | LEM | 15.31 | 7, 0.6 | 15.48 | 6, 0.5 |
| | LEM-D | 15.29 | 6, 0.02 | 15.48 | 1, 0.02 |
| | SpEM | 15.33 | 1, 0.1 | 15.33 | 1, 0.1 |
| | LSpEM | 15.31 | 7, 0.6, 0.1 | 15.46 | 6, 0.5, 0.1 |

TAB. 4.5 – *Le meilleur (+) et le moins bon (-) résultats de différents algorithmes accélérant la convergence de EM, avec le paramètrage correspondant, pour Sim3.*

| | Méthode | + | | - | |
|---|---|---|---|---|---|
| | | Résultat | Paramètres | Résultat | Paramètres |
| $L$ | EM | -45367.68 | - | -45367.68 | - |
| | LEM | -45367.68 | 1, 0.5 | -45544.21 | 8, 0.5 |
| | LEM-D | -45367.68 | 1, 0.0001 | -46983.79 | 5, 0.01 |
| | SpEM | -45367.68 | 1, 0.1 | -45368.76 | 8, 0.1 |
| | LSpEM | -45367.68 | 1, 0.5, 0.1 | -45404.58 | 8, 0.5, 0.1 |
| Coefficient | EM | 1.00 | - | 1.00 | |
| d'accélération | LEM | 1.62 | 6, 0.7 | 0.81 | 8, 0.9 |
| | LEM-D | 2.41 | 1, 0.02 | 0.53 | 6, 0.01 |
| | SpEM | 1.37 | 4, 0.01 | 0.75 | 5, 0.1 |
| | LSpEM | 1.57 | 7, 0.7, 0.1 | 0.59 | 5, 0.5, 0.001 |
| Pourcentage | EM | 4.86 | - | 4.86 | - |
| de mal-classés | LEM | 4.83 | 3, 0.5 | 6.76 | 8, 0.5 |
| | LEM-D | 4.84 | 1, 0.02 | 23.28 | 2, 0.02 |
| | SpEM | 4.86 | 1, 0.1 | 4.86 | 1, 0.1 |
| | LSpEM | 4.84 | 7, 0.5, 0.1 | 4.86 | 1, 0.5, 0.1 |

Pour Sim3, toutes les méthodes n'ont pas un comportement régulier en terme d'accélération. Parfois, elles vont plus vite que EM, parfois plus lentement. SpEM est toujours aussi stable et donne toujours de bons résultats, ainsi que LSpEM pour ce tableau. LEM-D, et dans une moindre mesure LEM, fournissent parfois des partitions très éloignées de celles simulées.

TAB. 4.6 – *Le meilleur (+) et le moins bon (-) résultats de différents algorithmes accélérant la convergence de EM, avec le paramètrage correspondant, pour Sim4.*

| | Méthode | + | | - | |
|---|---|---|---|---|---|
| | | Résultat | Paramètres | Résultat | Paramètres |
| $L$ | EM | -56891.18 | - | -56891.18 | - |
| | LEM | -56891.18 | 1, 0.5 | -56915.19 | 8, 0.5 |
| | LEM-D | -56891.18 | 1, 0.0001 | -57537.11 | 1, 0.015 |
| | SpEM | -56891.18 | 1, 0.1 | -56891.18 | 1, 0.1 |
| | LSpEM | -56891.18 | 1, 0.5, 0.1 | -56905.26 | 8, 0.5, 0.1 |
| Coefficient | EM | 1.00 | - | 1.00 | - |
| d'accélération | LEM | 1.44 | 5, 0.5 | 0.64 | 8, 0.8 |
| | LEM-D | 4.00 | 1, 0.015 | 1.20 | 4, 0.015 |
| | SpEM | 1.19 | 2, 0.01 | 0.55 | 8, 0.1 |
| | LSpEM | 1.18 | 2, 0.95, 0.01 | 0.42 | 7, 0.5, 0.001 |
| Pourcentage | EM | 15.22 | - | 15.22 | - |
| de mal-classés | LEM | 15.20 | 5, 0.6 | 15.25 | 8, 0.6 |
| | LEM-D | 15.15 | 4, 0.015 | 31.29 | 1, 0.015 |
| | SpEM | 15.22 | 1, 0.1 | 15.22 | 1, 0.1 |
| | LSpEM | 15.20 | 7, 0.5, 0.1 | 15.23 | 4, 0.5, 0.1 |

Ici, pour Sim4, seul LEM-D va toujours plus vite que EM. Par contre, il fournit parfois une très mauvais partition. Les autres méthodes sont très stables en terme de résultats, mais ne montrent pas d'intérêt flagrant pour leur accélération, très modique.

Cette première étude nous montre les difficultés à choisir les paramètres des algorithmes (taille des blocs, seuil, nombre d'itérations). C'est pour cette raison que nous avons préféré écarter LSpEM dans l'étude suivante, du fait du grand nombre de possibilité de paramètrage, et de ses performances moyennes en terme d'accélération, comparément aux autres méthodes étudiées ici.

### 4.2.9 Comparaison des algorithmes EM, IEM, LEM, LEM-D et SpEM

Afin de comparer efficacement les 4 différentes méthodes d'accélération de EM, nous avons lancé tous ces algorithmes sur des données simulées. Pour ceci, nous avons choisi de créer 10 tableaux avec les mêmes paramètres: $n = 5000$, $d = 10$, $p = \{0.5,0.2,0.3\}$, $c^j = 3, \forall j = 1,\ldots,d$. Comme pour le chapitre précédent, en choisissant les paramètres adéquat via des simulations de Monte-Carlo, nous avons simulé des données peu mélangées (+), moyennement mélangées (++) et très mélangées (+++). Dans le tableau suivant, nous récapitulons les paramètres choisis

TAB. 4.7 – *Résultats moyens pour l'algorithme EM, sur les différents degrés de mélange simulés.*

| Degré de mélange | $L$ | Coefficient d'accélération | Pourcentage de mal-classés |
|---|---|---|---|
| + | -70245.19 | 275.62 | 4.54 |
| ++ | -81915.25 | 476.15 | 13.21 |
| +++ | -85893.79 | 612.97 | 19.17 |

pour chacune des méthodes utilisées.

| Méthode | Paramètres | |
|---|---|---|
| | Taille des blocs | |
| IEM | 0.5, 1, 2.5, 5, 10, 25 et 50 % | |
| | Itération | Seuils |
| SpEM | 1, 2, 3 et 4 | 0.001, 0.005, 0.010 et 0.050 |
| LEM | 1, 2, 3 et 4 | 0.50, 0.60, 0.70, 0.80, 0.90, 0.95 et 0.99 |
| LEM-D | 1, 2, 3 et 4 | 0.001, 0.005, 0.010, 0.015 et 0.020 |

Dans le tableau 4.7, nous reportons les résultats moyens obtenus avec l'algorithme EM. Ensuite, dans les tableaux 4.8, 4.9 et 4.10, nous présentons les meilleurs résultats moyens de chaque méthode accélératrice pour les données peu mélangées (+), moyennement mélangées (++) et très mélangées (+++), avec les paramètres correspondant.

Puis, dans les figures 4.6, 4.7 et 4.8, nous décrivons le comportement moyen de ces algorithmes, en fonction de leur paramètres. Nous pouvons en ressortir les différents points suivants :

- Notre méthode LEM-D est toujours la plus rapide. Même si parfois, elle fournit de moins bon résultats que EM, ceux-ci n'en sont pas très éloignés.
- Les trois autres méthodes IEM, SpEM et LEM vont parfois moins vite que EM. Leur intérêt s'en retrouve donc amoindri, car ils obtiennent eux aussi parfois de moins bons résultats que EM.
- IEM semble être plus performant avec 2 blocs.
- SpEM va généralement le plus vite pour un petit seuil (proche de 0.001) et un petit nombre d'itérations *sparse*.
- LEM est plus rapide lorsque le seuil est grand (vers 0.99), mais la partition ainsi obtenue n'est pas très profitable.
- LEM-D a un comportement variant, et il est difficile d'établir une règle. Une stratégie utilisant plusieurs seuils est donc envisagée.

## 4.3 Conclusion

Tout d'abord, nous avons donc décrit ici plusieurs stratégies d'initialisation. Nous avons comparé trois stratégies différentes. La première est la stratégie standard, consistant à prendre une partition de départ au hasard. Les deux autres se basent

TAB. 4.8 – *Meilleur (+) et moins bon (-) résultats pour chaque méthode d'accélération de l'algorithme EM, sur la moyenne des 10 tableaux simulés pour le mélange (+).*

| | Méthode | + | | - | |
|---|---|---|---|---|---|
| | | Résultat | Paramètres | Résultats | Paramètres |
| $L$ | IEM | -70245.04 | 1.00 | -70279.88 | 0.50 |
| | SpEM | -70245.19 | tous | -70245.19 | tous |
| | LEM | -70245.19 | presque tous | -70245.21 | 0.5 et 4 |
| | LEM-D | -70245.59 | 0.005 et 4 | -70247.64 | 0.020 et 4 |
| Coefficient d'accélération | IEM | 1.82 | 50.00 | 0.16 | 0.50 |
| | SpEM | 1.42 | 0.01 et 2 | 1.00 | 0.05 et 4 |
| | LEM | 1.56 | 0.99 et 2 | 0.83 | 0.6 et 4 |
| | LEM-D | 3.14 | 0.010 et 2 | 2.95 | 0.020 et 3 |
| Pourcentage de mal-classés | IEM | 4.54 | 10.00 | 5.27 | 0.50 |
| | SpEM | 4.54 | tous | 4.54 | tous |
| | LEM | 4.54 | tous | 4.54 | tous |
| | LEM-D | 4.61 | 0.005 et 3 | 4.82 | 0.020 et 4 |

TAB. 4.9 – *Meilleur (+) et moins bon (-) résultats pour chaque méthode d'accélération de l'algorithme EM, sur la moyenne des 10 tableaux simulés pour le mélange (++).*

| | Méthode | + | | - | |
|---|---|---|---|---|---|
| | | Résultat | Paramètres | Résultats | Paramètres |
| $L$ | IEM | -81915.11 | 2.50 | -82004.18 | 0.50 |
| | SpEM | -81915.25 | tous | -81915.25 | tous |
| | LEM | -81915.25 | presque tous | -81915.26 | 0.50 et 4 |
| | LEM-D | -81918.52 | 0.001 et 1 | -81925.09 | 0.020 et 4 |
| Coefficient d'accélération | IEM | 2.10 | 50.0 | 0.28 | 0.50 |
| | SpEM | 1.25 | 0.005 et 2 | 1.02 | 0.050 et 4 |
| | LEM | 1.38 | 0.99 et 1 | 0.69 | 0.60 et 4 |
| | LEM-D | 4.25 | 0.005 et 2 | 3.94 | 0.020 et 3 |
| Pourcentage de mal-classés | IEM | 13.23 | 50.0 | 15.38 | 0.50 |
| | SpEM | 13.21 | tous | 13.21 | tous |
| | LEM | 13.21 | presque tous | 13.23 | 0.50 et 4 |
| | LEM-D | 13.93 | 0.001 et 1 | 14.59 | 0.020 et 4 |

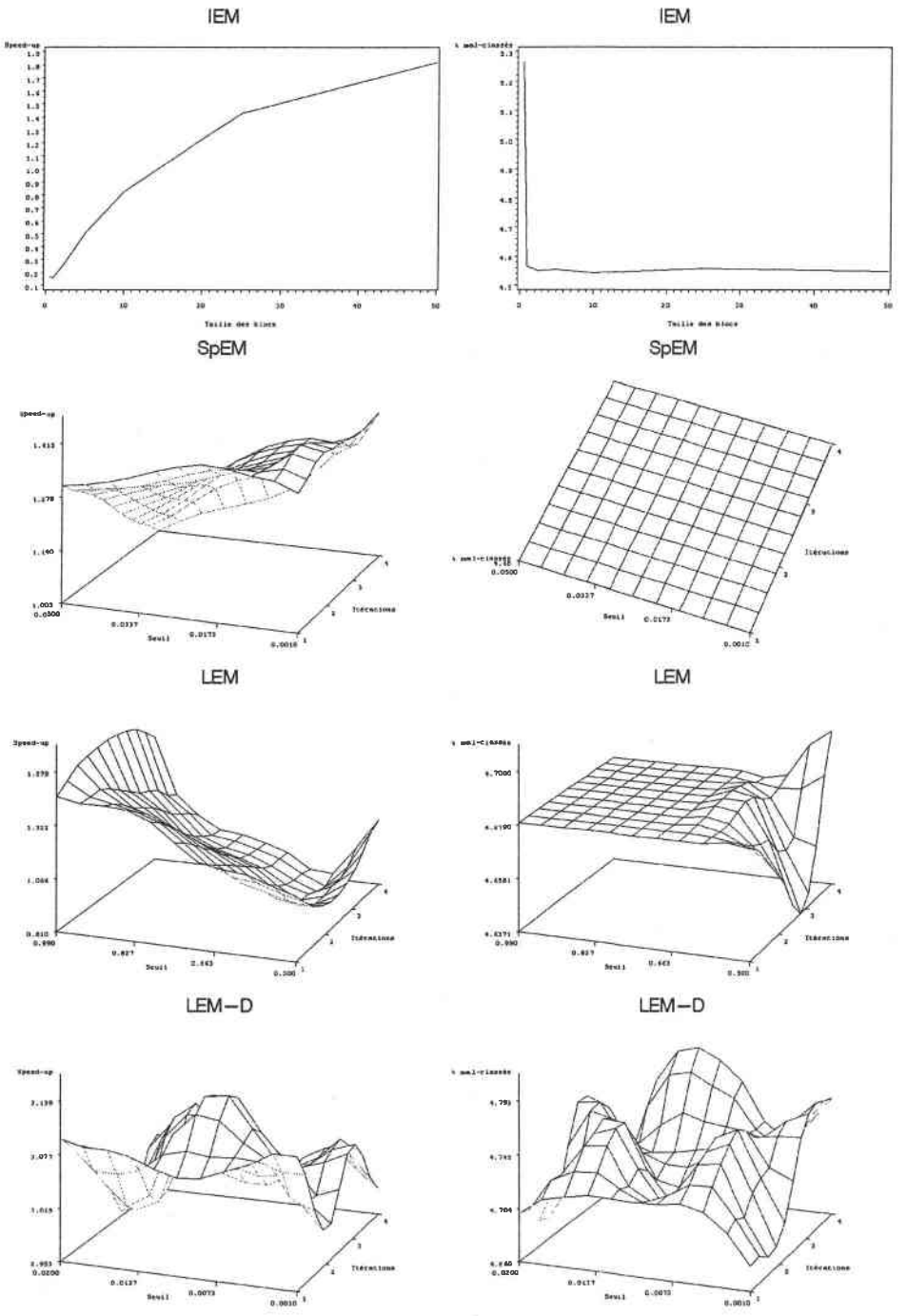



FIG. 4.6 – *Degré de mélange (+) : IEM, SpEM, LEM et LEM-D (Coefficient d'accélération à gauche et pourcentage de mal-classés à droite). Ici, SpEM trouve toujours le même pourcentage de mal-classés, c'est pourquoi le graphique est ainsi.*

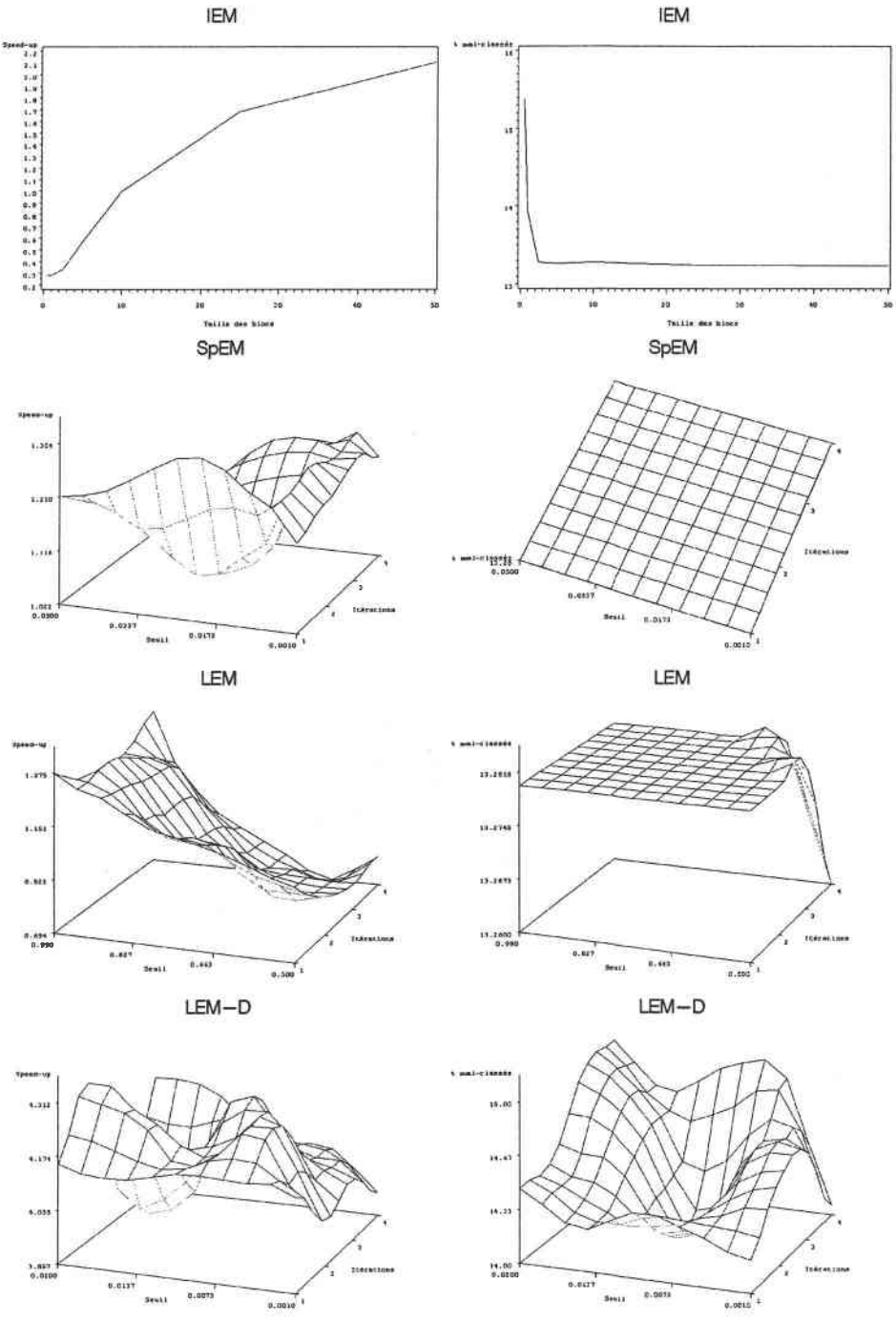



FIG. 4.7 – *Degré de mélange (++) : IEM, SpEM, LEM et LEM-D (Coefficient d'accélération à gauche et pourcentage de mal-classés à droite). Ici, SpEM trouve toujours le même pourcentage de mal-classés, c'est pourquoi le graphique est ainsi.*

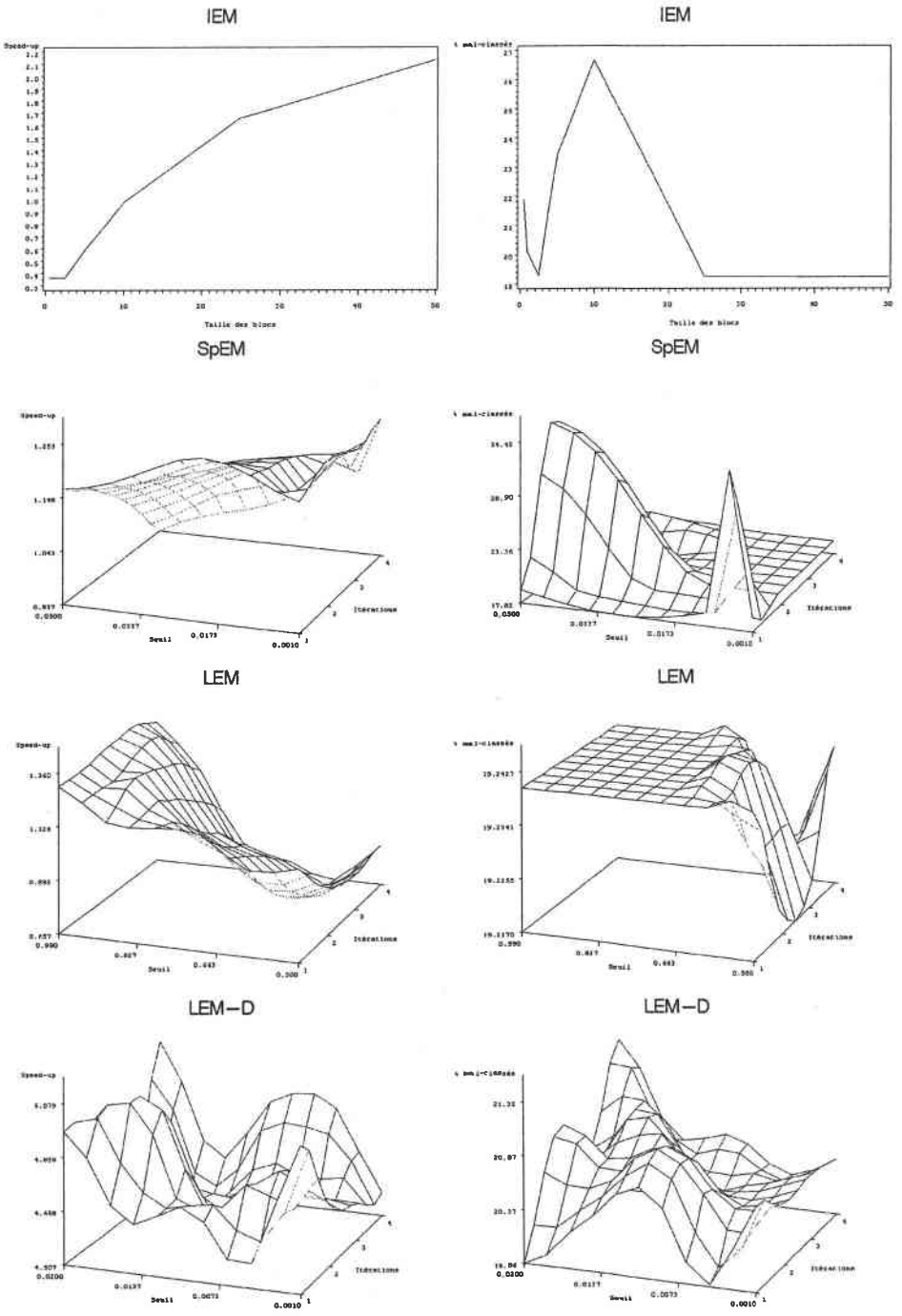



FIG. 4.8 – *Degré de mélange (+++) : IEM, SpEM, LEM et LEM-D (Coefficient d'accélération à gauche et pourcentage de mal-classés à droite).*

TAB. 4.10 – *Meilleur (+) et moins bon (–) résultats pour chaque méthode d'accélération de l'algorithme EM, sur la moyenne des 10 tableaux simulés pour le mélange (++).*

| | Méthode | + | | – | |
|---|---|---|---|---|---|
| | | Résultat | Paramètres | Résultats | Paramètres |
| $L$ | IEM | -85874.20 | 10.00 | -86047.27 | 0.50 |
| | SpEM | -85758.89 | 0.050 et 2 | -85893.79 | presque tous |
| | LEM | -85865.84 | 0.70 et 3 | -85893.80 | 0.50 et 4 |
| | LEM-D | -85868.09 | 0.015 et 3 | -85910.72 | 0.020 et 4 |
| Coefficient d'accélération | IEM | 2.13 | 50.00 | 0.36 | 0.50 |
| | SpEM | 1.23 | 0.010 et 1 | 0.94 | 0.050 et 4 |
| | LEM | 1.36 | 0.99 et 3 | 0.66 | 0.60 et 4 |
| | LEM-D | 5.08 | 0.020 et 4 | 4.61 | 0.010 et 2 |
| Pourcentage de mal-classés | IEM | 19.23 | 50.00 | 26.66 | 10.00 |
| | SpEM | 19.17 | presque tous | 31.71 | 0.005 et 2 |
| | LEM | 19.16 | 0.50 et 2 | 27.53 | 0.99 et 4 |
| | LEM-D | 19.92 | 0.020 et 1 | 27.37 | 0.015 et 3 |

sur l'utilisation d'un algorithme dérivé de EM, mais plus rapide, afin d'obtenir un point de départ *proche* du résultat. Ces deux algorithmes sont CEM et eM, équivalent à EM, mais avec un critère d'arrêt permettant un nombre d'itérations proche de celui de CEM.

Ensuite, nous avons présenté des méthodes d'accélération de la convergence de l'algorithme EM. Ces méthodes se basent sur un calcul restreint de probabilités a posteriori, en mettant de côté un certain nombre d'objets, selon des critères à définir. Deux méthodes se sont révélées meilleures, la première est LEM, consistant à ne garder que les objets qui ont toutes leurs probabilités $t_{ik}$ inférieur à un certain seuil. La seconde est LEM-D, une version dérivée de LEM, qui choisit les objets en fonction de leur différence moyenne des $t_{ik}$ entre deux itérations, ceux ayant une faible différence, considérés comme stables, étant écartés du calcul.

Nous présentons dans le tableau 4.11 les résultats de toutes les différentes méthodes étudiées (EM, CEM-EM, eM-EM, IEM, SpEM, LEM et LEM-D) en conditions réelles. Nous avons utilisé pour ceci les tableau de données ADN, Congressional Votes, Titanic et Mushroom. Nous présentons dans le tableau suivant les paramètres utilisés.

| Méthode | Paramètres | |
|---|---|---|
| | | Taille des blocs |
| IEM | | 0.5, 1, 2.5, 5, 10, 25 et 50 % |
| | Itération | Seuils |
| SpEM | 1, 2, 3 et 4 | 0.001, 0.005, 0.010 et 0.050 |
| LEM | 1, 2, 3 et 4 | 0.50, 0.60, 0.70, 0.80, 0.90, 0.95 et 0.99 |
| LEM-D | 1, 2, 3 et 4 | 0.001, 0.005, 0.010, 0.015 et 0.020 |

Pour chaque méthode, nous présentons le résultat obtenu le plus rapidement, et le paramètrage qui a permis de l'obtenir. De plus, nous proposons d'utiliser une méthodologie particulière pour l'utilisation de LEM-D (noté (*) par la suite), qui

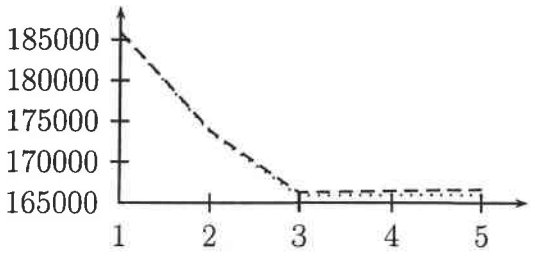



FIG. 4.9 – *Évolution des critères BIC (tiret) et AIC3 (pointillé) sur des données simulées de grande taille, calculés avec la méthode LEM-D (*). Les deux courbes sont très proches l'une de l'autre.*

consiste à prendre $n_{lazy} = 1$ et *seuil* = 0.001, 0.005, 0.010, et lancer 7 fois l'algorithme avec chaque seuil. Les résultats sont présentés dans le tableau 4.11.

La méthode CEM-EM est souvent rapide, mais peut fournir parfois de très mauvais résultats (cf Titanic). Cela est dû aux faiblesses de l'algorithme CEM, qui a parfois tendance à vider les petites classes, particulièrement lorsque les données sont très mélangées.

Mis à part pour Titanic, notre méthode LEM-D est la plus rapide, tout en fournissant de très bons résultats. Notons toutefois que pour le tableau Titanic, la vraisemblance obtenue par LEM-D est très proche de celle obtenue avec EM, contrairement à LEM. Les bons résultats de notre de méthode s'obtiennent avec des seuils différents, selon le tableau, ainsi qu'un nombre d'itérations *lazy* variant. Par contre, en utilisant notre stratégie d'utilisation de LEM-D, (*), on remarque que l'accélération proposée est plutôt intéressante et même très proche de celle obtenue avec un seul seuil, et les résultats sont toujours bons. Elle semble donc très intéressante à utiliser, afin d'éviter à l'utilisateur de choisir un seuil et un nombre d'itération *lazy*.

Comme nous l'avons déjà indiqué dans le chapitre précédent, une des utilisations possibles de cette méthode est la recherche du nombre de classes avec un critère (comme BIC et AIC3), de manière plus rapide qu'avec EM. Nous présentons ici une illustration, sur des données simulées, avec $n = 10000$, $d = 10$, $p = \{0.5,0.2,0.3\}$ et $c_j$ pour toutes les variables $j$. Nous avons choisi de prendre des classes moyennement mélangées (++). Nous reportons dans la figure 4.9 les évolutions des critères BIC et AIC3, qui nous semblent les plus intéressants. Les deux trouvent le bon nombre de classes $s = 3$, en 1046.35 secondes en tout. Pour comparaison, c'est environ le temps que met EM sur ce tableau en calculant juste les critères pour $s = 3$. Notre méthode se montre donc très performante et très rapide.

TAB. 4.11 – *Comparaison de différentes méthodes d'accélération et d'initialisation de EM, sur des données réelles.*

| Données | Méthode | Paramètres | $L$ | Temps (en s) | Coefficient d'accélération | % de mal classés |
|---|---|---|---|---|---|---|
| Votes | EM | | -4464.82 | 25.27 | 1.00 | 13.1 |
| | CEM-EM | | -4464.82 | 9.79 | 2.58 | 13.1 |
| | eM-EM | | -4464.82 | 9.70 | 2.61 | 13.1 |
| | IEM | 25 % | -4464.82 | 16.27 | 1.55 | 13.1 |
| | SpEM | 1, 0.005 | -4464.82 | 16.28 | 1.55 | 13.1 |
| | LEM | 1, 0.60 | -4464.82 | 6.78 | 3.72 | 13.1 |
| | LEM-D | 3, 0.001 | -4464.83 | 3.78 | 6.69 | 12.9 |
| | LEM-D | (*) | -4464.82 | 4.13 | 6.12 | 13.1 |
| Titanic | EM | | -4132.01 | 21.27 | 1.00 | 22.4 |
| | CEM-EM | | -4564.92 | 2.73 | 7.79 | 54.0 |
| | eM-EM | | -4132.01 | 16.79 | 1.27 | 22.7 |
| | IEM | 25 % | -4132.48 | 22.67 | 0.94 | 22.4 |
| | SpEM | 3, 0.050 | -4131.80 | 13.07 | 1.63 | 22.7 |
| | LEM | 3, 0.50 | -4383.57 | 1.65 | 12.89 | 22.7 |
| | LEM-D | 1, 0.010 | -4135.71 | 9.65 | 2.20 | 22.7 |
| | LEM-D | (*) | -4135.83 | 10.55 | 2.02 | 22.7 |
| ADN | EM | | -82507.50 | 485.68 | 1.00 | 4.8 |
| | CEM-EM | | -82507.50 | 89.50 | 5.43 | 4.8 |
| | eM-EM | | -82507.50 | 283.63 | 1.71 | 4.8 |
| | IEM | 50 % | -82507.52 | 230.85 | 2.10 | 4.8 |
| | SpEM | 1, 0.050 | -82507.47 | 292.20 | 1.66 | 4.8 |
| | LEM | 1, 0.99 | -82507.52 | 321.80 | 1.51 | 4.8 |
| | LEM-D | 1, 0.005 | -82513.28 | 141.39 | 3.44 | 4.9 |
| | LEM-D | (*) | -82512.43 | 155.30 | 3.13 | 4.8 |
| Mushroom | EM | | -150988.36 | 19028.05 | 1.00 | 10.0 |
| | CEM-EM | | -150987.70 | 1080.89 | 17.60 | 12.6 |
| | eM-EM | | -150987.58 | 1510.76 | 12.59 | 10.6 |
| | IEM | 50 % | -150995.17 | 529.04 | 36.31 | 10.1 |
| | SpEM | 1, 0.001 | -150986.33 | 475.65 | 40.00 | 10.6 |
| | LEM | 3, 0.99 | -151138.49 | 557.04 | 34.16 | 10.6 |
| | LEM-D | 1, 0.005 | -151189.60 | 226.53 | 84.00 | 10.6 |
| | LEM-D | (*) | -151089.06 | 265.20 | 71.75 | 11.0 |

# Chapitre 5

# Classification croisée

Lorsque nous sommes en présence de données résultant d'un croisement de deux ensembles, comme c'est généralement le cas, la classification automatique, telle qu'elle est couramment pratiquée, privilégie un des deux ensembles en ne faisant porter la structure recherchée que sur les objets ou les variables. Il est bien évidement possible de faire une double classification, en lignes puis en colonnes, et de croiser les partitions obtenues. Mais dans ce cas, les liens entre les deux partitions ne seront pas forcément mis en évidence. La figure 5.1 (Govaert, 1995) montre clairement l'intérêt d'une classification croisée. Nous voyons à gauche les données initiales, au milieu les données réordonnées selon la partition en lignes obtenu avec un algorithme de type simple, et à droite, le résultat fourni par un algorithme de classification croisé. Les blocs obtenus ainsi sont très nets et très illustratifs.

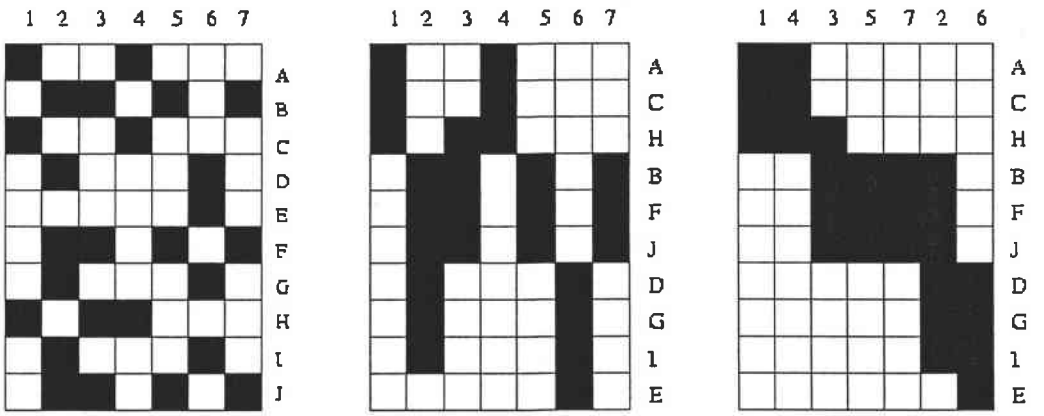



FIG. 5.1 – *Classification simple et classification croisée.*

C'est ainsi que des méthodes de classification croisée ont vues leur apparition. Elles visent à obtenir des blocs de données les plus homogènes possible, selon des critères métriques ou probabilistes. Nous nous intéressons dans ce chapitre aux méthodes appartenant au premier groupe. De plus, nous pouvons dissocier deux types de classification différentes :

- Le premier inclut les algorithmes cherchant simultanément une partition en lignes et une partition en colonnes. Ainsi, nous présentons les algorithmes proposés par Govaert (1983) : *Croeuc*, *Crobin* et *Croki2*.

- Le second type regroupe les méthodes de classification directe (Hartigan, 1972; Hartigan, 1975). Celles-ci cherchent à isoler dans les données des blocs homogènes, de toutes tailles, ainsi que des hiérarchies en lignes et en colonnes. Nous présenterons donc les deux algorithmes *One-way splitting* et *Two-way splitting*.

## 5.1 Partitionnement double

### 5.1.1 L'algorithme *Croeuc*

L'algorithme *Croeuc* (Govaert, 1983), appliqué aux données continues, consiste à optimiser un critère $W(\mathbf{z},\mathbf{w},\mathbf{g})$, où $\mathbf{z} = (z_1,\ldots,z_s)$ est la partition des objets en $s$ classes, $\mathbf{w} = (w_1,\ldots,w_m)$ la partition des variables en $m$ classes et $\mathbf{g} = (g_k^\ell)$ une matrice $s \times m$ qui correspond à un résumé de l'information. La recherche des partitions optimales $\mathbf{z}$ et $\mathbf{w}$ se fait grâce à un algorithme itératif. En adoptant la somme des distances euclidiennes comme une mesure de l'écart entre la matrice de données $\mathbf{x}$ et les structures décrites par $\mathbf{z} = (z_1,\ldots,z_s)$, $\mathbf{w} = (w_1,\ldots,w_m)$ et $\mathbf{g} = (g_k^\ell)$, le problème est de trouver une paire de partition $(\mathbf{z},\mathbf{w})$ et le paramètre correspondant $\mathbf{g}$, tels que le critère suivant soit minimisé :

$$W(\mathbf{z},\mathbf{w},\mathbf{g}) = \sum_{k=1}^{s}\sum_{\ell=1}^{m}\sum_{i\in z_k}\sum_{j\in w_\ell}(x_i^j - g_k^\ell)^2 \tag{5.1}$$

où $g_k^\ell$ est le centre du bloc $\mathbf{x}_k^\ell$. Il est facile de voir que pour $(\mathbf{z},\mathbf{w})$ fixés, les valeurs optimales de $(g_k^\ell)$ sont les moyennes de chaque $x_i^j$ appartenant au bloc $\mathbf{x}_k^\ell$. Les différentes étapes de l'algorithme *Croeuc* sont :

1. Démarrer d'une position initiale $(\mathbf{z}^0,\mathbf{w}^0,\mathbf{g}^0)$.
2. Calculer $(\mathbf{z}^{(c+1)},\mathbf{w}^{(c+1)},\mathbf{g}^{(c+1)})$ à partir de $(\mathbf{z}^{(c)},\mathbf{w}^{(c)},\mathbf{g}^{(c)})$ :
   (a) Calculer $(\mathbf{z}^{(c+1)},\mathbf{w}^{(c)},\mathbf{g}')$ à partir de $(\mathbf{z}^{(c)},\mathbf{w}^{(c)},\mathbf{g}^{(c)})$.
   (b) Calculer $(\mathbf{z}^{(c+1)},\mathbf{w}^{(c+1)},\mathbf{g}^{(c+1)})$ à partir de $(\mathbf{z}^{(c+1)},\mathbf{w}^{(c)},\mathbf{g}')$.
3. Recommencer l'étape 2 jusqu'à la convergence de l'algorithme.

Dans l'étape 2, on utilise l'algorithme de classification standard $k$-means sur les lignes, puis sur les colonnes. A chaque application, on bloque ainsi la partition qui n'est pas à optimiser. Nous cherchons donc à optimiser alternativement les critères suivants (déduits de 5.1) :

$$W(\mathbf{z},\mathbf{g}/\mathbf{w}) = \sum_{k=1}^{s}\sum_{i\in z_k}\sum_{\ell=1}^{m}\#w_\ell(u_i^\ell - g_k^\ell)^2,$$

où $u_i^\ell = \frac{\sum_{j\in w_\ell} x_i^j}{\#w_\ell}$, et

$$W(\mathbf{w},\mathbf{g}/\mathbf{z}) = \sum_{\ell=1}^{m}\sum_{j\in w_\ell}\sum_{k=1}^{s}\#z_k(v_k^j - g_k^\ell)^2,$$

où $v_k^j = \frac{\sum_{i\in z_k} x_i^j}{\#z_k}$ (# correspond à la cardinalité). Comme tout algorithme de classification, croisée ou non, *Croeuc* nécessite la connaissance du nombre de classes.

TAB. 5.1 – *Tableau 5 × 10 initial.*

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | -0.78 | -1.09 | 0.34 | 0.53 | 1.89 | -0.25 | 1.48 | -0.08 | 1.06 | -0.82 |
| 2 | 1.19 | 0.64 | 1.30 | -1.82 | -0.88 | 0.62 | -0.64 | 0.02 | -1.87 | 1.18 |
| 3 | 1.42 | -0.06 | 0.38 | -1.75 | -0.62 | -0.63 | -1.31 | -0.82 | -0.43 | 1.91 |
| 4 | 1.69 | -0.22 | 0.52 | -1.17 | -1.36 | 0.88 | -1.74 | -0.13 | -0.64 | 0.30 |
| 5 | -1.35 | 0.61 | -0.37 | 0.48 | 0.90 | 0.24 | 1.27 | -1.17 | 0.95 | -0.66 |

TAB. 5.2 – *Tableau 5 × 10 réordonné selon les partitions obtenus avec Croeuc.*

| | 1 | 10 | 2 | 3 | 6 | 8 | 4 | 5 | 7 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | -0.78 | -0.82 | -1.09 | 0.34 | -0.25 | -0.08 | 0.53 | 1.89 | 1.48 | 1.06 |
| 5 | -1.35 | -0.66 | 0.61 | -0.37 | 0.24 | -1.17 | 0.48 | 0.90 | 1.27 | 0.95 |
| 2 | 1.19 | 1.18 | 0.64 | 1.30 | 0.62 | 0.02 | -1.82 | -0.88 | -0.64 | -1.87 |
| 3 | 1.42 | 1.91 | -0.06 | 0.38 | -0.63 | -0.82 | -1.75 | -0.62 | -1.31 | -0.43 |
| 4 | 1.69 | 0.30 | -0.22 | 0.52 | 0.88 | -0.13 | -1.17 | -1.36 | -1.74 | -0.64 |

En effet, il est impératif de le connaître à l'avance afin de pouvoir lancer l'algorithme. Notons que cet algorithme se réduit à $k$-means classique lorsque nous nous restreignons à la recherche d'une seule des deux partitions.

### Illustration sur un exemple

Pour illustrer le fonctionnement de l'algorithme *Croeuc*, nous avons choisi de l'appliquer sur un exemple. Les données initiales, standardisées, sont présentées dans le tableau 5.1, et de manière réordonnée selon les partitions en lignes et en colonnes obtenues avec *Croeuc* dans le tableau 5.2. On peut noter que $W_{max} = 13.737329$ est le critère minimal que l'on a obtenu. De plus, dans le tableau 5.3, nous avons reporté les centres et les écarts-types pour chaque bloc de données.

## 5.1.2 L'algorithme *Crobin*

Quand les données sont binaires, nous pouvons définir les valeurs de $\mathbf{x}$ par $x_i^j = 1$ si l'objet $i$ possède l'attribut $j$ et $x_i^j = 0$ sinon. Étant donné $\mathbf{x}$, le problème consiste dans la minimisation du critère suivant

$$W(\mathbf{z},\mathbf{w},\mathbf{a}) = \sum_{k=1}^{s}\sum_{\ell=1}^{m}\sum_{i\in z_k}\sum_{j\in w_\ell} |x_i^j - a_k^\ell| \quad \text{avec} \quad a_k^\ell \in \{0,1\}. \tag{5.2}$$

TAB. 5.3 – *Centres et écarts-types obtenus avec Croeuc, avec les proportions.*

| | $\mathbf{g}^1$ | $\mathbf{g}^2$ | $\mathbf{g}^3$ |
|---|---|---|---|
| $\mathbf{g}_1$ | -0.90 (± 0.26) | -0.22 (± 0.60) | 1.07 (± 0.44) |
| $\mathbf{g}_2$ | 1.28 (± 0.51) | 0.21 (± 0.60) | -1.19 (± 0.51) |

TAB. 5.4 – *Noyaux et degrés d'homogénéité (entre parenthèses) obtenus avec Crobin sur le tableau binaire.*

| | $\mathbf{a}^2$ | $\mathbf{a}^2$ |
|---|---|---|
| $\mathbf{a}_1$ | 0 (0.71) | 1 (0.89) |
| $\mathbf{a}_2$ | 1 (0.86) | 0 (1.00) |

Les différentes étapes de l'algorithme *Crobin* sont

1. Démarrer d'une position initiale $(\mathbf{z}^0,\mathbf{w}^0,\mathbf{a}^0)$.
2. Calculer $(\mathbf{z}^{(c+1)},\mathbf{w}^{(c+1)},\mathbf{a}^{(c+1)})$ à partir de $(\mathbf{z}^{(c)},\mathbf{w}^{(c)},\mathbf{a}^{(c)})$:
   (a) Calculer $(\mathbf{z}^{(c+1)},\mathbf{w}^{(c)},\mathbf{a}')$ à partir de $(\mathbf{z}^{(c)},\mathbf{w}^{(c)},\mathbf{a}^{(c)})$.
   (b) Calculer $(\mathbf{z}^{(c+1)},\mathbf{w}^{(c+1)},\mathbf{a}^{(c+1)})$ à partir de $(\mathbf{z}^{(c+1)},\mathbf{w}^{(c)},\mathbf{a}')$.
3. Recommencer l'étape 2 jusqu'à la convergence.

Dans les étapes 2(a) et 2(b), pour trouver les partitions optimales $\mathbf{z}^{(c+1)}$ et $\mathbf{w}^{(c+1)}$, on utilise l'algorithme des nuées dynamiques (Diday, 1980) pour optimiser les critères suivant déduit de (5.2)

$$W(\mathbf{z},\mathbf{a}|\mathbf{w}) = \sum_{k=1}^{s}\sum_{i\in z_k}\sum_{\ell=1}^{m}|u_i^\ell - \#w_\ell a_k^\ell| \text{ où } u_i^\ell = \sum_{j\in w_\ell} x_i^j$$

$$\text{et } W(\mathbf{w},\mathbf{a}|\mathbf{z}) = \sum_{\ell=1}^{m}\sum_{j\in w_\ell}\sum_{k=1}^{s}|v_k^j - \#z_k a_k^\ell| \text{ où } v_k^j = \sum_{i\in z_k} x_i^j,$$

avec # correspondant à la cardinalité. L'étape 2(a) consiste en l'application de l'algorithme des nuées dynamiques utilisant la matrice $n \times m$ $(u_i^\ell)$, la distance $L_1$ et des noyaux de la forme $(\#w_1 a_k^1,\ldots,\#w_m a_k^m)$. De la même manière, l'étape 2(b) consiste en l'application du même algorithme en utilisant la matrice $s \times d$ $(v_k^j)$, la distance $L_1$ et des noyaux de la forme $(\#z_1 a_1^\ell,\ldots,\#z_s a_s^\ell)$. Ainsi, à la convergence, nous obtenons des blocs homogène de 0 ou de 1 en réorganisant les lignes et les colonnes selon les partitions $\mathbf{z}$ et $\mathbf{w}$. De cette manière, chaque bloc $\mathbf{x}_k^\ell$, défini par les éléments $x_i^j$ pour $i \in z_k$ et $j \in w_\ell$, est caractérisé par $a_k^\ell$ qui est simplement la valeur la plus fréquente.

### Illustration sur un exemple

De la même manière que pour *Croeuc*, nous avons choisi d'illustrer le comportement de *Crobin* sur un exemple. Nous avons choisi $n = 5$, $d = 10$, $s = 2$, $m = 2$. Dans la figure 5.2, nous présentons d'un côté la matrice initiale, puis la même matrice réordonnée selon les partitions obtenues avec *Crobin*. On peut noter que le critère optimal trouvé est $W_{max} = 9$. Dans le tableau 5.4, nous avons indiqué les noyaux et les degré d'homogénéité de chaque bloc. On voit ainsi que le bloc $\mathbf{x}_2^2$ est pur (degré d'homogénéité à 1, toutes les valeurs sont égales au mode $a_2^2$, 0). Par contre, le bloc $\mathbf{x}_1^1$ contient 29 % d'erreur.

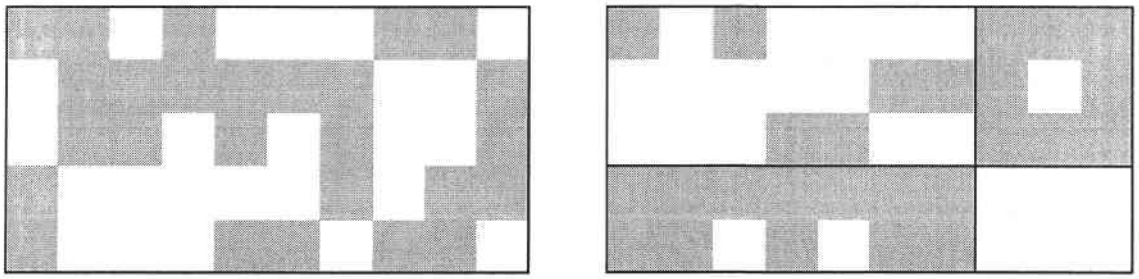


FIG. 5.2 – *Matrice initiale (à gauche) et réordonnée selon les partitions obtenues avec Crobin (à droite). Les 1 sont grisés et les 0 blancs.*

### 5.1.3 L'algorithme *Croki2*

Dans le cas d'un tableau de contingence, ou de tableaux présentant des propriétés équivalentes, Govaert (1983) a proposé l'algorithme *Croki2*. De la même manière que pour les deux précédents algorithmes, la méthode utilisée est l'optimisation alternée de la partition en lignes, puis de celle en colonnes. Pour obtenir ces partitions, on utilise une variante des nuées dynamiques, en prenant la métrique du $\chi^2$ et le centre de gravité comme noyau.

On défini donc le critère de convergence général par :

$$\chi^2(\mathbf{z},\mathbf{w}) = \sum_{k=1}^{s}\sum_{\ell=1}^{m} \frac{(f_{k\ell} - f_{k.}\, f_{.\ell})^2}{f_{k.}\, f_{.\ell}} \quad \text{avec} \quad f_{k\ell} = \frac{t_{k\ell}}{t} = \frac{\sum_{i\in z_k}\sum_{j\in w_\ell} t_i^j}{t}$$

**Optimisation en lignes.** On bloque la partition en colonnes $\mathbf{w}$ et on ne travaille que sur la partition en lignes $\mathbf{z}$. Pour ceci, on recréé un tableau de contingence $U$ à partir de $T$ de cette manière :

$$u_{i\ell} = \sum_{j\in w_\ell} t_{ij}$$

Et on cherche donc à optimiser le critère suivant avec l'algorithme basique des nuées dynamiques simples :

$$\begin{aligned} W(\mathbf{z}) &= \sum_{k=1}^{s}\sum_{i\in z_k} f_{i.}\ \mathrm{d}^2(u_i, g_k) \\ &= \sum_{k=1}^{s}\sum_{i\in z_k} f_{i.} \sum_{\ell=1}^{m} \frac{(u_{i\ell} - g_{k\ell})^2}{f_{.\ell}} \\ &= \sum_{k=1}^{s}\sum_{i\in z_k} u_{i.} \sum_{\ell=1}^{m} \frac{(u_{i\ell} - g_{k\ell})^2}{u_{.\ell}} \end{aligned}$$

**Optimisation en colonnes.** Contrairement à l'étape précédente, on fixe la partition en lignes $\mathbf{z}$ et on travaille que sur la partition en colonnes $\mathbf{w}$. On obtient donc le tableau de contingence $V$ à partir de $T$ ainsi :

$$v_{kj} = \sum_{i\in z_k} t_{ij}$$

Et on cherche donc à optimiser le critère suivant avec l'algorithme basique des nuées dynamiques simples :

$$\begin{aligned} W(\mathbf{w}) &= \sum_{\ell=1}^{m} \sum_{j \in w_\ell} f_{.j}\, \mathbf{d}^2(v_j, g_\ell) \\ &= \sum_{\ell=1}^{m} \sum_{j \in w_\ell} f_{.j} \sum_{k=1}^{s} \frac{(v_{kj} - g_{k\ell})^2}{f_{k.}} \\ &= \sum_{\ell=1}^{m} \sum_{j \in w_\ell} v_{.j} \sum_{k=1}^{s} \frac{(v_{kj} - g_{k\ell})^2}{v_{k.}} \end{aligned}$$

## 5.2 Algorithmes de classification directe

### 5.2.1 L'algorithme *One-way splitting*

Ce premier algorithme divisif, *One-way splitting*, a été présenté par Hartigan (1975). Il fait parti des algorithmes dit de classification directe. Ainsi, au lieu de proposer seulement une partition en lignes et une partition en colonnes, il propose un découpage en blocs homogènes des objets. Cet algorithme se concentre principalement sur la partition des objets, en essayant de construire des classes de telle manière que les variables ont une variance intra-classe inférieure à un certain seuil. L'idée de base de l'algorithme est de n'utiliser que les variables ayant une variance supérieure au seuil dans une classe donnée pour découper cette classe.

Avant de décrire les étapes de l'algorithme, nous allons définir quelques notations. On utilise une matrice de données $A(I,J)$ $(1 \leq I \leq M,\ 1 \leq J \leq N)$.

Les classes en lignes $1,2,\ldots,KC$ sont construites telle que la classe $I$ est déterminée par les classes qui la divisent $Min(I)$ et $Max(I)$. Pour une classe minimale (que l'on ne peut plus diviser), ces valeurs représentent la première et la dernière ligne de la classe $I$. A la fin de l'algorithme, on définit $V(I)$ l'ensemble des variables qui ont une variance inférieure au seuil dans $I$, et dans aucune autre classe plus grande.

L'algorithme procède par découpages de classes successifs. A l'étape $K$, il y a $K$ classes $I(1),I(2),\ldots,I(K)$ séparant les lignes, qui sont les classes minimales dans l'ensemble $1,2,3,\ldots,2K-1$. Durant l'algorithme, $V[I(J)]$ est l'ensemble des variables avec une variance supérieure au seuil $T$ pour toutes classes plus grandes.

Un découpage en deux est effectué sur les classes $I(J)$, en utilisant seulement les variables dans $V[I(J)]$ qui ont une variance supérieure au seuil. Les deux nouvelles classes $2K$ et $2K+1$ auront $V(2K) = V(2K+1)$ défini comme l'ensemble des variables de $I(J)$ qui ont une variance supérieure à $T$ dans $I(J)$. Et donc, $V[I(J)]$ sera changé en l'ensemble des variables de $V[I(J)]$ qui auront donc une variance inférieure à $T$ dans $I(J)$. Le découpage s'arrête lorsque tous les ensembles $V[I(J)]$ contiennent des variables avec une variance inférieure au seuil dans $I(J)$.

L'algorithme peut ainsi être décrit comme suit :

**Étape 1** On fixe le seuil $T$, et on démarre avec les objets dans une seule et même classe. On initialise de la manière suivante : $Min(1) = 1$, $Max(1) = M$, $K = 1$, $I(1) = 1$, $L = 1$ et $V(1) = \{1,2,\ldots,N\}$.

**Étape 2** On cherche à diviser $I(L)$. On calcule la variance pour chaque variable $J$ $\{J \in V[I(L)]\}$ pour tous les objets dans la classe $I(L)$. Si aucune de ces variances n'est supérieure à $T$, alors allez à l'étape 4. Sinon, on va à l'étape 3.

**Étape 3** Posons $VV$ l'ensemble des variables qui ont une variance supérieure à $T$ pour les objets de $I(L)$. On applique l'algorithme *binary splitting* (dérivé de $k$-means, où $k = 2$), sur les objets $I$, $Min[I(L)] \leq I \leq Max[I(L)]$, décrits par les variables de $VV$. On réordonne les lignes de tel sorte que la première classe comprendra les lignes $IL$,$IL+1$,...,$II$, et la seconde les lignes $II+1$,...,$IU$. On définit les deux nouvelles classes $K+1$ et $K+2$ avec:

$$Min(K+1) = IL,\ Max(K+1) = II$$
$$Min(K+2) = II+1,\ Max(K+2) = IU$$
$$Max[I(L)] = K+2,\ Min[I(L)] = K+1$$
$$V(K+1) = V(K+2) = VV,\ V[I(L)] = V[I(L)] - VV$$
$$I(L) = K+1,\ I(K+1) = K+2$$

On incrémente $K$ de 1, puis on retourne à l'étape 2.

**Étape 4** On incrémente $L$ de 1. Si $L \leq K$, alors on va à l'étape 2. Sinon, on arrête l'algorithme.

Pour mieux comprendre cet algorithme, nous allons voir son application sur l'exemple de (Hartigan, 1975), en détaillant chaque étape. Ce tableau représente le pourcentage de votes républicains pour les présidentielles, dans certains états du Sud des États-Unis (Caroline du Sud (SC), Mississippi (MS), Louisiane (LA), Kentucky (KY), Maryland (MD) et Missouri (MO)), pour plusieurs années (1932, 1936, 1940, 1960, 1964 et 1968). Ici, soit les états, soit les années peuvent être traités comme variables. Nous choisissons de prendre les années comme variables. Comme toutes techniques de classification directe, un seuil petit donnera trop de petites classes, alors qu'un seuil trop grand donnera peu de classes, et de grandes tailles. Le seuil choisi de 25 correspond à un écart-type de 5, qui semble être une erreur raisonnable entre états similaires. Nous décrivons les différentes étapes de cet algorithme pour ce tableau dans la figure 5.3, ainsi qu'une représentation des résultats dans la figure 5.4.

On voit ainsi que les états KY, MD et MO ont le même comportement pour chacune des années étudiées. Par contre, les états MS, SC et LA ont un comportement plutôt différent, mis à part pour l'année 1932 où ils se ressemblent, et pour l'année 1964 où SC et LA sont similaires.

### 5.2.2 L'algorithme *Two-way splitting*

Le second algorithme divisif que nous allons présenté, appelé *Two-way splitting algorithm*, a aussi été introduit par Hartigan (1975). De la même manière que l'algorithme précédent, celui-ci fait parti des algorithmes de classification directe, qui choisit à chaque étape entre une division des objets et une division des variables. Ce choix se base sur la réduction maximum de l'hétérogénéité du groupe d'objets ou de variables divisé. Afin de respecter les contraintes hiérarchiques imposées pour cet algorithme, les divisions effectuées à une étape ne sont jamais remises en cause aux étapes suivantes. Cet algorithme ne nécessite pas de savoir à l'avance le nombre de blocs que l'on veut obtenir.

Tableau initial

| | 32 | 36 | 40 | 60 | 64 | 68 |
|---|---|---|---|---|---|---|
| SC | 2 | 1 | 4 | 49 | 59 | 39 |
| MS | 4 | 3 | 4 | 25 | 87 | 14 |
| LA | 7 | 11 | 14 | 29 | 57 | 23 |
| KY | 40 | 40 | 42 | 57 | 36 | 44 |
| MD | 36 | 37 | 41 | 46 | 35 | 42 |
| MO | 35 | 38 | 48 | 50 | 36 | 45 |

Premier découpage

| | 32 | 36 | 40 | 60 | 64 | 68 |
|---|---|---|---|---|---|---|
| SC | 2 | 1 | 4 | 49 | 59 | 39 |
| MS | 4 | 3 | 4 | 25 | 87 | 14 |
| LA | 7 | 11 | 14 | 29 | 57 | 23 |
| KY | 40 | 40 | 42 | 57 | 36 | 44 |
| MD | 36 | 37 | 41 | 46 | 35 | 42 |
| MO | 35 | 38 | 48 | 50 | 36 | 45 |

Deuxième découpage

| | 32 | 36 | 40 | 60 | 64 | 68 |
|---|---|---|---|---|---|---|
| MS | 4 | 3 | 4 | 25 | 87 | 14 |
| SC | 2 | 1 | 4 | 49 | 59 | 39 |
| LA | 7 | 11 | 14 | 29 | 57 | 23 |
| KY | 40 | 40 | 42 | 57 | 36 | 44 |
| MD | 36 | 37 | 41 | 46 | 35 | 42 |
| MO | 35 | 38 | 48 | 50 | 36 | 45 |

Dernier découpage

| | 32 | 36 | 40 | 60 | 64 | 68 |
|---|---|---|---|---|---|---|
| MS | 4 | 3 | 4 | 25 | 87 | 14 |
| SC | 2 | 1 | 4 | 49 | 59 | 39 |
| LA | 7 | 11 | 14 | 29 | 57 | 23 |
| KY | 40 | 40 | 42 | 57 | 36 | 44 |
| MD | 36 | 37 | 41 | 46 | 35 | 42 |
| MO | 35 | 38 | 48 | 50 | 36 | 45 |

FIG. 5.3 – *Déroulement de l'algorithme One-way splitting, avec les découpages successifs. Les valeurs encadrées sont ignorées dans les étapes suivantes, car la variance de ces blocs est inférieur au seuil.*

| | 32 | 36 | 40 | 60 | 64 | 68 |
|---|---|---|---|---|---|---|
| MS | | 3 | 4 | 25 | 87 | 14 |
| SC | 4 | 1 | 4 | 49 | 58 | 39 |
| LA | | 11 | 14 | 29 | | 23 |
| KY | | | | | | |
| MD | 37 | 39 | 44 | 50 | 36 | 44 |
| MO | | | | | | |

FIG. 5.4 – *Représentation des résultats obtenus avec l'algorithme One-way splitting sur les votes républicains, avec les moyennes pour chaque variable et pour chaque classe.*

Nous allons d'abord décrire les notations utilisées. On travaille sur la matrice de données $A(I,J)$ $(1 \leq I \leq M,\ 1 \leq J \leq N)$. L'algorithme créé les classes en lignes $1,2,\ldots,KR$, les classes en colonnes $1,2,\ldots,KC$ et les blocs de données (sous-matrices) $1,2,\ldots,KD$. Le bloc $K$ est la sous-matrice obtenues avec les lignes dans la classe $IR(K)$ et les colonnes dans la classe $IC(K)$. Les classes en lignes forment un arbre, où la structure est décrite par une paire $MinR(I)$, $MaxR(I)$, qui sont les deux plus grosses classes inclues dans la classe $I$. Dans le cas de classes minimales (indivisibles pour l'algorithme), ces valeurs seront négatives et représenteront le premier et le dernier objet de la classe. Les lignes sont réordonnées durant l'algorithme pour avoir des classes avec des lignes contiguës. Similairement, on aura les classes en colonnes décrites avec $MinC(I)$ et $MaxC(I)$.

Pendant l'algorithme, $VR(K)$ représente la variance moyenne des colonnes dans le bloc $K$, et $VC(K)$ la variance moyenne des lignes dans ce même bloc. $VR(K)$ (respectivement $VC(K)$) est mis à zéro si toutes les variances des colonnes de $K$ (respectivement des lignes de $K$) sont inférieures au seuil. A la fin de l'algorithme,

ces quantités seront donc nulles.

A chaque itération, une variance moyenne est calculée pour la classe $I$ en lignes, en moyennant les $VR(K)$, avec $K$ tel que $VR(K) \neq 0$ et $IR(K) = I$, que l'on notera par la suite $Rsum(I)/RDF(I)$. De la même manière, on définira une variance moyenne $Csum(I)/CDF(I)$ pour la classe $I$ en colonnes, en moyennant les $VC(K)$, avec $K$ tel que $VR(K) \neq 0$ et $IC(K) = I$.

L'algorithme peut donc être décrit de cette manière :

**Étape 1** On fixe un seuil $T$ et on démarre avec les objets dans une seule classe, ainsi que les variables. Ainsi, on initialise les différentes variables ainsi : $Kd = 1$, $MinR(1) = -1$, $MaxR(1) = -n$, $MinC(1) = -1$, $MaxC(1) = -d$. Et $Kr = 1$, $Kc = 1$, $IR(1) = 1$, $IC(1) = 1$.

$$VR(1) = \sum_{j=1}^{d} \sum_{i=1}^{n} \frac{(x_i^j - \bar{x}^j)^2}{(n-1)d} \quad \text{où} \quad \bar{x}^j = \frac{1}{n} \sum_{i=1}^{n} x_i^j$$

$$VC(1) = \sum_{i=1}^{n} \sum_{j=1}^{d} \frac{(x_i^j - \bar{x}_i)^2}{n(d-1)} \quad \text{où} \quad \bar{x}_i = \frac{1}{d} \sum_{j=1}^{d} x_i^j$$

**Étape 2** On cherche ici à calculer les variances moyennes de chaque classe en lignes. On pose $Rsum(Lr) = RDF(Lr) = 0\ \forall Lr$. Et, $\forall Ld$, on a[1] :

$$\begin{aligned} Rsum(IR(Ld)) &+= VR(Ld)\big(MaxC(IC(Ld)) - MinC(IC(Ld)) + 1\big) \\ RDF(IR(Ld)) &+= MaxC(IC(Ld)) - MinC(IC(Ld)) + 1 \end{aligned}$$

**Étape 3** On cherche maintenant à calculer les variances moyennes de chaque classe en colonnes. On pose $Csum(Lc) = CDF(Lc) = 0\ \forall Lc$. Et, $\forall Ld$, on a :

$$\begin{aligned} Csum(IC(Ld)) &+= VC(Ld)\big(MaxR(IR(Ld)) - MinR(IR(Ld)) + 1\big) \\ CDF(IC(Ld)) &+= MaxR(IR(Ld)) - MinR(IR(Ld)) + 1 \end{aligned}$$

**Étape 4** Si la plus grande valeur de $\frac{Rsum(Lr)}{RDF(Lr)}$ et $\frac{Csum(Lc)}{CDF(Lc)}$ est égale à zéro, alors, on stoppe l'algorithme. Sinon, on choisit la classe en lignes ou en colonnes $L$ qui a la variance la plus grande. On combine les blocs $Ld$, pour lesquels $IR(Ld) = L$ et $VR(Ld) \neq 0$ (pour un découpage en lignes) ou pour lesquels $IC(Ld) = L$ et $VC(Ld) \neq 0$ (pour un découpage en colonnes), en une seule matrice. On divise cette matrice en deux (en lignes ou en colonnes) avec l'algorithme *binary splitting*, et on réordonne les lignes ou les colonnes de façon à avoir des lignes ou des colonnes contiguës pour chaque classe. Nous avons donc des lignes ou des colonnes ainsi : $\{I1, I1+1, \ldots, I2\}$ et $\{I2+1, I2+2, \ldots, I3\}$.

1. pour simplifier l'écriture, $a \mathrel{+}= b$ signifie $a = a + b$

**Étape 5** Maintenant, les deux nouvelles classes sont définies de cette manière:

| découpage en lignes | découpage en colonnes |
|---|---|
| $Kr+1$ et $Kr+2$ | $Kc+1$ et $Kc+2$ |
| $MinR(Kr+1) = -I1$ | $MinC(Kc+1) = -I1$ |
| $MinR(Kr+2) = -I2+1$ | $MinC(Kc+2) = -I2+1$ |
| $MaxR(Kr+1) = -I2$ | $MaxC(Kc+1) = -I2$ |
| $MaxR(Kr+2) = -I3$ | $MaxC(Kc+2) = -I3$ |
| $MinR(L) = Kr+1$ | $MinC(L) = Kc+1$ |
| $MaxR(L) = Kr+2$ | $MaxC(L) = Kc+1$ |
| $Kr = Kr+2$ | $Kc = Kc+2$ |
| $\forall Ld,\ IR(Ld) = L,\ VR(Ld) \neq 0$ | $\forall Ld,\ IC(Ld) = L,\ VC(Ld) \neq 0$ |
| $IR(Ld) = Kr+1$ | $IC(Ld) = Kc+1$ |
| $Kd = Kd+1$ | $Kd = Kd+1$ |
| $IR(Kd) = Kr+2$ | $IC(Kd) = Kc+2$ |
| $IC(Kd) = IC(Ld)$ | $IR(Kd) = IR(Ld)$ |

Ensuite, allez à l'étape 2.

Afin d'illustrer le comportement de cet algorithme, nous allons l'appliquer sur l'exemple de (Hartigan, 1975), en détaillant ici aussi chaque étape. Pour ce tableau, on choisit donc un seuil de variance égal à 25. Le déroulement est ainsi montré dans la figure 5.5, et une représentation des résultats est donnée dans la figure 5.6.

Les premières conclusions sont sensiblement les mêmes qu'avec l'algorithme *One-way splitting* (voir 5.4). C'est-à-dire que KY, MD et MO sont similaires pour toutes les années. L'année 1694 présentent des similarités entre SC et LA. Mais de plus, on peut aussi voir que les années 1932, 1936 et 1940 sont similaires pour chacun des états MS, SC et LA, mais les deux premières (1932 et 1936) diffèrent de 1940 pour les états KY, MD et MO. C'est ici que l'on peut voir la différence d'analyse entre les deux algorithmes. Alors que le premier indiquait un similitude entre les trois états MS, SC et LA pour 1932 et une différence pour 1936 et 1940, le second algorithme montre que ces années sont similaires par états, et que chaque état présente un comportement différent des autres.

Nous voyons donc que *Two-way splitting* permet de retrouver une structure en blocs plus fine et plus intéressante que *One-way splitting*. Nous avons donc décidé de nous intéresser dans la suite qu'à ce premier algorithme de classification directe. De plus, traitant les lignes et les colonnes de la même manière, il nous permet d'éviter de se focaliser sur un des deux ensembles.

### 5.2.3 Application sur des données binaires

Pour illustration de cet algorithme, nous avons choisi de l'appliquer sur des données binaires, de taille $n = 20$ et $d = 10$. Afin de prendre le seuil le plus approprié, nous avons testé les différentes valeurs de 0.1 à 1.0, par pas de 0.1. Les moyennes des variances de chaque bloc sont reportées, pour chaque seuil, dans la figure 5.7. A partir de 0.3, *Two-way splitting* ne sépare plus les données, et garde un seul bloc. C'est pourquoi nous avons choisi de retenir un seuil de 0.2.

Tableau initial

| | 32 | 36 | 40 | 60 | 64 | 68 |
|---|---|---|---|---|---|---|
| SC | 2 | 1 | 4 | 49 | 59 | 39 |
| MS | 4 | 3 | 4 | 25 | 87 | 14 |
| LA | 7 | 11 | 14 | 29 | 57 | 23 |
| KY | 40 | 40 | 42 | 57 | 36 | 44 |
| MD | 36 | 37 | 41 | 46 | 35 | 42 |
| MO | 35 | 38 | 48 | 50 | 36 | 45 |

Premier découpage

| | 32 | 36 | 40 | 60 | 68 | 64 |
|---|---|---|---|---|---|---|
| SC | 2 | 1 | 4 | 49 | 39 | 59 |
| MS | 4 | 3 | 4 | 25 | 14 | 87 |
| LA | 7 | 11 | 14 | 29 | 23 | 57 |
| KY | 40 | 40 | 42 | 57 | 44 | 36 |
| MD | 36 | 37 | 41 | 46 | 42 | 35 |
| MO | 35 | 38 | 48 | 50 | 45 | 36 |

Deuxième découpage

| | 32 | 36 | 40 | 60 | 68 | 64 |
|---|---|---|---|---|---|---|
| SC | 2 | 1 | 4 | 49 | 39 | 59 |
| MS | 4 | 3 | 4 | 25 | 14 | 87 |
| LA | 7 | 11 | 14 | 29 | 23 | 57 |
| KY | 40 | 40 | 42 | 57 | 44 | 36 |
| MD | 36 | 37 | 41 | 46 | 42 | 35 |
| MO | 35 | 38 | 48 | 50 | 45 | 36 |

Troisième découpage

| | 32 | 36 | 40 | 60 | 68 | 64 |
|---|---|---|---|---|---|---|
| MS | 4 | 3 | 4 | 25 | 14 | 87 |
| SC | 2 | 1 | 4 | 49 | 39 | 59 |
| LA | 7 | 11 | 14 | 29 | 23 | 57 |
| KY | 40 | 40 | 42 | 57 | 44 | 36 |
| MD | 36 | 37 | 41 | 46 | 42 | 35 |
| MO | 35 | 38 | 48 | 50 | 45 | 36 |

Quatrième découpage

| | 32 | 36 | 40 | 60 | 68 | 64 |
|---|---|---|---|---|---|---|
| MS | 4 | 3 | 4 | 25 | 14 | 87 |
| SC | 2 | 1 | 4 | 49 | 39 | 59 |
| LA | 7 | 11 | 14 | 29 | 23 | 57 |
| KY | 40 | 40 | 42 | 57 | 44 | 36 |
| MD | 36 | 37 | 41 | 46 | 42 | 35 |
| MO | 35 | 38 | 48 | 50 | 45 | 36 |

Cinquième découpage

| | 32 | 36 | 40 | 60 | 68 | 64 |
|---|---|---|---|---|---|---|
| MS | 4 | 3 | 4 | 25 | 14 | 87 |
| SC | 2 | 1 | 4 | 49 | 39 | 59 |
| LA | 7 | 11 | 14 | 29 | 23 | 57 |
| KY | 40 | 40 | 42 | 57 | 44 | 36 |
| MD | 36 | 37 | 41 | 46 | 42 | 35 |
| MO | 35 | 38 | 48 | 50 | 45 | 36 |

Sixième découpage

| | 32 | 36 | 40 | 60 | 68 | 64 |
|---|---|---|---|---|---|---|
| MS | 4 | 3 | 4 | 25 | 14 | 87 |
| SC | 2 | 1 | 4 | 49 | 39 | 59 |
| LA | 7 | 11 | 14 | 29 | 23 | 57 |
| KY | 40 | 40 | 42 | 57 | 44 | 36 |
| MD | 36 | 37 | 41 | 46 | 42 | 35 |
| MO | 35 | 38 | 48 | 50 | 45 | 36 |

Dernier découpage

| | 32 | 36 | 40 | 60 | 68 | 64 |
|---|---|---|---|---|---|---|
| MS | 4 | 3 | 4 | 25 | 14 | 87 |
| SC | 2 | 1 | 4 | 49 | 39 | 59 |
| LA | 7 | 11 | 14 | 29 | 23 | 57 |
| KY | 40 | 40 | 42 | 57 | 44 | 36 |
| MD | 36 | 37 | 41 | 46 | 42 | 35 |
| MO | 35 | 38 | 48 | 50 | 45 | 36 |

FIG. 5.5 – *Déroulement de l'algorithme Two-way splitting, avec les découpages successifs*

| | 32 | 36 | 40 | 60 | 68 | 64 |
|---|---|---|---|---|---|---|
| MS | 4 | | | 25 | 14 | 87 |
| SC | 2 | | | 49 | 39 | 58 |
| LA | 11 | | | 26 | | |
| KY | | | | | | |
| MD | 38 | | 44 | 50 | 43 | 36 |
| MO | | | | | | |

FIG. 5.6 – *Représentation réduite des votes républicains, basée sur les résultats de l'algorithme Two-way splitting, avec les moyennes pour chaque bloc.*

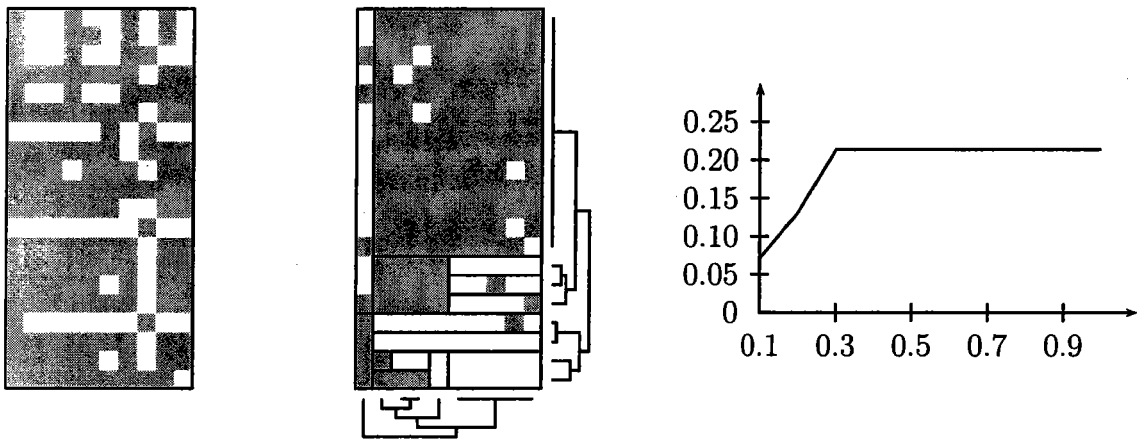



FIG. 5.7 – *Matrice initiale et réordonnée selon les résultats de Two-way splitting (avec les arbres hiérarchiques associés) et variances moyennes pour chaque seuil, pour le tableau binaire.*

Dans la figure 5.7, nous présentons la matrice réordonnée selon les blocs et les hiérarchies en lignes et en colonnes fournis par *Two-way splitting*. Il est très clair que cet algorithme propose un découpage en blocs clair. Nous allons maintenant étudier plus en détail le comportement de cette méthode sur des données réelles.

### 5.2.4 Exemple des *Poisson d'Amiard*

#### Présentation

Afin de comprendre les différences que l'on peut obtenir entre *Croeuc* et l'algorithme *Two-way splitting*, nous avons appliqué les deux méthodes sur l'exemple des poissons d'Amiard.

24 mulets (sorte de rougets) ont été répartis dans trois aquariums radio-contaminés de façon identique. A ces trois aquariums correspondent des durées de contact avec le polluant radio-actif différentes. Les poissons ont été répartis de la manière suivante :

- de 1 à 8 dans le premier aquarium,
- de 9 à 16 dans le deuxième,
- de 17 à 23 dans le troisième.

16 caractéristiques ont été mesurées, et que l'on peut séparer en deux groupes distincts :

1. Caractéristiques de radio-activité, de 1 à 9, avec dans l'ordre : radio-activité des yeux, des branchies, des opercules, des nageoires, du foie, du tube digestif, des reins, des écailles et des muscles.
2. Caractéristiques de taille, de 10 à 16, avec dans l'ordre : poids, taille, longueur, longueur standard, largeur de la tête, largeur, largeur du museau et diamètre des yeux.

### Résultats

Avec la partition en bloc et les moyennes de chacun de ses blocs (voir figure 5.8), ainsi qu'avec les arbres de découpages successifs en lignes et en colonnes présentés dans les figures 5.9 et 5.10, nous pouvons faire les observations suivantes :

- Pour les poissons 1 à 4 (ainsi que pour le poisson 20), qui ont les caractéristiques de radio-activités les plus faibles, on note qu'ils sont plus grands en moyenne que le reste des poissons.
- Par contre, les poissons 6, 7, 9, 5 et 8, alors qu'ils présentent une radio-activité moins importante que la moyenne, ont des tailles inférieures aux autres poissons.
- Les poissons 12, 10, 14, 18 et 19 présentent le profil inverse, ils montrent une radio-activité assez importante et un développement physique en retrait par rapport aux autres poissons.
- Enfin, les poissons 17, 22 et 21, bien que présentant des caractéristiques de radio-activité assez élevées, ont des tailles assez moyennes. Et à l'inverse, les poissons 12, 10 et 14, ayant une radio-activité moyenne, ont des tailles plutôt faibles. Et les poissons 11, 23, 13, 15 et 16 ont des tailles moyennes et une radio-activité plutôt faible.
- En ce qui concerne les variables, l'algorithme arrive à séparer d'un côté les variables concernant le physique (de 10 à 16), en leur adjoignant la variable 7 (radio-activité des reins), et de l'autre les caractéristiques de radio-activité (de 1 à 6, plus 8 et 9), au sein desquelles il fait encore quelques découpages.

Il est ainsi difficile de conclure que la radio-activité a une influence directe sur le développement des poissons, dans cet exemple. Mais une légère tendance permet de montrer que les poissons les moins exposés semblent être plus grands que les autres.

### Comparaison avec *Croeuc*

Afin de mieux cerner cet algorithme, nous allons comparer ces résultats à ceux obtenus avec *Croeuc* dans un premier temps, puis avec ceux de l'Analyse des Composantes Principales (ou ACP) ensuite. Voici les classes en lignes et en colonnes obtenues avec *Croeuc* :

- En lignes
  - $\mathbf{z}_1$ : {5, 6, 7, 8, 14}
  - $\mathbf{z}_2$ : {9, 10, 11, 12, 13, 15, 16, 23}
  - $\mathbf{z}_3$ : {17, 18, 19, 21, 22}
  - $\mathbf{z}_4$ : {1, 2, 3, 4, 20}
- En colonnes
  - $\mathbf{w}_1$ : {1, 2, 3, 4, 8}
  - $\mathbf{w}_2$ : {10, 11, 12, 13, 14, 15, 16}
  - $\mathbf{w}_3$ : {5, 6, 7, 9}

*Croeuc* retrouve plutôt bien la classe des poissons du deuxième aquarium, en leur ajoutant le poisson 23 (classe 2). Par contre, il découpe le premier groupe en deux (classe 1 et 4), en leur ajoutant un poisson à chaque (respectivement 14 et 20).

Enfin, la classe 3 correspond assez bien au troisième groupe de poissons. Pour les variables, il retrouve parfaitement l'ensemble des caractéristiques de forme dans la classe 2, et découpe les caractéristiques de radio-activité en 2 classes (classes 1 et 3).

Les plus importantes différences entre l'algorithme *Two-way splitting* et *Croeuc* résident dans l'association que fait le premier entre les poissons 6, 7 avec le 9, et dans le regroupement des poissons 10 et 14, plus dans une moindre mesure les poissons 5, 8, 12 et 18.

Au vu des deux premiers plans factoriels pour les lignes (figures 5.11 et 5.12), résultant d'une ACP (analyse en composantes principales), ainsi que le premier cercle de corrélation pour les colonnes (figure 5.13), on remarque que l'axe 1 est plutôt lié aux caractéristiques de taille. Les plus grands se trouvent donc dans la partie droite du plan (poissons 1 à 4 et 20). L'axe 2 est lui plus lié aux caractéristiques de radio-activité (avec une opposition à la variable 6 à noter). Cette opposition peut se retrouver avec les résultats de Two-way. En effet, on voit que cette variable est rapidement écartée de l'ensemble des autres variables de caractéristiques. Ainsi, dans le tableau résultat (présenté dans la figure 5.8), on voit qu'elle présente un comportement assez particulier par rapport aux autres. De plus, on voit très bien que la variable 7, pourtant caractéristique de radio-activité des reins, est très proche des variables de taille, ce que n'arrive pas à détecter *Croeuc*.

Si l'on s'intéresse maintenant aux plans factoriels 1 et 2, on peut constater les choses suivantes, qui vont dans le sens des résultats de *Two-way splitting*:

- Le poisson 20 est bien proche des poissons 1 à 4, alors qu'ils n'étaient pas dans le même aquarium.
- Les poissons 6, 7 et 9 sont très proches, surtout sur le deuxième plan factoriel. Il est donc normal que *Two-way splitting* les rassemble, au contraire de *Croeuc*.
- Les poissons 11 et 23 sont tous les deux proches de 6, 7 et 9. Et le groupe {13, 15, 16} est assez visible, surtout sur le plan 2. Tous ces poissons forment de plus une classe assez compacte.
- Les poissons 5 et 8 sont à côté l'un de l'autre, et proche de 12, 10 et 14.
- Les poissons 18 et 19 sont seuls, ce que retrouve assez bien *Two-way splitting*.

Pour conclure, grâce aux plans factoriels, on comprend la principale différence entre *Croeuc* et *Two-way splitting*:

- *Croeuc* cherche à faire des classes plutôt sphériques, et linéairement séparable sur le plan factoriel. Pour cet algorithme, grâce à l'approche mélange, nous verrons qu'il y a une hypothèse implicite liée à la distribution des données dans chaque bloc.
- *Two-way splitting* découpe les groupes en deux étape par étape, ce qui justifie la séparation du groupe de poissons du premier et du deuxième aquarium en deux groupes distincts. La structure ainsi obtenue est plus fine et plus riche d'enseignement, mais parfois complexe à analyser.

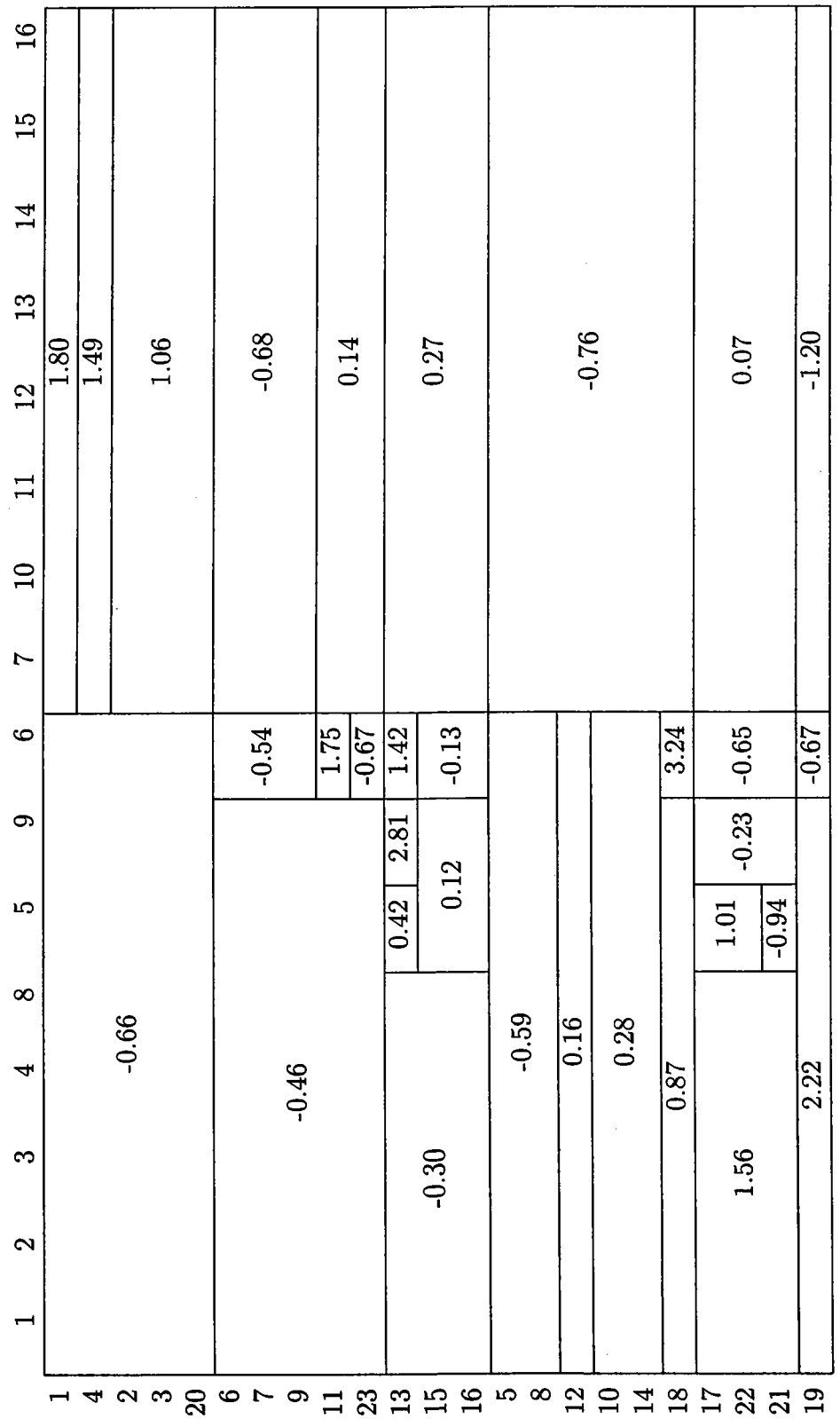



FIG. 5.8 – *Moyennes par blocs des résultats de Two-way splitting sur les poissons d'Amiard standardisés, avec un seuil égal à 1.*

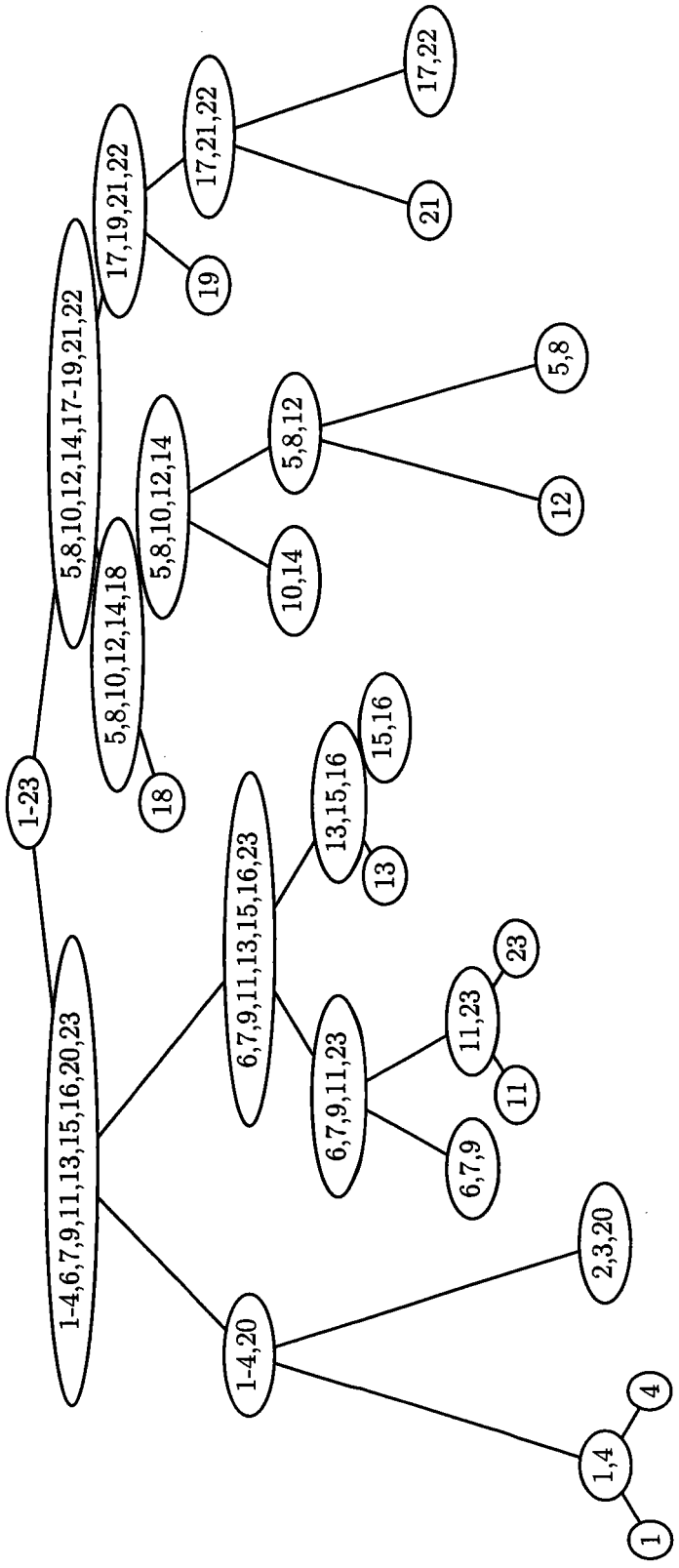



FIG. 5.9 – *Découpages successives des classes en lignes pour les poissons d'Amiard, avec Two-way splitting. Pour éviter de surcharger, nous avons simplifier la notation. 1-23 signifie les poissons de 1 à 23.*

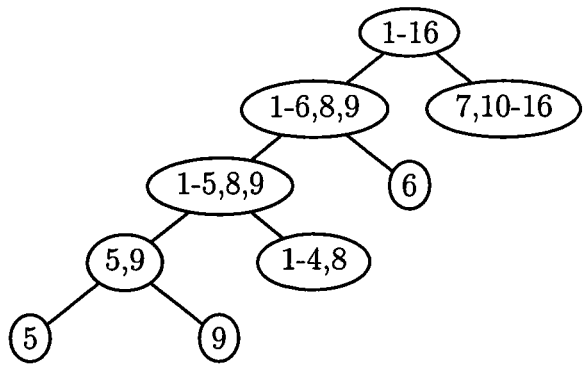



FIG. 5.10 – *Découpages successives des classes en colonnes pour les poissons d'Amiard, avec Two-way splitting. Pour éviter de surcharger, nous avons simplifier la notation. 1-16 signifie les variables de 1 à 16.*

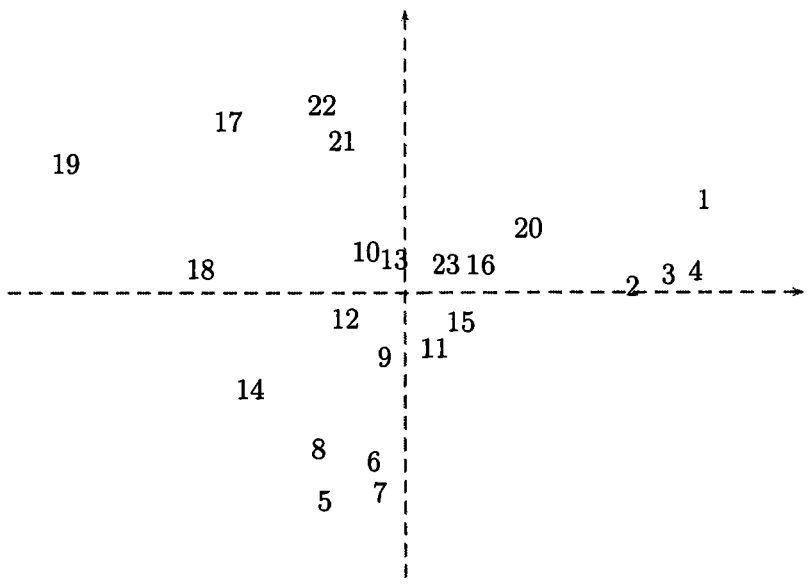



FIG. 5.11 – *Plan factoriel (1,2) des lignes (47 %, 23 %) pour les poissons d'Amiard*

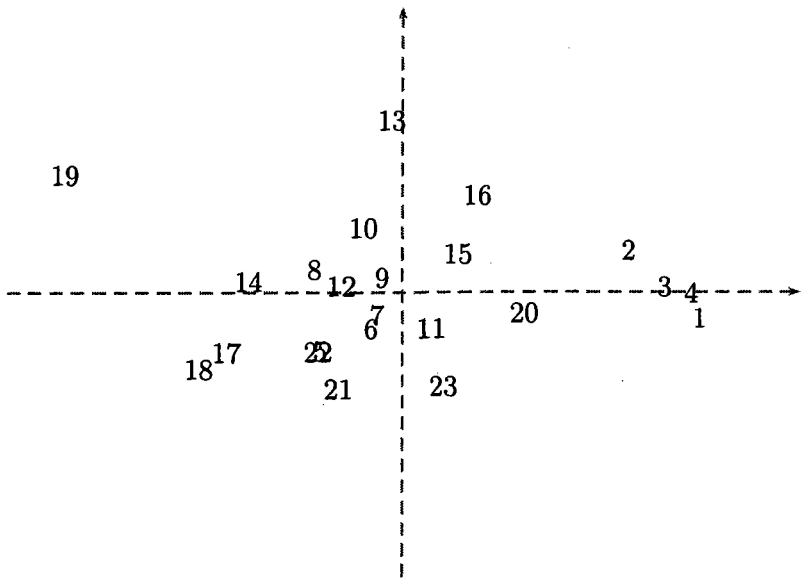



FIG. 5.12 – *Plan factoriel (1,3) des lignes (47 %, 8 %) pour les poissons d'Amiard*

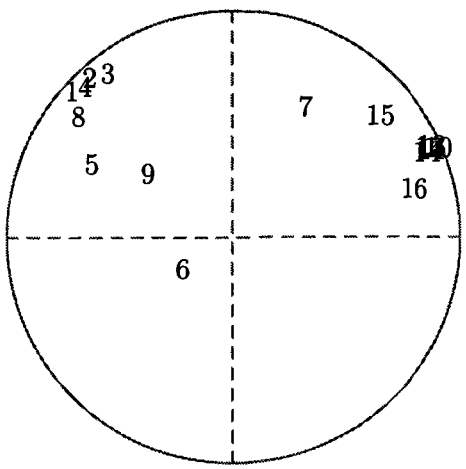



FIG. 5.13 – *Cercle de corrélations (1,2) des colonnes (47 %, 23 %) pour les poissons d'Amiard. Les variables 10, 11, 12, 13 et 14 ont sensiblement les mêmes coordonnées sur le cercle des corrélations.*

# Chapitre 6

# Modèles de mélange croisé et application

Les méthodes proposées par Govaert (1983), présentées dans le chapitre précédent, reposent sur des critères métriques différents suivant le type des données. De la même manière que pour le cas de la classification simple, il est naturel de savoir si ces critères sont associés ou non à des modèles de mélange. En effet, cette extension aux modèles de mélange nous permettrait de mieux appréhender les résultats fournis par ces algorithmes, et nous donnerait ainsi une justification théorique de ceux-ci.

En utilisant l'approche probabiliste, nous présentons ici un modèle de mélange croisé pour résoudre le problème de la classification en blocs. En restreignant les modèles, nous parvenons ainsi à retrouver les critères optimisés par les algorithmes *Croeuc* et *Crobin*, et à leur donner un sens probabiliste. De plus, nous introduisons une nouvelle méthode de classification croisée hiérarchique, noté *HBCM*, qui cherche simultanément une hiérarchie en ligne et une hiérarchie en colonne.

Enfin, dans l'optique Fouille de Données qui nous motive, nous étudierons deux stratégies de combinaison d'algorithmes de type classification croisée, sur des données simulées. En premier lieu, nous aborderons la possibilité d'utiliser conjointement les algorithmes CEM croisé et *Two-way splitting*. Ensuite, nous aborderons la possibilité d'étendre la méthode mixte de Wong (1977) au cas croisé, avec modèle de mélange. Nous terminons par une comparaison de ces deux approches sur un même tableau de données réelles.

## 6.1 Modèle de mélange croisé

### 6.1.1 Notations

Dans la suite, la matrice des données est notée et définie par $\mathbf{x} = \{(x_i^j); i \in I \text{ and } j \in J\}$, où $I$ est l'ensemble des $n$ objets (individus, lignes) et $J$ est l'ensemble des $d$ variables (attributs, colonnes). Notre objectif est de trouver pour $s$ et $m$ supposés connus un couple de partitions $(\mathbf{z},\mathbf{w})$, avec $\mathbf{z} = (z_1,...,z_s)$ une partition de l'ensemble $I$ en $s$ classes et $\mathbf{w} = (w_1,...,w_m)$ une partition de l'ensemble $J$ en $m$ classes. Par convenance, nous adopterons les notations suivantes pour une partition $\mathbf{z}$ et qui seront également valables pour une partition $\mathbf{w}$. Une partition $\mathbf{z} = (z_1, \ldots, z_s)$

peut être aussi représentée par $\mathbf{z} = (\mathbf{z}_1, \ldots, \mathbf{z}_n)$, où $\mathbf{z}_i \in \{1, \ldots, g\}$ indique la classe de l'objet $i$. En outre, $\mathbf{z}_i = k$ peut être représenté par $\mathbf{z}_i = (z_{i1}, \ldots, z_{is})$ avec $z_{ik} = 1$ et $z_{ik} = 0$ sinon. Ainsi la classe $z_k$ correspond à l'ensemble des objets $i$ tel que $\mathbf{z}_i = k$ ou $z_{ik} = 1$.

### 6.1.2 Extension

Plusieurs auteurs (Bock, 1979; Govaert, 1983; Hartigan, 1972; Hartigan, 1975) ont déjà étudié le cas de la classification croisée, en abordant le problème avec l'optimisation d'un critère métrique. Se basant sur des distances ou mesures de dissimilarité simples, ils ont ainsi pu proposé un certain nombre d'algorithmes (dont *Croeuc*, *Crobin*, *Croki2*, *One-way* et *Two-way splitting* que nous avons précédemment décrits). L'intérêt des modèles de mélange pour la redéfinition de critères métriques dans la classification simple nous permet d'envisager de faire la même opération dans le cas croisé.

Robardet (2002) a proposé une méthode de bi-partitionnement non paramètrique, nommée Bi-Clust. Ici, l'objectif est de rechercher un ensemble de *bi-clusters* tels que le nombre de relations entre un objet et une modalité d'une variable du *bi-cluster* soit maximum, et tel que le nombre de relations entre un objet et une modalité d'une variable de *bi-clusters* différents soit minimum. Notons que dans l'algorithme proposé, les nombres de classes en lignes et en colonnes sont les mêmes.

Dans l'objectif d'associer un critère probabiliste à un critère métrique tel que celui de *Croeuc* (ou encore celui de *Crobin*), Bencheikh (1992) a proposé un modèle de mélange croisé en considérant le produit cartésien des deux ensembles $I$ et $J$ comme un échantillon de taille $Card(I) \times Card(J)$ d'une variable à valeurs dans l'ensemble des réels $\mathbb{R}$. Puis, elle a étudié le problème du partitionnement sous l'approche classification dans le cas où les proportions du modèle sont supposées égales. Bien que la définition du modèle proposé par l'auteur repose sur l'écriture du modèle de mélange classique, elle ne présente pas de lien avec ce dernier. Dans le même sens, Bzioui (1999) a présenté un autre modèle de mélange en considérant que le tableau de données est un échantillon de taille 1. Malheureusement, pour la maximisation de ces critères, l'auteur a utilisé des algorithmes de type EM. Dans ces deux approches, à partir des modèles, les auteurs ne justifient pas l'utilisation de tels algorithmes.

Enfin, récemment, Govaert et Nadif (2003) proposent un modèle à partir duquel ils ont proposé une extension de *Croeuc* et de *Crobin*. Dans la suite de ce travail, nous nous inscrirons dans cette démarche.

### 6.1.3 Définition du modèle

Nous présentons ici la classification croisée sous l'approche modèle de mélange. Govaert et Nadif (1993) considèrent que la distribution des données est représentée par la fonction de densité de probabilité suivante

$$\varphi(\mathbf{x},\theta) = \sum_{\mathbf{u} \in U} p(\mathbf{u};\theta)\varphi(\mathbf{x}|\mathbf{u};\theta), \tag{6.1}$$

où $U$ correspond à l'ensemble de toutes les partitions de l'ensemble $I \times J$. Si nous restreignons ce modèle à l'ensemble $\mathcal{Z} \times \mathcal{W}$ défini par le produit cartésien des ensembles des partitions de $I$ en $s$ classes et de l'ensemble $J$ en $m$ classes, qui seront supposés indépendantes, nous obtenons la décomposition suivante

$$\varphi(\mathbf{x};\theta) = \sum_{(\mathbf{z},\mathbf{w})\in\mathcal{Z}\times\mathcal{W}} p(\mathbf{z};\theta)p(\mathbf{w};\theta)\varphi(\mathbf{x}|\mathbf{z},\mathbf{w};\theta). \tag{6.2}$$

En supposant que les $x_i^j$ seront indépendants sachant que $\mathbf{z}_i$ et $\mathbf{w}_j$ sont fixés et en notant $\theta = (\mathbf{p},\mathbf{q},\alpha)$ avec $\mathbf{p} = (p_1,\ldots,p_s)$ et $\mathbf{q} = (q_1,\ldots,q_m)$, nous obtenons

$$\begin{aligned} p(\mathbf{z};\theta) &= \prod_{i=1}^{n} p_{\mathbf{z}_i}, \\ p(\mathbf{w};\theta) &= \prod_{j=1}^{d} q_{\mathbf{w}_j}, \\ \varphi(\mathbf{x}|\mathbf{z},\mathbf{w};\theta) &= \prod_{i,j} \varphi_{\mathbf{z}_i\mathbf{w}_j}(x_i^j;\alpha), \end{aligned}$$

où $\varphi_{k\ell}(x,\alpha)$ est une fonction de densité de probabilité définie sur $\mathbb{R}$ de paramètre $\alpha$. Ainsi, nous obtenons le modèle croisé suivant :

$$\varphi(\mathbf{x};\theta) = \sum_{(\mathbf{z},\mathbf{w})\in\mathcal{Z}\times\mathcal{W}} \prod_i p_{\mathbf{z}_i} \prod_j q_{\mathbf{w}_j} \prod_{i,j} \varphi_{\mathbf{z}_i\mathbf{w}_j}(x_i^j;\alpha),$$

où les densités $\varphi_{k\ell}$ appartiennent à la même famille de densités, les paramètres $p_k$ et $q_\ell$ sont les probabilités qu'une ligne et une colonne appartiennent respectivement à la $k$ième classe de $\mathbf{z}$ et à la $\ell$ième classe de $\mathbf{w}$.

### 6.1.4 Approche Classification

Pour traiter le problème de la classification croisée, nous proposons d'utiliser le modèle proposé précédemment. Comme pour le modèle de mélange classique, ce problème peut être traité soit sous l'approche estimation soit sous l'approche classification (Symons, 1981). Ici nous nous intéresserons à la deuxième approche qui consiste à maximiser la log-vraisemblance complétée associée à notre modèle et qui s'écrit:

$$L_c(\mathbf{z},\mathbf{w},\theta) = L(\mathbf{x},\mathbf{z},\mathbf{w};\theta) = \log\left(p(\mathbf{z};\theta)p(\mathbf{w};\theta)\varphi(\mathbf{x}|\mathbf{z},\mathbf{w};\theta)\right)$$

ou encore

$$\begin{aligned} L_c(\mathbf{z},\mathbf{w},\theta) &= \sum_{i=1}^{n}\sum_{k=1}^{s} z_{ik}\log(p_k) + \sum_{j=1}^{d}\sum_{\ell=1}^{m} w_{j\ell}\log(q_\ell) \\ &+ \sum_{i=1}^{n}\sum_{j=1}^{d}\sum_{k=1}^{s}\sum_{\ell=1}^{m} z_{ik}w_{j\ell}\log\varphi_{k\ell}(x_i^j;\alpha). \end{aligned}$$

Comme la maximisation de $L_c(\mathbf{z},\mathbf{w},\theta)$ ne peut se faire directement, pour résoudre ce problème nous proposons d'utiliser une extension de l'algorithme CEM (Celeux et Diebolt, 1992) qui est une variante classifiante de l'algorithme EM.

### 6.1.5 L'algorithme CEM croisé

Pour maximiser $L_c(\mathbf{z},\mathbf{w},\theta) = L_c(\mathbf{z},\mathbf{w},(\mathbf{p},\mathbf{q},\alpha))$, nous proposons de maximiser alternativement la log-vraisemblance complétée avec $\mathbf{w}$ et $\mathbf{q}$ fixés puis avec $\mathbf{z}$ et $\mathbf{p}$ fixés. Si nous supposons en plus que la fonction de densité de probabilité $\varphi_{k\ell}$ admet une statistique exhaustive $u_i^\ell$, en notant $\mathbf{u}_i = (u_i^1,\ldots,u_i^m)$ et $\#\mathbf{w} = (\#w_1,\ldots,\#w_m)$, nous pouvons écrire

$$L_c(\mathbf{z},\mathbf{w},\theta) = L_c(\mathbf{z},\theta/\mathbf{w}) + g(\mathbf{x},\mathbf{w},\mathbf{q}),$$

où

$$L_c(\mathbf{z},\theta/\mathbf{w}) = \sum_{i=1}^{n}\sum_{k=1}^{s} z_{ik}\log\left(p_k\psi_k(\mathbf{u}_i;\alpha,\#\mathbf{w})\right),$$

$\psi_k(\mathbf{u}_i;\alpha,\#\mathbf{w})$ est la fonction de densité de probabilité de $\mathbf{u}_i$ et $g(\mathbf{x},\mathbf{w},\mathbf{q})$ est une fonction qui ne dépend pas de $\mathbf{z}$. Ainsi, la log-vraisemblance conditionnelle $L_c(\mathbf{z},\theta/\mathbf{w})$ correspond à la log-vraisemblance associée au modèle de mélange classique, qui s'écrit

$$\sum_{k=1}^{s} p_k\psi_k(\mathbf{u};\alpha,\#\mathbf{w}). \tag{6.3}$$

De cette façon, la maximisation de $L_c(\mathbf{z},\mathbf{w},\theta)$ pour $\mathbf{w}$ et $\mathbf{q}$ fixés est équivalente à la maximisation de la log-vraisemblance conditionnelle $L_c(\mathbf{z},\theta/\mathbf{w})$, qui peut être réalisée par l'algorithme CEM appliqué au modèle de mélange (6.3).

Comme précédemment, la log-vraisemblance complétée peut aussi s'écrire:

$$L_c(\mathbf{z},\mathbf{w},\theta) = L_c(\mathbf{w},\theta/\mathbf{z}) + g(\mathbf{x},\mathbf{z},\mathbf{p}),$$

où

$$L_c(\mathbf{w},\theta/\mathbf{z}) = \sum_{j=1}^{d}\sum_{\ell=1}^{m} w_{j\ell}\log(q_\ell\psi_\ell(\mathbf{v}^j;\alpha,\#\mathbf{z})).$$

Ici, $\mathbf{v}^j = (v_1^j,\ldots,v_s^j)$, où $v_i^\ell$ est une fonction d'une statistique exhaustive de $\{x_i^j; i \in z_k\}$, $\#\mathbf{z} = (\#z_1,\ldots,\#z_s)$ et $g(\mathbf{x},\mathbf{z},\mathbf{p})$ ne dépend pas de $\mathbf{w}$.

Cette log-vraisemblance conditionnelle $L_c(\mathbf{w},\theta/\mathbf{z})$, qui correspond à la log-vraisemblance associée au modèle de mélange classique

$$\sum_{\ell=1}^{m} q_\ell\psi_\ell(\mathbf{v};\alpha,\#\mathbf{z})$$

peut être maximisée par l'algorithme CEM appliqué à ce modèle de mélange.

#### Algorithme

Maintenant, nous pouvons décrire les différentes étapes de l'algorithme.

1. Initialisation avec une solution $(\mathbf{z}^0,\mathbf{w}^0,\alpha^0)$.
2. Calcul de $(\mathbf{z}^{(c+1)},\mathbf{w}^{(c+1)},\alpha^{(c+1)})$ à partir de $(\mathbf{z}^{(c)},\mathbf{w}^{(c)},\alpha^{(c)})$:
   (a) Calcul de $(\mathbf{z}^{(c+1)},\mathbf{p}^{(c+1)},\alpha')$ avec l'algorithme CEM sur les données $(\mathbf{u}_1,\ldots,\mathbf{u}_n)$ à partir de $(\mathbf{z}^{(c)},\mathbf{p}^{(c)},\alpha^{(c)})$.
   (b) Calcul de $(\mathbf{w}^{(c+1)},\mathbf{q}^{(c+1)},\alpha^{(c+1)})$ avec l'algorithme CEM sur les données $(\mathbf{v}^1,\ldots,\mathbf{v}^d)$ à partir de $(\mathbf{w}^{(c)},\mathbf{q}^{(c)},\alpha')$.
3. Répéter les deux étapes jusqu'à la convergence.

### 6.1.6 Application à des données quantitatives

Ici, nous détaillons le modèle de mélange croisé pour des données continues (Govaert et Nadif, 2002). Dans cette situation, la distribution de probabilité $\varphi_{k\ell}(x_i^j;\alpha)$ peut être choisie comme une distribution Gaussienne :

$$\varphi_{k\ell}(x_i^j;\alpha) = \Phi(x_i^j;g_k^\ell,{\sigma_0}^2) = \frac{1}{\sqrt{2\pi{\sigma_0}^2}}\exp\left(-\frac{1}{2{\sigma_0}^2}(x_i^j - g_k^\ell)^2\right)$$

où le paramètre $\alpha = (g_1^1,g_1^2,\ldots,g_s^m;{\sigma_0}^2)$. Dans la suite, nous supposons que le paramètre réel $\sigma_0$ est fixé. Un choix judicieux de ce paramètre est discuté plus tard. Ainsi, la log-vraisemblance complétée est donnée par

$$\begin{aligned} L_c(\mathbf{z},\mathbf{w},\theta) &= \sum_{k=1}^{s} \#z_k \log p_k + \sum_{\ell=1}^{m} \#w_\ell \log q_\ell - nd\log(\sigma_0\sqrt{2\pi}) \\ &- \frac{1}{2{\sigma_0}^2} W(\mathbf{z},\mathbf{w},\mathbf{g}), \end{aligned} \tag{6.4}$$

où

$$W(\mathbf{z},\mathbf{w},\mathbf{g}) = \sum_{k=1}^{s}\sum_{\ell=1}^{m}\sum_{i\in z_k}\sum_{j\in w_\ell}(x_i^j - g_k^\ell)^2.$$

La maximisation de $L_c(\mathbf{z},\mathbf{w},\theta)$ est obtenue en maximisant les log-vraisemblances complétées conditionnelles $L_c(\mathbf{z},\theta/\mathbf{w})$ et $L_c(\mathbf{w},\theta/\mathbf{z})$. Ces log-vraisemblances sont associées au modèle de mélange classique, où nous avons

$$\begin{cases} \psi_k(\mathbf{u}_i;\alpha,\#\mathbf{w}) = \displaystyle\prod_{\ell=1}^{m}\Phi(u_i^\ell;g_k^\ell,\sigma_0^2/\#w_\ell) \quad \text{avec} \quad u_i^\ell = \sum_{j=1}^{d}\frac{w_{j\ell}x_i^j}{\#w_\ell} \\ \psi^\ell(\mathbf{v}^j;\alpha,\#\mathbf{z}) = \displaystyle\prod_{k=1}^{s}\Phi(v_k^j;g_k^\ell,\sigma_0^2/\#z_k) \quad \text{avec} \quad v_k^j = \sum_{i=1}^{n}\frac{z_{ik}x_i^j}{\#z_k}. \end{cases}$$

#### L'algorithme

À l'étape 2(a), l'algorithme CEM calcule $(\mathbf{z}^{(c+1)},\mathbf{p}^{(c+1)},\alpha')$, à partir de $(\mathbf{z}^{(c)},\mathbf{p}^{(c)},\alpha^{(c)})$ selon les étapes suivantes :

**E-step** : Calcul des probabilités a posteriori

$$t_{ik} \propto p_k \prod_{\ell=1}^{m}\exp\left(-\frac{\#w_\ell}{2}\left(\frac{u_i^\ell - g_k^\ell}{\sigma_0}\right)^2\right).$$

**C-step** : La partition $\mathbf{z}^{(c+1)}$ est définie en maximisant $t_{ik}$, ce qui est équivalent à minimiser

$$\sum_{\ell=1}^{m}\frac{\#w_\ell}{2{\sigma_0}^2}(u_i^\ell - g_k^\ell)^2 - \log(p_k).$$

**M-step** : Calcul des paramètres $(\mathbf{p}^{(c+1)},\alpha')$, ce qui revient à faire, pour la classe $k$ :

$$\begin{cases} p_k^{(c+1)} = \dfrac{\#z_k^{(c+1)}}{n} \\ (g_k^\ell)' = \dfrac{\sum_{i\in z_k^{(c+1)}} u_i^\ell}{\#z_k^{(c+1)}}. \end{cases}$$

Alternativement, et de la même manière, une itération de l'algorithme CEM pour l'étape 2(b) de l'algorithme croisé, qui calcule $(\mathbf{w}^{(c+1)},\mathbf{q}^{(c+1)},\alpha^{c+1})$ à partir de $(\mathbf{w}^{(c)},\mathbf{q}^{(c)},\alpha')$, sera décrite ainsi :

**E-step** : Calcul des probabilités a posteriori

$$t_{j\ell} \propto q_\ell \prod_{k=1}^{s} \exp\left(-\frac{\#z_k}{2}\left(\frac{v_k^j - g_k^\ell}{\sigma_0}\right)^2\right).$$

**C-step** : La partition $\mathbf{z}^{(c+1)}$ est définie en maximisant $t_{j\ell}$, qui est équivalent à minimiser

$$\sum_{k=1}^{s} \frac{\#z_k}{2{\sigma_0}^2}(v_k^j - g_k^\ell)^2 - \log(q_\ell).$$

**M-step** : Calculer les paramètres $(q_\ell^{(c+1)},\alpha^{(c+1)})$, ce qui revient à calculer, pour la classe $\ell$ :

$$\begin{cases} q_\ell^{(c+1)} = \dfrac{\#w_\ell^{(c+1)}}{n} \\ (g_k^\ell)^{(c+1)} = \dfrac{\sum_{j\in w_\ell^{(c+1)}} v_k^j}{\#w_\ell^{(c+1)}}. \end{cases}$$

**L'algorithme** *Croeuc*

Quand les proportions sont supposées égales, il est facile de voir que maximiser $L_c(\mathbf{z},\mathbf{w},\theta)$ est équivalent à minimiser $W(\mathbf{z},\mathbf{w},\mathbf{g})$ qui est utilisé dans d'autres méthodes (Hartigan, 1975; Bock, 1979; Govaert, 1983; Govaert, 1995). Et nous avons une signification probabiliste du critère optimisé par *Croeuc* (Govaert, 1983). Ainsi, il est aisé de prouver que la maximisation des deux log-vraisemblances complétées conditionnelles par l'algorithme CEM est équivalent à la minimisation de

$$\begin{cases} W(\mathbf{z},\mathbf{g}|\mathbf{w}) &= \sum_{k=1}^{s}\sum_{i\in z_k}\sum_{\ell=1}^{m} \#w_\ell(u_i^\ell - g_k^\ell)^2 \\ W(\mathbf{w},\mathbf{g}|\mathbf{z}) &= \sum_{\ell=1}^{m}\sum_{j\in w_\ell}\sum_{k=1}^{s} \#z_k(v_k^j - g_k^\ell)^2. \end{cases}$$

Ces critères sont utilisés par l'algorithme *Croeuc* et sont optimisés par l'algorithme $k$-means, qui est une version restreinte de CEM dans ce cas. Ainsi, notre approche par modèle de mélange nous permet de proposer un algorithme de classification croisée et de donner un sens probabiliste au critère optimisé par *Croeuc*.

TAB. 6.1 – *Décomptes d'objets et de variables mal-classés pour CEM croisé et double k-means, pour les trois tableaux de données continues simulées (*$\mathbf{z}$ *et* $\mathbf{w}$ *représente les partitions simulées).*

| | Tableau 1 | | Tableau 2 | | Tableau 3 | |
|---|---|---|---|---|---|---|
| | $\mathbf{z}$ | $\mathbf{w}$ | $\mathbf{z}$ | $\mathbf{w}$ | $\mathbf{z}$ | $\mathbf{w}$ |
| CEM croisé | 0 | 0 | 5 | 0 | 0 | 0 |
| double $k$-means | 0 | 0 | 63 | 33 | 90 | 0 |

En effet, celui-ci cherche donc des classes sphériques et de même taille, comme nous l'avons remarquer dans le chapitre précédent. A partir (6.4), pour estimer $\sigma_0$, nous prenons $\frac{W(\mathbf{z},\mathbf{w},\mathbf{g})}{nd}$ qui maximise $L_c(\mathbf{z},\mathbf{w},\theta)$. Pour cela, il est suffisant de lancer l'algorithme *Croeuc*, avant d'appliquer CEM croisé.

### 6.1.7 Comparaison de CEM croisé et double $k$-means

Nous allons ici voir l'intérêt d'utiliser une approche croisée, avec CEM. Ce que nous appelons par la suite double $k$-means est la double utilisation de cet algorithme, une fois pour avoir les classes en colonnes, puis pour avoir les classes en lignes. Cette méthode correspond donc à l'utilisation d'algorithmes de classification simple. Pour ceci, nous avons simuler trois tableaux différents avec $n = 1000$, $d = 500$, $s = 7$ et $m = 6$. La différence entre les deux premiers tableaux se situe sur les centres choisis, décrits ci-après :

$$\begin{pmatrix} 5.0 & 5.0 & 5.0 & 5.0 & 5.0 & 5.0 \\ 5.0 & 5.0 & 5.0 & 5.0 & 5.0 & 3.5 \\ 5.0 & 5.0 & 5.0 & 5.0 & 3.5 & 3.5 \\ 5.0 & 5.0 & 5.0 & 3.5 & 3.5 & 3.5 \\ 5.0 & 5.0 & 3.5 & 3.5 & 3.5 & 3.5 \\ 5.0 & 3.5 & 3.5 & 3.5 & 3.5 & 3.5 \\ 3.5 & 3.5 & 3.5 & 3.5 & 3.5 & 3.5 \end{pmatrix} \quad \text{et} \quad \begin{pmatrix} 5.0 & 5.0 & 5.0 & 5.0 & 5.0 & 5.0 \\ 5.0 & 5.0 & 5.0 & 5.0 & \mathbf{4.0} & 3.5 \\ 5.0 & 5.0 & 5.0 & 5.0 & 3.5 & 3.5 \\ 5.0 & 5.0 & 5.0 & 3.5 & 3.5 & 3.5 \\ 5.0 & 5.0 & 3.5 & 3.5 & 3.5 & 3.5 \\ 5.0 & 3.5 & 3.5 & 3.5 & 3.5 & 3.5 \\ 3.5 & 3.5 & 3.5 & 3.5 & 3.5 & 3.5 \end{pmatrix}$$

Les proportions sont supposées égales pour toutes les classes ($p_k = \frac{1}{s}$ et $q_\ell = \frac{1}{m}$) pour les deux premiers tableaux. Pour le troisième tableau, nous avons choisi de prendre les premiers centres, ainsi que $p_7 = 0.6$ et $q_6 = 0.6$ (avec $p_k = \frac{1}{15} \forall\, k = 1,\ldots,6$ et $q_\ell = \frac{2}{25} \forall\, \ell = 1,\ldots,5$). Nous reportons dans le tableau 6.1 le nombre d'objets mal-classés en lignes et en colonnes, pour les deux méthodes. Alors que pour le premier tableau, elles trouvent les mêmes partitions, pour le deuxième tableau, CEM croisé trouve 5 objets mal-classés en lignes et aucun en colonnes, contre respectivement 63 et 33 pour double $k$-means. Pour le troisième tableau, avec les proportions différentes, CEM croisé retrouvent exactement les partitions simulées en lignes et en colonnes. Par contre, double $k$-means ne parvient pas à faire de même, et trouve 90 objets mal-classés (mais aucune variable). Il est très clair que l'algorithme de classification croisée CEM croisé est plus nettement plus performant qu'un double $k$-means.

### 6.1.8 Application à des données binaires

Maintenant, nous allons nous intéresser aux données binaires, et décrire le modèle de mélange associé (Govaert et Nadif, 2003). Dans cette situation, la distribution de probabilité $\varphi_{k\ell}(x_i^j;\alpha)$ est la distribution de Bernoulli, qui peut être écrite de cette manière :

$$\varphi_{k\ell}(x_i^j;\alpha) = (\alpha_k^\ell)^{x_i^j}(1-\alpha_k^\ell)^{(1-x_i^j)}.$$

Ainsi

$$\log \varphi_{k\ell}(x_i^j;\alpha) = x_i^j \log\left(\frac{\alpha_k^\ell}{1-\alpha_k^\ell}\right) + \log(1-\alpha_k^\ell)$$

et la log-vraisemblance complétée peut être décrite comme suit :

$$\begin{aligned} L_c(\mathbf{z},\mathbf{w},\theta) &= \sum_{k=1}^{s} \#z_k \log(p_k) + \sum_{\ell=1}^{m} \#w_\ell \log(q_\ell) + \sum_{k=1}^{s}\sum_{\ell=1}^{m} \#z_k \#w_\ell \log(1-\alpha_k^\ell) \\ &+ \sum_{i=1}^{n}\sum_{j=1}^{d}\sum_{k=1}^{s}\sum_{\ell=1}^{m} z_{ik} w_{j\ell} x_i^j \log\left(\frac{\alpha_k^\ell}{1-\alpha_k^\ell}\right). \end{aligned}$$

Comme nous l'avons déjà précisé, la maximisation de $L_c(\mathbf{z},\mathbf{w},\theta)$ est obtenue en maximisant les log-vraisemblances conditionnelles $L_c(\mathbf{z},\theta/\mathbf{w})$ et $L_c(\mathbf{w},\theta/\mathbf{z})$, associées au modèle de mélange simple, avec

$$\begin{cases} \psi_k(\mathbf{u}_i;\alpha_k,\#\mathbf{w}) = \prod_{\ell=1}^{m} \psi_{k\ell}(u_i^\ell;\alpha_k^\ell,\#w_\ell) & \text{avec} \quad u_i^\ell = \sum_{j=1}^{d} w_{j\ell} x_i^j \\ \psi_\ell(\mathbf{v}_j;\alpha_\ell,\#\mathbf{z}) = \prod_{k=1}^{s} \psi_{k\ell}(v_k^j;\alpha_k^\ell,\#z_k) & \text{avec} \quad v_k^j = \sum_{i=1}^{n} z_{ik} x_i^j \end{cases}$$

où $\psi_{k\ell}(u_i^\ell;\alpha_k^\ell,\#w_\ell)$ est la distribution binomiale de paramètre $(\alpha_k^\ell,\#w_\ell)$ et $\binom{\#w_\ell}{u_i^\ell}$ est le coefficient binomial (identique avec $v_k^j$ et $\#z_k$).

#### L'algorithme

Nous décrivons maintenant une itération de CEM, effectué dans la phase 2(a) de notre algorithme, et qui calcule $(\mathbf{z}^{(c+1)},\mathbf{p}^{(c+1)},\alpha')$ à partir de $(\mathbf{z}^{(c)},\mathbf{p}^{(c)},\alpha)$ selon les étapes suivantes

**E-step** : Calcul des probabilités a posteriori

$$t_{ik} \propto p_k \prod_{\ell=1}^{m} \binom{\#w_\ell}{u_i^\ell} (\alpha_k^\ell)^{u_i^\ell}(1-\alpha_k^\ell)^{\#w_\ell - u_i^\ell}.$$

**C-step** : La partition $\mathbf{z}^{(c+1)}$ est définie en maximisant $t_{ik}$ qui est équivalent à maximiser

$$\sum_{\ell=1}^{m} u_i^\ell \log\left(\frac{\alpha_k^\ell}{1-\alpha_k^\ell}\right) + \sum_{\ell=1}^{m} \#w_\ell \log(1-\alpha_k^\ell) + \log(p_k).$$

**M-step** : Calcul des paramètres $(\mathbf{p}^{(c+1)},\alpha')$, ce qui revient à calculer, pour la classe $k$

$$\begin{cases} p_k^{(c+1)} = \dfrac{\#z_k^{(c+1)}}{n} \\ (\alpha_k^\ell)' = \dfrac{\sum_{i=1}^{n} z_{ik}^{(c+1)} u_i^\ell}{\#z_k^{(c+1)} \#w_\ell}. \end{cases}$$

Alternativement, une itération de l'algorithme CEM de la phase 2(b), calculant $(\mathbf{w}^{(c+1)},\mathbf{q}^{(c+1)},\alpha^{(c+1)})$ à partir de $(\mathbf{w}^{(c)},\mathbf{q}^{(c)},\alpha')$, avec les étapes suivantes :

**E-step** : Calcul des probabilités a posteriori

$$t_{j\ell} \propto q_\ell \prod_{k=1}^{g} \binom{\#z_k}{v_k^j} (\alpha_k^\ell)^{v_k^j} (1-\alpha_k^\ell)^{\#z_k - v_k^j}.$$

**C-step** : La partition $\mathbf{w}^{(c+1)}$ est définie en maximisant $t_{j\ell}$, qui est équivalent à maximiser

$$\sum_{k=1}^{g} v_k^j \log\left(\frac{\alpha_k^\ell}{1-\alpha_k^\ell}\right) + \sum_{k=1}^{g} \#z_k \log(1-\alpha_k^\ell) + \log(q_\ell).$$

**M-step** : Calcul des paramètres $(\mathbf{q}^{(c+1)},\alpha^{(c+1)})$, ce qui revient à faire, pour la classe $k$

$$\begin{cases} q_\ell^{(c+1)} = \dfrac{\#w_\ell^{(c+1)}}{d} \\ (\alpha_k^\ell)^{(c+1)} = \dfrac{\sum_{j=1}^{d} w_{j\ell}^{(c+1)} v_k^j}{\#z_k \#w_\ell^{(c+1)}}. \end{cases}$$

### Modèles restreints

Comme nous l'avons vu auparavant, l'approche modèle de mélange, dans le cas de la classification croisée, nous permet de définir le critère optimisé par l'algorithme *Croeuc* (ainsi que par d'autres méthodes), et ainsi de proposer une explication probabiliste de celui-ci. Maintenant, de la même manière que dans le cas Gaussien, nous appliquons une paramétrisation de $\alpha_k^\ell$ du modèle croisé de Bernoulli. Ainsi, la fonction de densité de probabilité de Bernoulli peut s'écrire :

$$\varphi_{k\ell}(x_i^j;(a_k^\ell,\varepsilon_k^\ell)) = (\varepsilon_k^\ell)^{|x_i^j - a_k^\ell|}(1-\varepsilon_k^\ell)^{1-|x_i^j-a_k^\ell|}$$

si nous remplaçons chaque $\alpha_k^\ell$ par $a_k^\ell \in \{0,1\}$ et $\varepsilon_k^\ell \in [0,1/2]$ avec

$$\begin{cases} a_k^\ell = 1 \text{ et } \varepsilon_k^\ell = 1-\alpha_k^\ell & \text{si} \quad \alpha_k^\ell \in [1/2,1] \\ a_k^\ell = 0 \text{ et } \varepsilon_k^\ell = \alpha_k^\ell & \text{si} \quad \alpha_k^\ell \in [0,1/2]. \end{cases}$$

Après de simples calculs, la log-vraisemblance complétée associée peut ainsi être écrite de la manière suivante :

$$\begin{aligned} L_c(\mathbf{z},\mathbf{w},\theta) &= \sum_{i=1}^{n}\sum_{j=1}^{d}\sum_{k=1}^{s}\sum_{\ell=1}^{m} z_{ik}w_{j\ell}\log\left(\frac{\varepsilon_k^\ell}{1-\varepsilon_k^\ell}\right)|x_i^j - a_k^\ell| \\ &+ \sum_{i,j}\sum_{k,\ell} z_{ik}w_{j\ell}\log(1-\varepsilon_k^\ell) \\ &+ \sum_{k=1}^{s}\#z_k\log(p_k) + \sum_{\ell=1}^{m}\#w_\ell\log(q_\ell). \end{aligned}$$

Comme la phase 2(b) de notre algorithme peut facilement être déduite, nous décrivons seulement la phase 2(a). Ainsi, les différentes étapes seront

**E-step** : Calcul des probabilités a posteriori

$$t_{ik} \propto p_k \prod_{\ell=1}^{m}\binom{\#w_\ell}{u_i^\ell}(\varepsilon_k^\ell)^{|u_i^\ell - \#w_\ell a_k^\ell|}(1-\varepsilon_k^\ell)^{\#w_\ell - |u_i^\ell - \#w_\ell a_k^\ell|}.$$

**C-step** : La partition $\mathbf{z}^{(c+1)}$ est définie en maximisant $t_{ik}$, qui est équivalent à maximiser

$$\sum_{\ell=1}^{m}\log\left(\frac{\varepsilon_k^\ell}{1-\varepsilon_k^\ell}\right)|u_i^\ell - \#w_\ell a_k^\ell| + \sum_{\ell=1}^{m}\#w_\ell\log(1-\varepsilon_k^\ell) + \log(p_k).$$

**M-step** : Calcul des paramètres $(\mathbf{p}^{(c+1)},(\mathbf{a}',\varepsilon'))$, qui revient à faire, pour la classe $k$ :

$$\begin{cases} p_k^{(c+1)} = \frac{\#z_k^{(c+1)}}{n} \\ (a_k^\ell)' = 1 \text{ si } u_k^\ell \geq \#z_k^{(c+1)}\#w_\ell;\ 0 \text{ sinon} \\ (\varepsilon_k^\ell)' = \dfrac{|u_k^\ell - \#z_k^{(c+1)}\#w_\ell(a_k^\ell)'|}{\#z_k^{(c+1)}\#w_\ell}, \end{cases}$$

où $u_k^\ell = \sum_{i=1}^{n}\sum_{k=1}^{s} z_{ik}u_i^\ell$. Avec cette paramétrisation, nous voyons que chaque bloc $\mathbf{x}_k^\ell$ est caractérisé par le mode $a_k^\ell$ et le paramètre $\varepsilon_k^\ell$, qui varie dans l'intervalle [0,1/2], et qui indique le degré d'homogénéité du bloc. On peux noter que $(1-\varepsilon_k^\ell)$ est simplement la proportion de valeurs dans le bloc $\mathbf{x}_k^\ell$ égales au mode $a_k^\ell$.

Par commodité, le modèle de mélange ainsi défini est noté $\lfloor p_k,q_\ell,\varepsilon_k^\ell \rfloor$. Comme pour le modèle de mélange simple, il est évident que nous pouvons dériver de celui-ci différents modèles, en imposant certaines contraintes sur les paramètres $p_k$, $q_\ell$ et $\varepsilon_k^\ell$. Les hypothèses différentes sur ces paramètres conduisent à 16 modèles. Par exemple, on peut supposer $\varepsilon_k^\ell$ tous égaux à $\varepsilon_k$ pour $\ell = 1,\ldots,m$, ainsi que des proportions $p_k$ et $q_\ell$ égales respectivement pour $k = 1,\ldots,s$ et $\ell = 1,\ldots,m$. Nous noterons ce modèle ainsi défini $\lfloor p,q,\varepsilon_k \rfloor$. Avec cette notation, le modèle $\lfloor p,q,\varepsilon \rfloor$ est le modèle le plus simple, tous les $\varepsilon_k^\ell$ sont supposés égaux à $\varepsilon$, inconnu.

### Lien entre CEM croisé et *Crobin*

Avec le modèle le plus simple, l'expression de $L_c(\mathbf{z},\mathbf{w},\theta)$ est réduite à

$$L_c(\mathbf{z},\mathbf{w},\theta) = \log\left(\frac{\varepsilon}{1-\varepsilon}\right)\sum_{i=1}^{n}\sum_{j=1}^{d}\sum_{k=1}^{k}\sum_{\ell=1}^{m} z_{ik}w_{j\ell}|x_i^j - a_k^\ell| + D,$$

où $D = \{nd\log(1-\varepsilon) - n\log s - d\log m\}$. En utilisant l'expression du critère de l'algorithme *Crobin*, la log-vraisemblance complétée devient

$$L_c(\mathbf{z},\mathbf{w},\theta) = \log\left(\frac{\varepsilon}{1-\varepsilon}\right) W(\mathbf{z},\mathbf{w},\mathbf{a}) + D$$

De cette manière, maximiser $L_c(\mathbf{z},\mathbf{w},\theta)$ est équivalent à maximiser $\log(\frac{\varepsilon}{1-\varepsilon})W(\mathbf{z},\mathbf{w},\mathbf{a})$, et donc à minimiser $W(\mathbf{z},\mathbf{w},\mathbf{a})$, car on a $\log(\frac{\varepsilon}{1-\varepsilon}) \leq 0$. Ainsi, il est facile de montrer, que dans cette situation, l'algorithme CEM croisé est exactement l'algorithme *Crobin* décrit dans le chapitre précédent. Les maximisations de $L_c(\mathbf{z},\theta/\mathbf{w})$ et $L_c(\mathbf{w},\theta/\mathbf{z})$ sont donc équivalentes aux minimisations respectives de $W(\mathbf{z},\mathbf{a}/\mathbf{w})$ et$W(\mathbf{w},\mathbf{a}/\mathbf{z})$.

On note qu'à la convergence, le paramètre $\varepsilon$ peut être estimé à partir de la paire optimale $(\mathbf{z}^*,\mathbf{w}^*)$ par

$$\varepsilon = \frac{\sum_{k=1}^{s}\sum_{\ell=1}^{m}\sum_{i\in z_k^*}\sum_{j\in w_\ell^*}|x_i^j - a_k^\ell|}{nd}$$

et $(1-\varepsilon)$ représente le degré global d'homogénéité. La valeur 1 correspond alors à une partition en blocs parfaite.

### Application sur des données simulées

Pour illustrer le bon comportement de cet algorithme, nous l'avons appliqué sur des données binaires simulées. Pour ceci, nous avons choisi de prendre $n = 200$, $d = 100$, $s = 4$, $m = 3$ et les noyaux et proportions suivants :

| | 0.2 | 0.2 | 0. 6 |
|---|---|---|---|
| 0.2 | 1 | 1 | 1 |
| 0.2 | 1 | 1 | 0 |
| 0.2 | 1 | 0 | 0 |
| 0.4 | 0 | 0 | 0 |

Nous avons volontairement choisi des proportions différentes, afin de montrer l'intérêt de CEM croisé, par rapport à CROBIN. Dans le tableau 6.2, nous reportons le croisement des deux partitions obtenues en lignes et en colonnes avec celles simulées, elles sont identiques. Dans la figure 6.1, nous présentons la matrice simulée et la matrice réordonnée avec les classes de CEM croisé. Pour ce tableau, CROBIN ne parvient pas à fournir de partitions sans classes vides. Il est donc incapable de traiter des cas avec des classes de tailles dramaticalement différentes.

TAB. 6.2 – *Croisement des partitions en lignes (à gauche) et en colonnes (à droite) simulées* $\mathbf{z}$ *et* $\mathbf{w}$*, et celles obtenues avec CEM croisé, sur les données binaires simulées.*

| | $\mathbf{z}$ | | | |
|---|---|---|---|---|
| CEM croisé | 37 | 0 | 0 | 0 |
| | 0 | 0 | 0 | 93 |
| | 0 | 27 | 0 | 0 |
| | 0 | 0 | 43 | 0 |

| | $\mathbf{w}$ | | |
|---|---|---|---|
| CEM croisé | 31 | 0 | 0 |
| | 0 | 0 | 52 |
| | 0 | 17 | 0 |

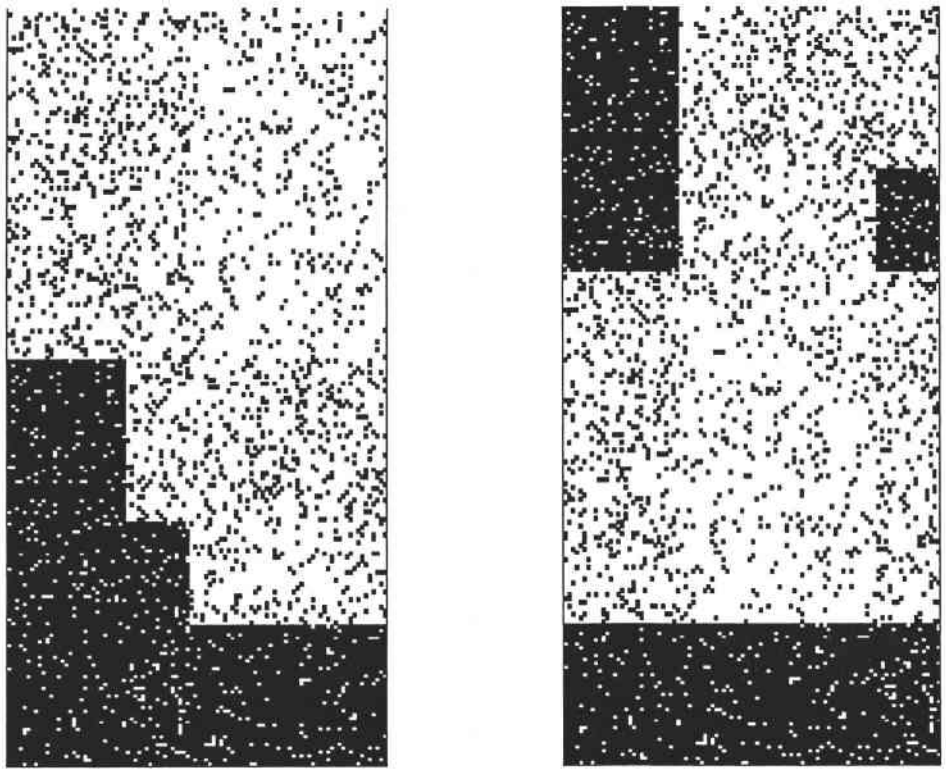


FIG. 6.1 – *Matrice binaire initiale (à gauche) et réordonnée (à droite) selon les partitions en lignes et en colonnes fournies par CEM croisé.*

## 6.2 Méthode hiérarchique de classification croisée (*HBCM*)

### 6.2.1 Description

La classification hiérarchique classique est une méthode agglomérative de classification, démarrant avec chaque objet dans sa propre classe. Ensuite, par agglomérations successives des paires de classes les plus proches, elle combine celles-ci pour n'avoir plus qu'une seule classe à la convergence de l'algorithme. On peut représenter cette séquence par un arbre hiérarchique, qui peut être couper à n'importe quel niveau, pour fournir un nombre de classes spécifié.

Quand nous avons étudié l'utilisation de la classification croisée, pour partitionner simultanément les lignes et les colonnes, une approche simple est d'appliquer la méthode de classification hiérarchique classique sur les objets, puis sur les variables, séparément. Malheureusement, dans un contexte de classification croisée, nous cherchons à utiliser ensemble les informations des objets et des variables afin de mieux les classifier.

Nous proposons une version croisée de l'algorithme de classification hiérarchique classique (Nadif *et al.*, 2002; Jollois *et al.*, 2003), que nous appelons *HBCM* (pour *Hierarchical Block Clustering Method*). Démarrant avec chaque objet et chaque variable dans leur propre classe, nous cherchons à chaque étape à combiner les deux classes les plus proches (en ligne ou en colonnes). Les deux classes d'objet les plus proches $z_k$ et $z_{k'}$ sont regroupées conditionnellement à la dernière partition en colonne $\mathbf{w}$, pour former une nouvelle classe $z_{kk'}$, rassemblant les deux anciennes. Il en est de même en colonne, avec les classes de variables $w_\ell$ et $w_{\ell'}$ regroupées dans une nouvelle classe $w_{\ell\ell'}$, selon la partition en ligne $\mathbf{z}$. Le but est de combiner toutes les classes en ligne et en colonne entre elles, afin de n'avoir plus qu'une seule classe d'objets et une seule classe de variables.

Les distances ou mesures de dissimilarité entre les classes d'objets $\Delta(z_k,z_{k'})$ ou de variables $\Delta(w_\ell,w_{\ell'})$ sont définies de la même manière. Nous pouvons décrire $\Delta(z_k,z_{k'})$ comme suit :

$$\Delta(z_k,z_{k'}) = L_c(\mathbf{z},\theta/\mathbf{w}) - L_c(\mathbf{z}',\theta/\mathbf{w}),$$

où $\mathbf{z}'$ est la partition obtenu quand $z_k$ et $z_{k'}$ sont regroupés. Comme $L_c(\mathbf{z}',\theta/\mathbf{w}) \leq L_c(\mathbf{z},\theta/\mathbf{w})$, il est automatique que $\Delta(z_k,z_{k'})$ est défini de tel sorte que la vraisemblance classifiante est minimisée à chaque étape.

A partir de la vraisemblance classifiante décomposée selon les classes de cette manière :

$$\begin{aligned} L_c(\mathbf{z},\theta/\mathbf{w}) &= \sum_{k=1}^{s} L_c(z_k,\theta_k/\mathbf{w}) \\ \text{et } L_c(\mathbf{w},\theta/\mathbf{z}) &= \sum_{\ell=1}^{m} L_c(w_\ell,\theta^\ell/\mathbf{z}), \end{aligned}$$

cette mesure de dissimilarité devient :

$$\Delta(z_k,z_{k'}) = L_c(z_k,\theta_k/\mathbf{w}) + L_c(z_{k'},\theta_{k'}/\mathbf{w}) - L_c(z_{kk'},\theta_{k'}/\mathbf{w}). \tag{6.5}$$

Puisque $\Delta(z_k,z_{k'})$ et $\Delta(w_\ell,w_{\ell'})$ dépendent seulement des objets et des variables appartenant respectivement aux classes concernées par la fusion, toutes les autres distances entre les autres classes restent inchangées. De plus, les paramètres $\theta_k$ et $\theta^\ell$, ainsi que les valeurs de $\Delta(z_k,z_{k'})$ et $\Delta(w_\ell,w_{\ell'})$ dépendent uniquement des statistiques exhaustives, qui sont stockées pour chaque classe.

### 6.2.2 Application aux données continues

Après quelques calculs, et à partir du modèle de mélange Gaussien croisé précédemment décrit dans ce chapitre, nous obtenons un modèle de mélange croisé adapté aux données continues (Nadif *et al.*, 2002), décrit par

$$\Delta(z_k,z_{k'}) = \frac{n}{2{\sigma_0}^2}\delta(z_k,z_{k'})$$

où

$$\delta(z_k,z_{k'}) = \frac{\#z_k \times \#z_{k'}}{\#z_k + \#z_{k'}}\mathbf{d}_M^2(\overline{z}_k,\overline{z}_{k'})$$

avec $M = diag(\#w_1,\ldots,\#w_m)$, et $\overline{z}_k$ et $\overline{z}_{k'}$ représentent les moyennes des objets dans les classes $z_k$ et $z_{k'}$. Notons que $\Delta(z_k,z_{k'})$ est proportionnel à la distance entre classes selon le critère de Ward. Comme pour les méthodes classiques, il y a une simple relation de récurrence, pour mettre à jour le coût de regroupement des classes. Dans la méthode classique, associée au critère de Ward, cette récurrence peut être décrite de cette manière

$$\begin{aligned}\Delta(z_{kk'},z_{k''}) &= \frac{n}{2{\sigma_0}^2}\frac{(n_k+n_{k''})\delta(z_k,z_{k''}) + (n_{k'}+n_{k''})\delta(z_{k'},z_{k''}) - n_{k''}\delta(z_k,z_{k'})}{n_k+n_{k'}+n_{k''}} \\ &+ (n_k+n_{k'})\log\frac{n_k+n_{k'}}{n_k+n_{k'}+n_{k''}} + n_{k''}\log\frac{n_{k''}}{n_k+n_{k'}+n_{k''}}\end{aligned}$$

où $n_k = \#z_k$. Une fois que le coût initial de regroupement de chaque paire est obtenu, l'algorithme peut se dérouler sans avoir à refaire une référence aux données. Les tailles de chaque classe doivent seulement être retenues et mises à jour. L'espace total nécessaire pour $\Delta(z_k,z_{k'})$ décroît quand le nombre de classes diminue.

### 6.2.3 Illustration de la méthode *HBCM*

Maintenant, pour illustrer cette méthode, nous utilisons un exemple simple. Dans le tableau 6.3, nous présentons les pourcentages de votes républicains pour les présidentielles dans les états du Sud des États-Unis (déjà étudié dans le chapitre précédent, voir (Hartigan, 1975)). Il est naturel de chercher des classes d'états et des classes d'années, afin d'analyser le comportement des états en fonction des années, et les années en fonction des états.

Dans le tableau 6.4, nous décrivons l'historique du processus de classification. Ensuite, grâce aux dendrogrammes de la figure 6.2, nous suggérons de prendre 2 classes en lignes et 3 classes en colonnes. Nous pouvons voir le découpage résultant dans cette figure. Nous voyons que les états MD, MO, KY sont très proches. De même, les années 1932, 1936 et 1940 le sont aussi. Et la différence entre cette classe

d'années et celle composée de 1960 et 1968 est principalement due aux états LA, SC et MS. L'année 1964 est très particulière et ainsi seule dans sa classe. Il y a une inversion entre les états MD, MO et KY votant légèrement moins républicains cette année là que les états LA, SC et MS, contrairement à toutes les autres années. On peut noter que *Croeuc* lancé avec ces nombre de classes $s = 2$ et $m = 3$ donne les mêmes résultats.

TAB. 6.3 – *Votes républicains pour les états du sud des États-Unis.*

| | 32 | 36 | 40 | 60 | 64 | 68 |
|---|---|---|---|---|---|---|
| MS | 4 | 3 | 4 | 25 | 87 | 14 |
| SC | 2 | 1 | 4 | 49 | 59 | 39 |
| LA | 7 | 11 | 14 | 29 | 57 | 23 |
| KY | 40 | 40 | 42 | 54 | 36 | 44 |
| MD | 36 | 37 | 41 | 46 | 35 | 42 |
| MO | 35 | 38 | 48 | 50 | 36 | 45 |

TAB. 6.4 – *Historique de l'algorithme HBCM (nous reportons ici le critère $W$ plutôt que le critère $L_c$, qui sont proportionnels).*

| itération | classes formées | $W$ |
|---|---|---|
| 1 | 32 36 | 14.00 |
| 2 | KY MO | 52.75 |
| 3 | KY MO MD | 96.33 |
| 4 | 32 36 40 | 189.33 |
| 5 | 60 68 | 378.00 |
| 6 | SC LA | 808.17 |
| 7 | SC LA MS | 1705.22 |
| 8 | 32 36 40 60 68 | 4012.00 |
| 9 | SC LA MS KY MO MD | 11124.03 |
| 10 | 32 36 40 60 68 64 | 13715.64 |

### 6.2.4 Application aux données binaires

À partir du modèle de mélange binaire croisé précédemment décrit dans ce chapitre, nous obtenons un modèle de mélange croisé pour données binaires (Jollois *et al.*, 2003), et nous pouvons décrire la log-vraisemblance classifiante associé à une classe avec :

$$L_c(z_k,\mathbf{w},\theta) = \sum_{i\in z_k}\sum_{\ell=1}^{m}\sum_{j\in w_\ell} x_i^j \log(\alpha_k^\ell) + (1-x_i^j)\log(1-\alpha_k^\ell)$$

Ainsi, avec ce calcul et celui de la distance entre deux classes (6.5), on défini les paramètres de la classe $z_{kk'}$ par

$$\alpha_{kk'}^\ell = \frac{\#z_k\alpha_k^\ell + \#z_{k'}\alpha_{k'}^\ell}{\#z_k + \#z_{k'}}$$

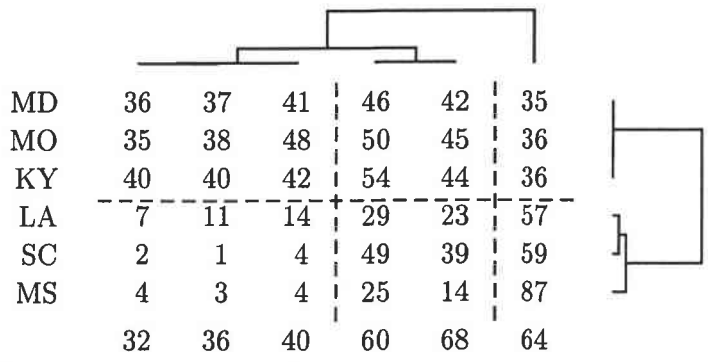



FIG. 6.2 – *Résultats de HBCM sur les votes républicains (matrice réordonnée et arbres hiérarchiques associés).*

Malheureusement, nous n'avons pas trouvé de formule de récurrence pour ce modèle, au contraire du modèle Gaussien croisé.

### 6.2.5 Comparaison entre *HBCM* et CAH classique, sur des données binaires réelles

Afin d'illustrer l'algorithme *HBCM* pour données binaires, et de plus comprendre son intérêt par rapport à une CAH simple, nous avons appliquer ces deux algorithmes sur des données binaires réelles. Ce tableau décrit le diagnostic des images cardiaques par tomographie d'émission monophotonique (ou SPECT). Chacun des 267 patients est classé dans deux catégories : normal et anormal. A partir de 44 mesures continues (par paires au repos et activé), 22 variables binaires ont été obtenues. Ici, nous utilisons donc le tableau binaire.

TAB. 6.5 – *Croisement de la partition initiale et des partitions obtenues après découpages des arbres hiérarchiques en ligne pour CAH et HBCM, pour les données binaires SPECT.*

| | CAH | |
|---|---|---|
| Partition initiale | 18 | 37 |
| | 161 | 51 |

| | *HBCM* | |
|---|---|---|
| Partition initiale | 29 | 26 |
| | 191 | 21 |

Dans la figure 6.3, nous présentons les deux arbres hiérarchiques obtenus pour les algorithmes CAH et *HBCM*, sur les patients. Ensuite, dans le tableau 6.5, nous voyons le croisement entre la partition initiale et les partitions obtenues avec les algorithmes CAH et *HBCM*. L'intérêt de l'approche croisée est clairement visible ici, dans le sens où nous obtenons un taux de mauvais classement de 18.7 % avec *HBCM*, contre 25.8 % pour CAH.

### 6.2.6 Boucles mérovingiennes

Comme autre exemple de comparaison entre une double CAH et notre méthode HBCM, nous avons choisi un tableau concernant des plaques-boucles de ceintures, du

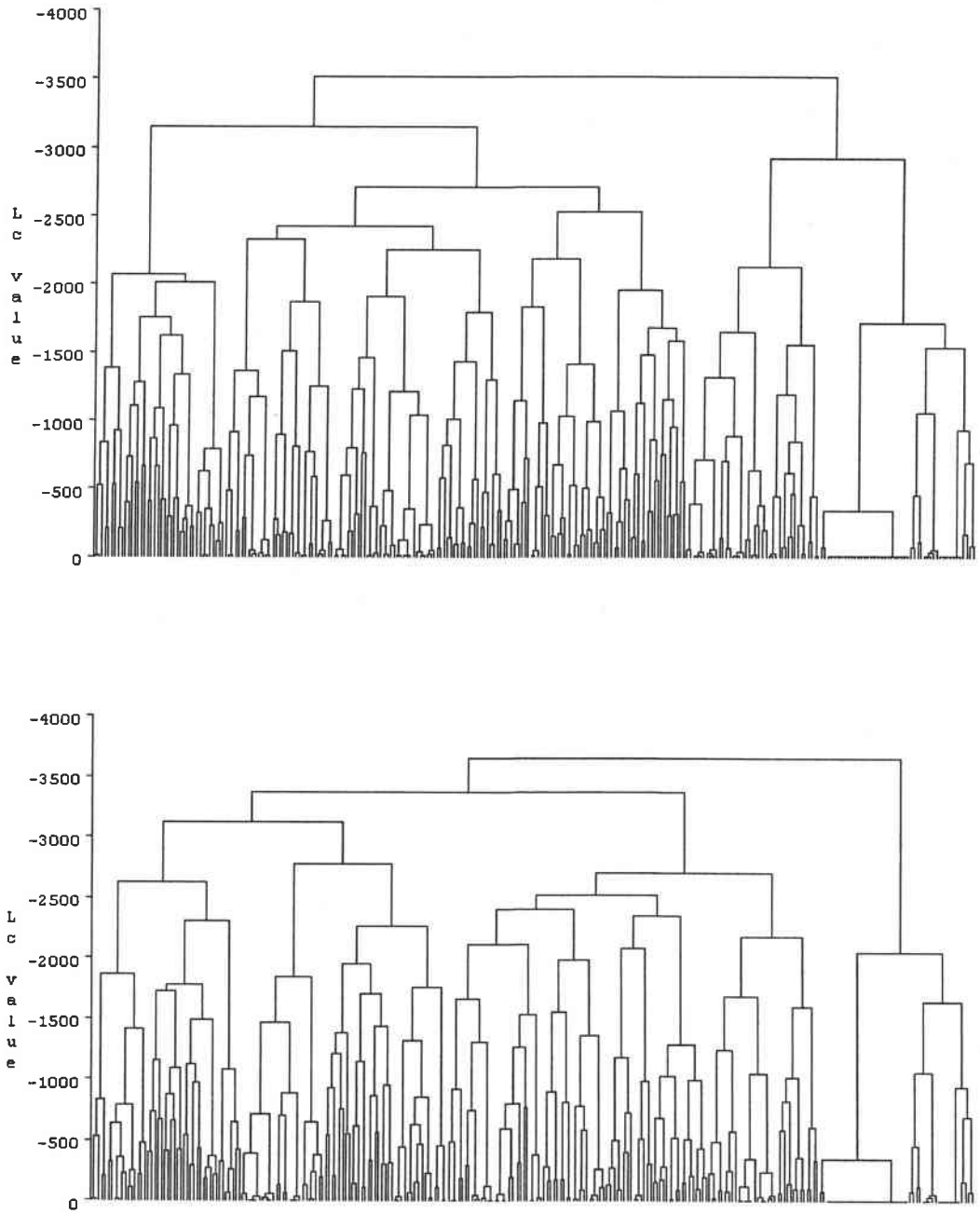



FIG. 6.3 – *Arbres hiérarchiques des lignes de CAH simple (en haut) et de HBCM (en bas) sur les données binaires SPECT.*

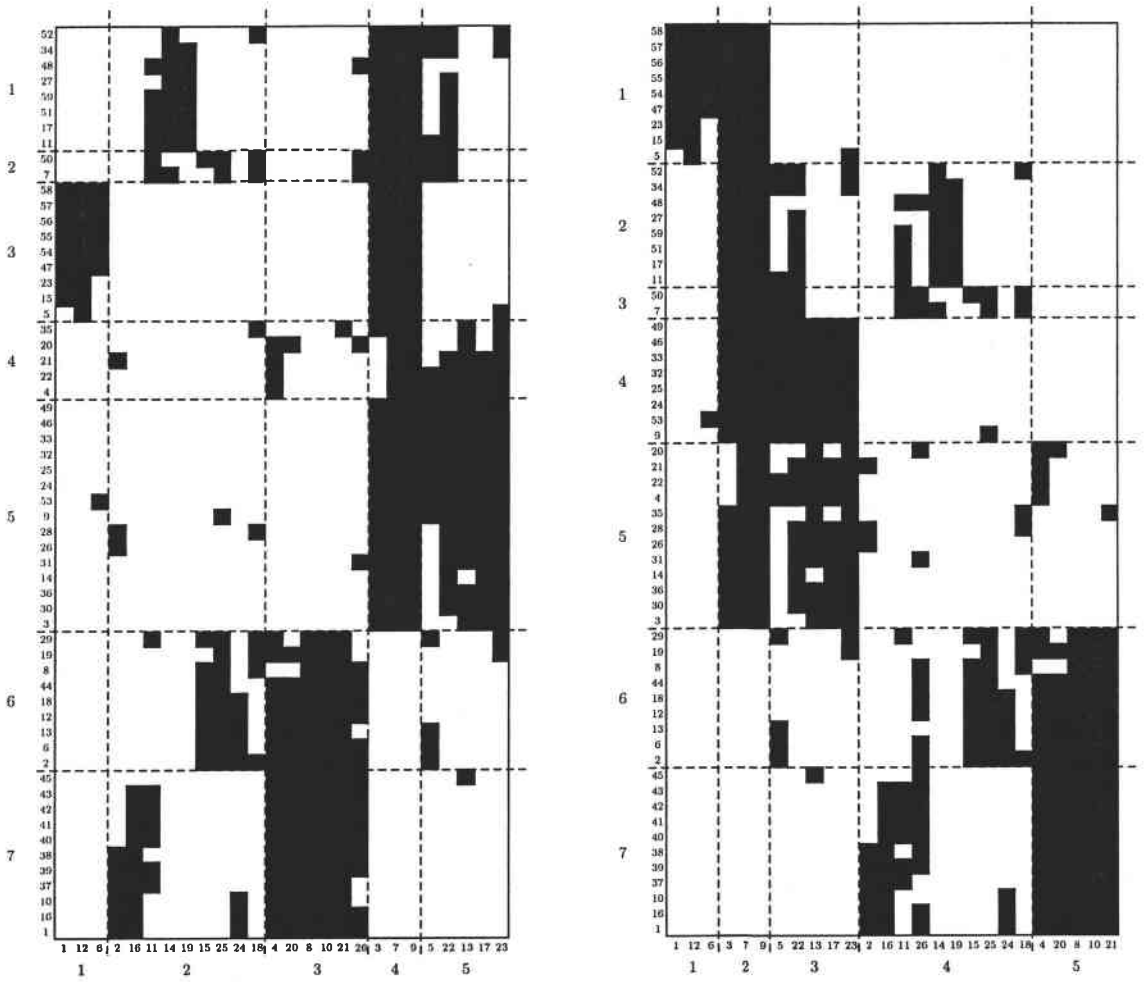


FIG. 6.4 – *Boucles mérovingiennes : données réordonnées selon les résultats obtenus par une double CAH (à gauche) et par HBCM (à droite).*

Nord-Est de la France, datant du VIème au VIIIème siècle. L'ensemble est constitué de 59 boucles sur lesquelles ont été observées la présence ou l'absence (1 ou 0) d'une sélection de 26 critères techniques de fabrication, de forme et de décor.

Les données réordonnées selon les résultats d'une CAH en lignes et en colonnes, et des résultats de *HBCM* sont présentées dans la figure 6.4. Et les arbres hiérarchiques en lignes et en colonnes sont présentés dans la figure 6.5 pour une double CAH et dans la figure 6.6 pour *HBCM*. La répartition des boucles en sept classes et des spécificités en cinq classes nous est apparue intéressante, et appropriée pour les deux méthodes.

On voit que certains groupes de boucles sont très liés à certains groupes de spécifités. Les classes 1, 2, 3, 6 et 7 obtenues avec double CAH sont les mêmes respectivement que les classes 2, 3, 1, 6 et 7 obtenues avec *HBCM*. La différence entre les deux méthodes repose sur les classes 4 et 5 de chacune, ainsi que sur le regroupement de toutes les classes. Premièrement, on remarque que *HBCM* regroupe dans la classe 4 les boucles ayant la valeur 1 dans les classes 2 et 3 en colonnes. Ces deux blocs sont d'ailleurs purs, car toutes les valeurs sont à 1. Ainsi, la structure

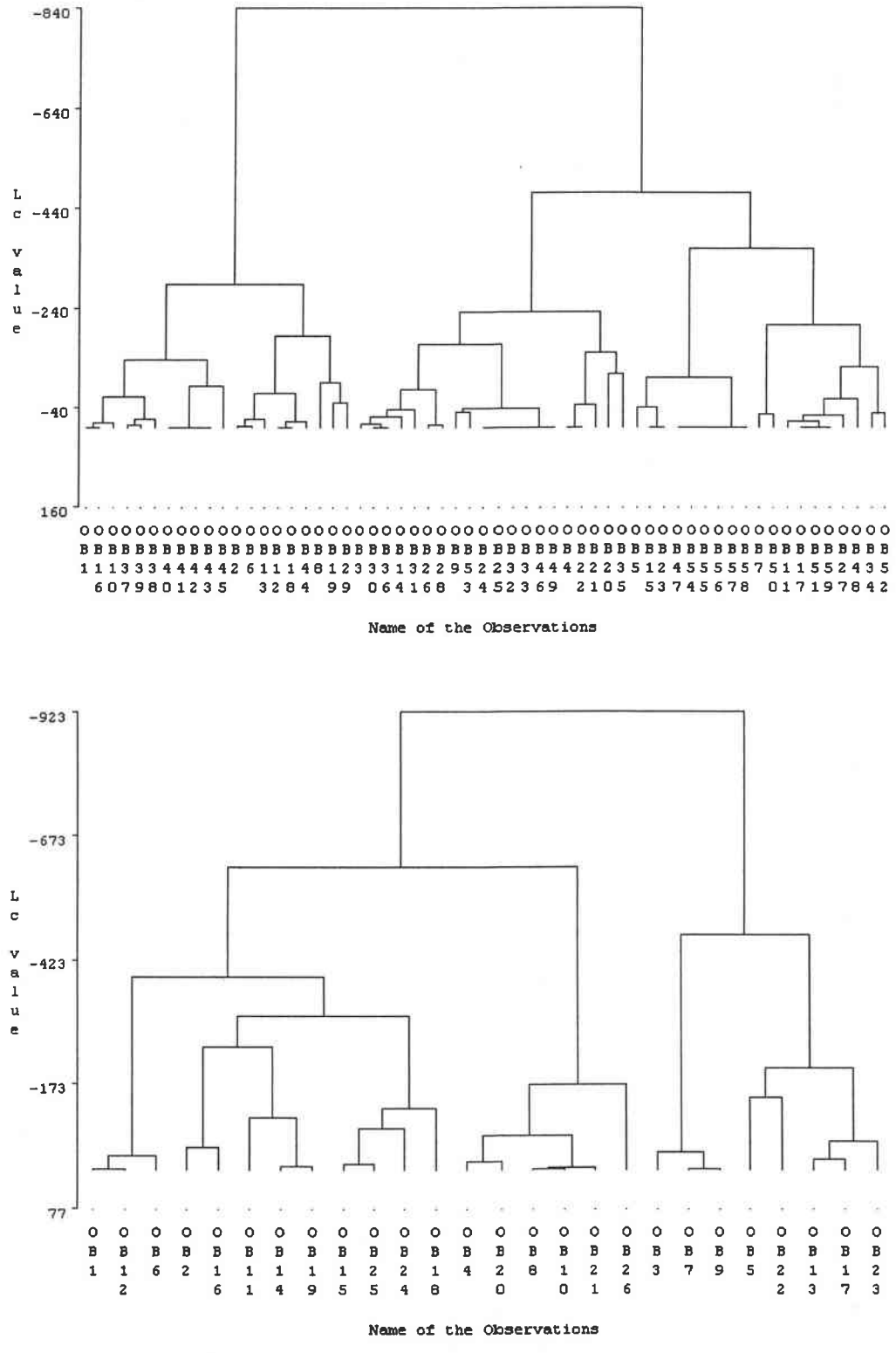
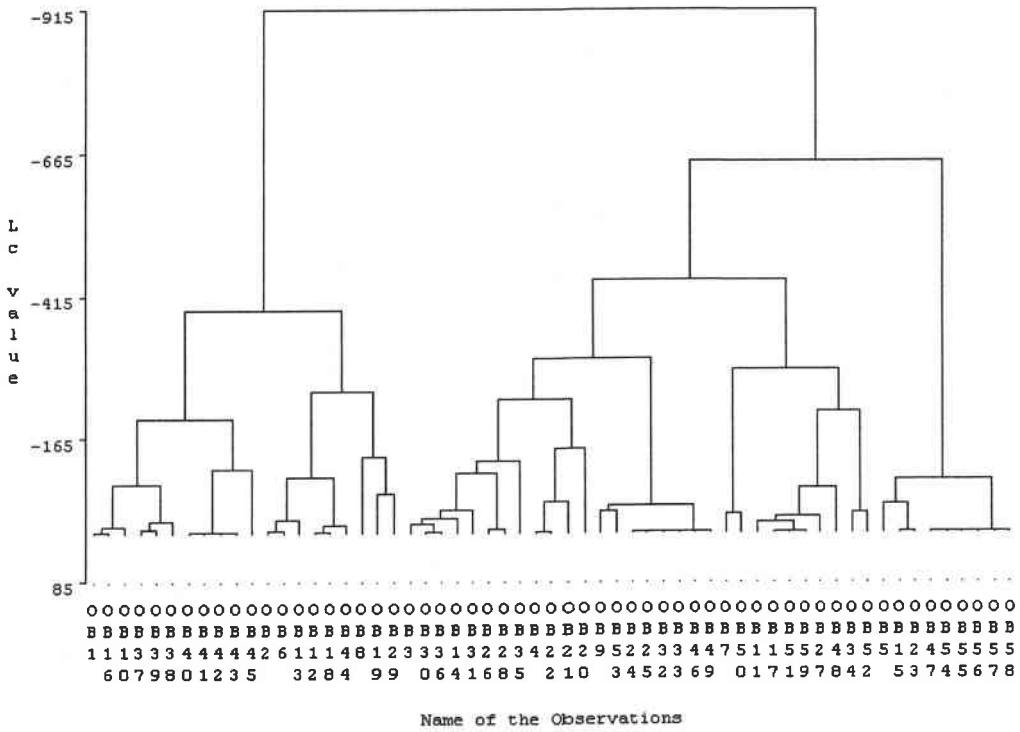



FIG. 6.5 – *Boucles mérovingiennes : arbres hiérarchiques en lignes (en haut) et en colonnes (en bas), obtenus grâce à une double CAH.*



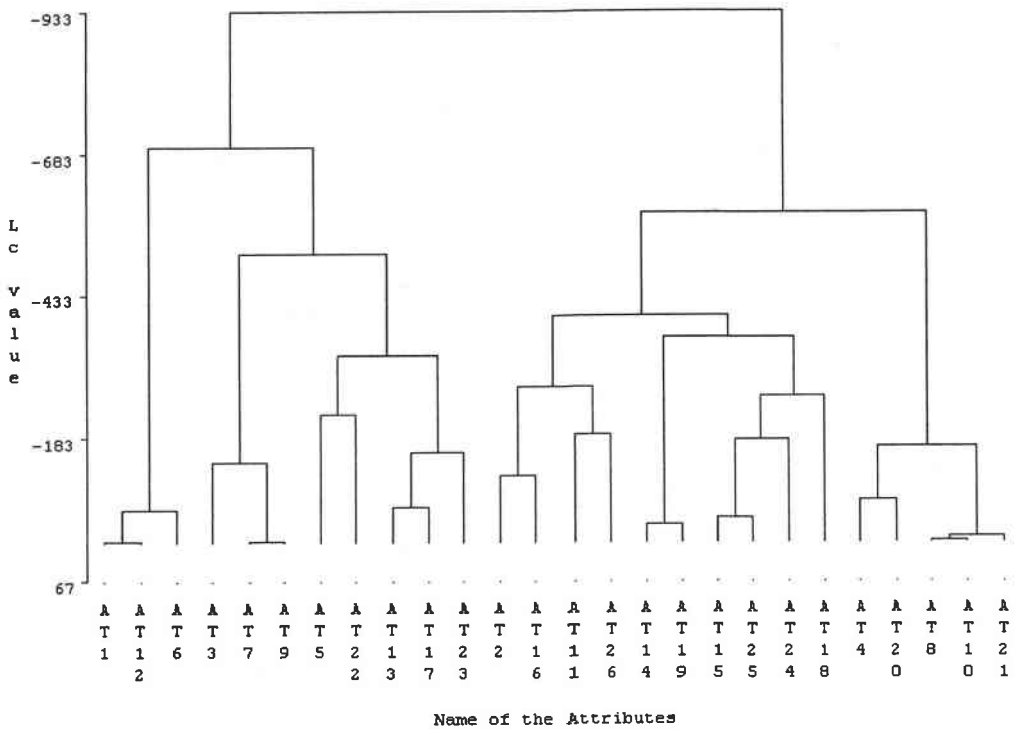




FIG. 6.6 – *Boucles mérovingiennes : arbres hiérarchiques en lignes (en haut) et en colonnes (en bas), obtenus grâce à HBCM.*

fournie par *HBCM* semble être plus intéressante, car moins éparse que celle fournie par une double CAH.

Les spécificités sont elles aussi regroupées différemment entre CAH et *HBCM*. Les classes 1, 4 et 5 de la première méthode correspondent respectivement aux classes 1, 2 et 3. Alors que la double CAH regroupe ces classes à la dernière itération, *HBCM* à l'avant-dernière étape, en les considérant ainsi comme proches. Cette solution nous semble plus cohérente.

Il est donc très clair que cette approche croisée permet de mettre en évidence une évolution des techniques utilisées pour la fabrication des boucles. Les regroupements effectués semblent plus en accord, dans l'optique d'une classification croisée.

## 6.3 Travail sur des grandes bases de données

### 6.3.1 Combinaison de CEM croisé avec *Two-way splitting*

Puisque l'algorithme *Two-way splitting* travaille sur des attributs comparables, il est possible de le combiner avec un algorithme de type CEM croisé. En effet, ce dernier propose comme résultat une matrice d'information réduite, contenant les moyennes de chaque bloc. Nous obtenons donc un tableau sur lequel *Two-way splitting* peut parfaitement s'appliquer. Ainsi, nous allons utiliser le schéma de la figure 6.7 pour les combiner.

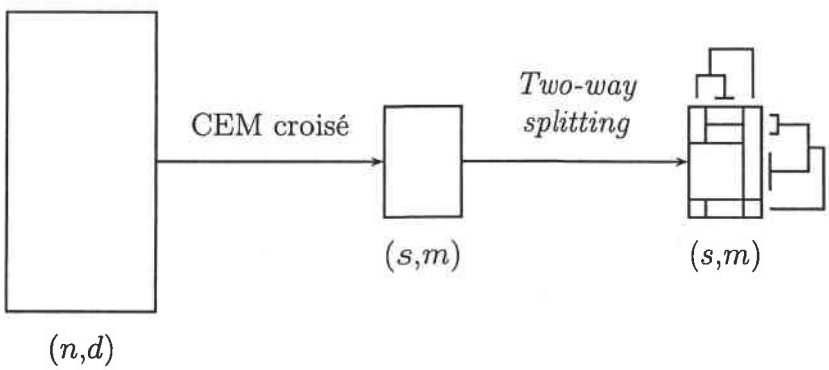



FIG. 6.7 – *Schéma illustratif décrivant l'utilisation des algorithmes CEM croisé et Two-way splitting combinés ensemble.*

De plus, grâce à cette combinaison, nous pouvons nous affranchir du problème du nombre de classes pour CEM croisé. En effet, lors d'une exécution, il sera alors judicieux de choisir $s$ et $m$ assez grands. Et ensuite, nous pourrons appliquer *Two-way splitting* sur les centres obtenus et avoir ainsi une structure claire de la matrice d'information (et donc des données de base).

Pour illustrer la méthodologie présentée, nous avons décidé de l'appliquer sur des données simulées. Nous avons choisi $n = 5000$, $d = 500$, des proportions égales (en lignes et en colonnes), chaque valeur $x_i^j$ est obtenue avec une loi normale $N(\mu,1)$, où $\mu$ est le centre du bloc auquel $x_i^j$ appartient. Nous avons pris comme structure de base la matrice présentée dans la figure 6.8 (à gauche).

Pour résumer la matrice de données, nous utilisons donc CEM croisé. Nous avons choisi de prendre $s = 10$ et $m = 5$, afin d'avoir une sous-matrice réduite très simple à utiliser. Il est bien sûr possible de choisir d'autres nombres de classes, en fonction de la granularité désirée.

Lorsque nous avons lancé *Two-way splitting*, nous avons vu que pour un seuil de 1, il fournit un découpage en bloc approprié, et identique à ceux fournis par les seuils inférieurs (0.1, 0.2, 0.3, ..., 0.9). Nous avons donc décidé de garder ce seuil de 1. Dans la figure 6.8, on voit très bien que la structure proposé par *Two-way splitting* est la même que celle simulée. Ces résultats sur des données simulées sont très encourageant. De plus, celle-ci est applicable dans le cas où les variables sont comparables (comme les données de type biopuces, données textuelles, tableaux de pourcentage, ...). Nous envisageons donc de tester cette méthode sur des données réelles, et de plus comparer celle-ci avec notre méthode hybride hiérarchique *HBCM* (Jollois *et al.*, 2003; Nadif *et al.*, 2002).

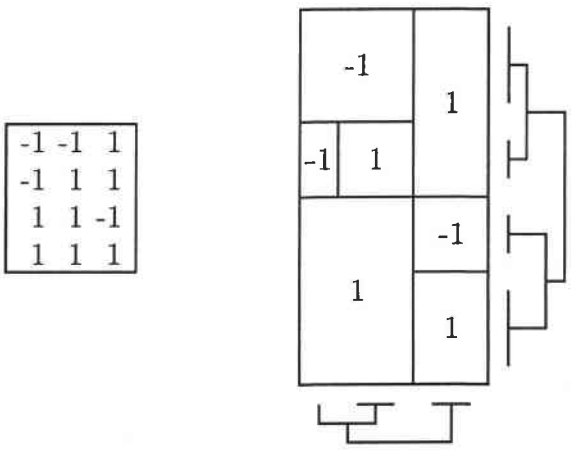



FIG. 6.8 – *Structure simulée (à gauche) et structure obtenue à l'aide de Two-way splitting appliquée sur les noyaux donnés par CEM croisé (à droite).*

### 6.3.2 Combinaison de CEM croisé avec *HBCM*

#### Exemple de données continues

Notre méthode de classification hiérarchique croisée, bien que plus rapide d'une double CAH sur les lignes et les colonnes, n'est pas réalisable sur de grands tableaux de données. Mais, de la même manière que précédemment, il est possible de combiner l'algorithme CEM croisé avec la méthode *HBCM*. Ainsi, avec le premier, nous synthétisons l'information en une matrice réduite, et avec le second, nous recherchons plus précisément le nombre de classes pour chaque ensemble (ligne ou colonne). Il est évident qu'il faut choisir un nombre de classes suffisamment grand dans la première partie. Ensuite, une fois que l'on a choisi le nombre de classes à prendre, on relance CEM croisé avec une partition initiale déduite des résultats précédents.

Comme nous l'avons vu auparavant, un des problèmes majeurs de la classification est de déterminer le nombre de classes à prendre. Ici, nous proposons de choisir celui-

ci en étudiant l'évolution du critère $W$ à chaque étape (proportionnel à $L_c$ dans la méthode *HBCM*), et en identifiant le ou les points où un coude apparaît.

Pour illustrer cette méthodologie, nous avons simulé deux tableaux de données (Set 1 : classes très séparées, Set 2 : classes moyennement séparées), avec $n = 1000$ et $d = 500$, à partir d'un modèle croisé gaussien. Les centres des classes sont respectivement $\mathbf{g}_1$ et $\mathbf{g}_2$ (décrits ci-après), la variance ${\sigma_0}^2$ est égale à 1 et les proportions sont supposées égales en lignes et en colonnes.

$$\mathbf{g}_1 = \begin{pmatrix} 5 & 5 & 5 & 5 & 5 & 5 \\ 5 & 5 & 5 & 5 & 5 & 3.5 \\ 5 & 5 & 5 & 5 & 3.5 & 3.5 \\ 5 & 5 & 5 & 3.5 & 3.5 & 3.5 \\ 5 & 5 & 3.5 & 3.5 & 3.5 & 3.5 \\ 5 & 3.5 & 3.5 & 3.5 & 3.5 & 3.5 \\ 3.5 & 3.5 & 3.5 & 3.5 & 3.5 & 3.5 \end{pmatrix} \quad \text{et} \quad \mathbf{g}_2 = \begin{pmatrix} 5 & 5 & 5 & 5 & 5 & 5 \\ 5 & 5 & 5 & 5 & 5 & 4 \\ 5 & 5 & 5 & 5 & 4 & 4 \\ 5 & 5 & 5 & 4 & 4 & 4 \\ 5 & 5 & 4 & 4 & 4 & 4 \\ 5 & 4 & 4 & 4 & 4 & 4 \\ 4 & 4 & 4 & 4 & 4 & 4 \end{pmatrix}$$

A partir de ces paramètres, nous obtenons une paire de partitions $\mathbf{z}^{(s)}$,$\mathbf{w}^{(s)}$. Pour Set 1, nous avons $\#\mathbf{z}^{(s)} = (138, 143, 150, 138, 136, 156, 139)$ et $\#\mathbf{w}^{(s)} = (88, 71, 85, 64, 79, 113)$ . Et pour Set 2, nous avons $\#\mathbf{z}^{(s)} = (156, 144, 131, 131, 145, 153, 140)$ et $\#\mathbf{w}^{(s)} = (80, 91, 90, 84, 67, 88)$. Sur les deux tableaux simulés, nous avons appliqué CEM croisé avec $s^* = 20$ et $m^* = 10$ (ces valeurs correspondant respectivement à 2 % du nombre d'individus et de variables).

Dans la figure 6.9, nous pouvons voir un coude à l'étape 17. Ainsi, nous coupons les deux arbres à la valeur du critère correspondant à cette étape. De la figure 6.10, il apparaît clairement que $s = 7$ et $m = 6$ sont les nombres appropriés de classes. Nous avons appliqué une seconde fois CEM croisé à partir d'une partition déduite par *HBCM*, et nous obtenons la paire optimale de partition $\mathbf{z}^{(h)}$,$\mathbf{w}^{(h)}$. Pour évaluer la performance de notre méthode, nous utilisons les tables croisées qui montrent les fréquences combinées des deux partitions $\mathbf{z}^{(h)}$,$\mathbf{z}^{(s)}$ et des partitions $\mathbf{w}^{(h)}$,$\mathbf{w}^{(s)}$ (voir le tableau 6.6). Nous voyons que notre algorithme trouve exactement les partitions simulées. Notons que nous obtenons les mêmes résultats (non reportés ici) avec un double $k$-means, avec $s = 7$ et $m = 6$.

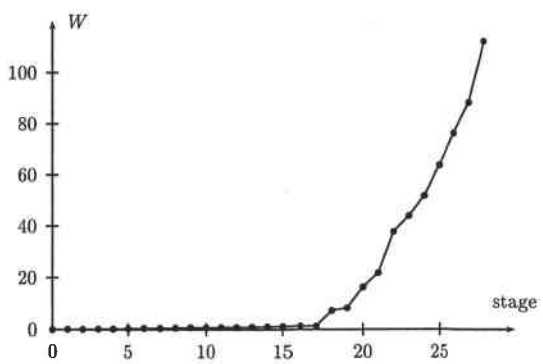



FIG. 6.9 – *Évolution du critère avec HBCM pour Set 1 ($s^* = 20$ and $m^* = 10$).*

De la même manière, nous reportons les résultats pour Set 2 (où les classes des objets et des variables sont moyennement mélangées) dans les figures 6.11 et 6.12. Nous trouvons exactement les mêmes nombres de classes $s = 7$ et $m = 6$, ainsi que de très bons résultats (voir tableaux 6.7 et 6.8). De plus, en comparant les partitions

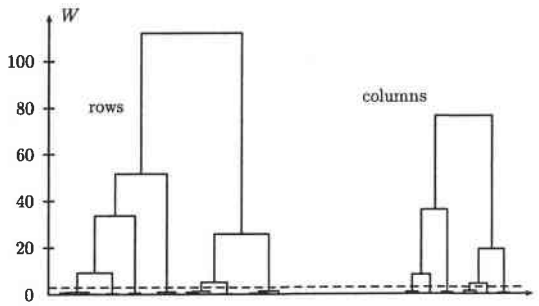



FIG. 6.10 – *Arbres hiérarchiques pour les lignes et les colonnes, à partir des centres de CEM croisé (Set 1).*

TAB. 6.6 – *Croisement des partitions en lignes et en colonnes simulées* ($\mathbf{z}^{(s)}$ *et* $\mathbf{w}^{(s)}$) *et obtenues avec CEM croisé* ($\mathbf{z}^{(h)}$ *et* $\mathbf{w}^{(h)}$), *pour Set 1.*

| $\mathbf{z}^{(s)}$ \ $\mathbf{z}^{(h)}$ | | | | | | | |
|---|---|---|---|---|---|---|---|
| | 138 | 0 | 0 | 0 | 0 | 0 | 0 |
| | 0 | 0 | 0 | 0 | 143 | 0 | 0 |
| | 0 | 0 | 0 | 0 | 0 | 150 | 0 |
| | 0 | 138 | 0 | 0 | 0 | 0 | 0 |
| | 0 | 0 | 0 | 136 | 0 | 0 | 0 |
| | 0 | 0 | 156 | 0 | 0 | 0 | 0 |
| | 0 | 0 | 0 | 0 | 0 | 0 | 139 |

| $\mathbf{w}^{(s)}$ \ $\mathbf{w}^{(h)}$ | | | | | | |
|---|---|---|---|---|---|---|
| | 88 | 0 | 0 | 0 | 0 | 0 |
| | 0 | 71 | 0 | 0 | 0 | 0 |
| | 0 | 0 | 85 | 0 | 0 | 0 |
| | 0 | 0 | 0 | 64 | 0 | 0 |
| | 0 | 0 | 0 | 0 | 79 | 0 |
| | 0 | 0 | 0 | 0 | 0 | 113 |

obtenues par CEM croisé ($\mathbf{z}^{(h)}$,$\mathbf{w}^{(h)}$) et celles obtenues par deux $k$-means ($\mathbf{z}^{(k)}$,$\mathbf{w}^{(k)}$) (voir tableaux 6.7 et 6.8), on voit clairement notre méthode hybride fonctionne beaucoup mieux qu'un double $k$-means, qui fournit un grand nombre d'observations mal-classés.

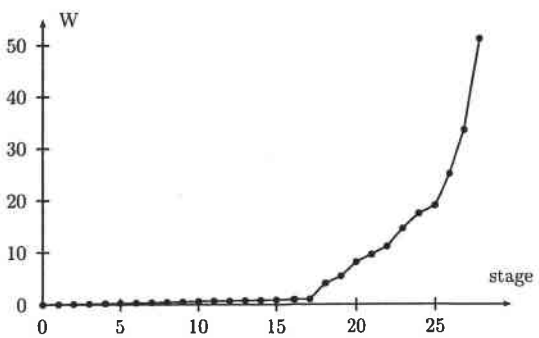



FIG. 6.11 – *Évolution du critère avec HBCM pour Set 2* ($s^* = 20$ *and* $m^* = 10$).

### Exemple de données binaires

Puisque nous avons développé les méthodes CEM croisé et *HBCM* pour ce type de données, nous allons voir ici une illustration du bon comportement de la combinaison de ces deux algorithmes. Nous avons simulé un tableau de données de taille $n = 1000$ et $d = 100$, avec cinq classes en lignes et trois classes en colonnes. Nous avons choisi

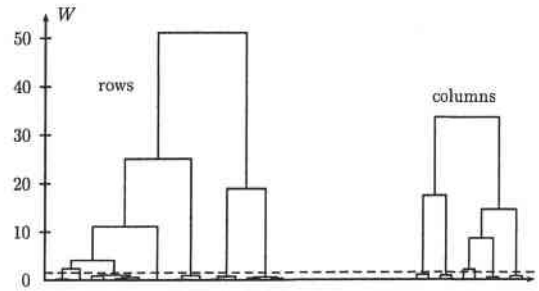



FIG. 6.12 – *Arbres hiérarchiques pour les lignes et les colonnes, à partir des centres de CEM croisé (Set 2).*

TAB. 6.7 – *Croisement des partitions en lignes simulées* ($\mathbf{z}^{(s)}$) *et obtenues avec CEM croisé* ($\mathbf{z}^{(h)}$) *et double k-means* ($\mathbf{z}^{(k)}$), *pour Set 2.*

| $\mathbf{z}^{(s)}$ \ $\mathbf{z}^{(h)}$ | | | | | | | |
|---|---|---|---|---|---|---|---|
| | 156 | 0 | 0 | 0 | 0 | 0 | 0 |
| | 0 | 144 | 0 | 0 | 0 | 0 | 0 |
| | 0 | 0 | 0 | 131 | 0 | 0 | 0 |
| | 0 | 0 | 0 | 0 | 131 | 0 | 0 |
| | 0 | 0 | 0 | 0 | 0 | 145 | 0 |
| | 0 | 0 | 0 | 0 | 0 | 0 | 153 |
| | 0 | 0 | 139 | 0 | 0 | 0 | 1 |

| $\mathbf{z}^{(s)}$ \ $\mathbf{z}^{(k)}$ | | | | | | | |
|---|---|---|---|---|---|---|---|
| | 0 | 93 | 0 | 63 | 0 | 0 | 0 |
| | 0 | 0 | 0 | 0 | 0 | 0 | 144 |
| | 18 | 0 | 0 | 0 | 0 | 0 | 113 |
| | 131 | 0 | 0 | 0 | 0 | 0 | 0 |
| | 0 | 0 | 145 | 0 | 0 | 0 | 0 |
| | 0 | 0 | 0 | 0 | 152 | 1 | 0 |
| | 0 | 0 | 0 | 0 | 0 | 140 | 0 |

TAB. 6.8 – *Croisement des partitions en colonnes simulées* ($\mathbf{w}^{(s)}$) *et obtenues avec CEM croisé* ($\mathbf{w}^{(h)}$) *et double k-means* ($\mathbf{w}^{(k)}$), *pour Set 2.*

| $\mathbf{w}^{(s)}$ \ $\mathbf{w}^{(h)}$ | | | | | | |
|---|---|---|---|---|---|---|
| | 0 | 0 | 0 | 0 | 80 | 0 |
| | 91 | 0 | 0 | 0 | 0 | 0 |
| | 0 | 0 | 0 | 0 | 0 | 90 |
| | 0 | 84 | 0 | 0 | 0 | 0 |
| | 0 | 0 | 0 | 67 | 0 | 0 |
| | 0 | 0 | 88 | 0 | 0 | 0 |

| $\mathbf{w}^{(s)}$ \ $\mathbf{w}^{(k)}$ | | | | | | |
|---|---|---|---|---|---|---|
| | 0 | 0 | 0 | 0 | 80 | 0 |
| | 91 | 0 | 0 | 0 | 0 | 0 |
| | 0 | 0 | 0 | 0 | 0 | 90 |
| | 0 | 84 | 0 | 0 | 0 | 0 |
| | 0 | 1 | 0 | 66 | 0 | 0 |
| | 0 | 0 | 88 | 0 | 0 | 0 |

de prendre les noyaux suivants :

| | | |
|---|---|---|
| 1 | 0 | 0 |
| 1 | 1 | 1 |
| 0 | 1 | 1 |
| 0 | 0 | 1 |
| 0 | 0 | 0 |

Le degré d'homogénéité, pour chaque bloc, varie entre 0.3 et 0.4. En premier lieu, nous cherchons à réduire les données en une matrice d'information de taille réduite à l'aide de CEM croisé. Pour ceci, nous avons choisi de prendre $s = 20$ et $m = 10$ pour l'initialiser. Sur ce résumé, nous avons appliqué *HBCM*. Les noyaux obtenus par CEM croisé, et les arbres hiérarchiques en lignes et en colonnes obtenus avec *HBCM* sont présenté dans la figure 6.13. Il apparaît clairement cette méthode est

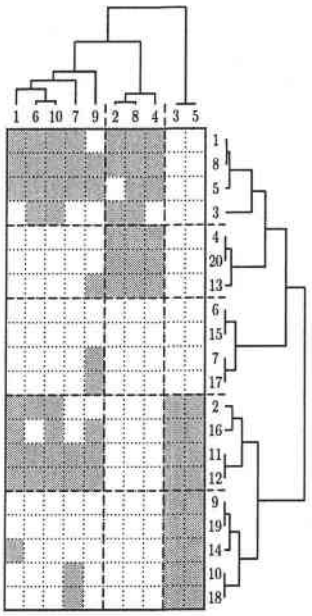



FIG. 6.13 – *Matrice d'information obtenue avec CEM croisé, avec les arbres hiérarchiques en lignes et en colonnes, obtenus avec HBCM, sur les données binaires simulées.*

TAB. 6.9 – *Croisement entre les partitions simulées et celles obtenues avec CEM croisé.*

| | | simulé | | | |
|---|---|---|---|---|---|
| CEM croisé | 177 | 3 | 1 | 0 | 25 |
| | 2 | 185 | 10 | 0 | 0 |
| | 0 | 7 | 168 | 12 | 0 |
| | 0 | 1 | 22 | 178 | 3 |
| | 2 | 1 | 0 | 0 | 203 |

| | | simulé | |
|---|---|---|---|
| CEM croisé | 35 | 0 | 0 |
| | 0 | 29 | 0 |
| | 0 | 0 | 36 |

capable de retrouver la structure simulée avec cinq classes en lignes et trois classes en colonnes. La figure 6.14 montre les données réorganisées selon la partition optimale obtenue en coupant les arbres hiérarchiques.

Afin de comparer les partitions simulées avec celles fournies par notre méthode, nous présentons dans la table 6.9 les croisements de celles-ci. Les valeurs en dehors de la diagonale correspondent aux nombres d'objets et de variables mal-classés (8.9 % en lignes et 8% en colonnes).

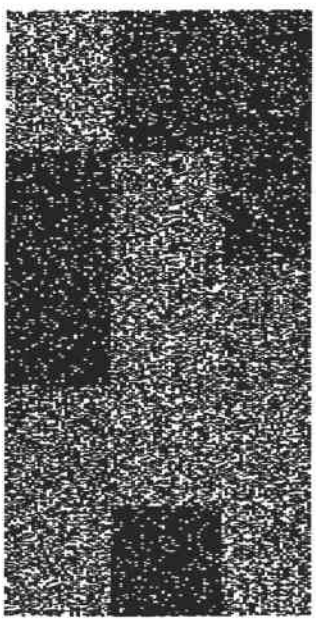


FIG. 6.14 – *Données initiales réorganisées selon les partitions obtenues avec HBCM.*

# Conclusion et perspectives

La classification est un outil très intéressant, dans un contexte de Fouille des Données, afin d'avoir un résumé informatif de la population étudiée. Malheureusement, nombre de méthodes proposées sont généralement basées sur une optimisation d'un critère métrique, avec certains défauts connus. Dernièrement, l'approche probabiliste, et essentiellement les modèles de mélange, permettent de justifier ces critères, et d'en proposer d'autres, plus génériques, et remédiant ainsi aux défauts de ces algorithmes. Nous nous sommes donc tout naturellement inscrit dans cette approche, en utilisant des algorithmes de type EM. Et comme les données réelles sont rarement complètes, nous avons aborder le problème des données manquantes sous cette approche.

Un des inconvénients majeurs d'une classification est la connaissance requise du nombre de classes. Bien évidemment, cette information est généralement peu disponible, principalement dans le cadre Fouille de Données, où le but est d'apprendre à partir des données. Ainsi, nous avons proposé ici deux méthodes distinctes de choix du nombre de classes. La première, très classique, réside dans l'utilisation de critères, afin de tester plusieurs nombres de classes $s$ candidats, et de ne retenir que celui qui optimise ces critères. Nous en avons comparé certains sur des données qualitatives, et les deux plus intéressants que nous avons retenu sont BIC et AIC3.

La seconde méthode consiste à tirer partie des deux types de méthodes de classification non supervisée, la classification simple et hiérarchique. En premier lieu, nous résumons l'information contenue dans la table en une matrice de taille réduite. Ensuite, nous recherchons un nombre de classes approprié en appliquant une méthode hiérarchique ascendante sur les classes précédemment obtenues. Une fois ce nombre $s$ choisi, on lance une dernière fois un algorithme de classification simple, pour obtenir une partition profitable de la population. Cette approche est connue depuis longtemps dans le cadre d'algorithmes basés sur des critères métriques, nous l'avons utilisée ici en exploitant les avantages du modèle de mélange. Les résultats obtenus sont bons, et confirment l'intérêt évident d'une telle méthode.

Comme nous avons vu que l'algorithme EM peut parfois converger lentement (plusieurs centaines d'itérations, voire milliers), nous avons présenté deux stratégies d'initialisation (avec CEM et eM), ainsi que plusieurs algorithmes permettant d'accélérer cette convergence (IEM, SpEM et LEM). A partir de l'un d'entre-eux, nous avons développé notre propre méthode LEM-D, qui s'avère plus performante que les autres. De plus, alors que cet algorithme nécessite un paramètrage (seuil et nombre d'itérations), nous proposons une méthodologie d'utilisation, qui permet de retrouver les mêmes coefficient d'accélération que LEM-D, et qui de plus, obtient

des paramètres et des partitions plus proches de ceux obtenus avec EM. Grâce à de telles techniques d'accélération, il sera donc possible de tester différents nombres de classes, en utilisant les différents critères de sélection.

Ensuite, et puisque la classification croisée apparaît maintenant comme un outil très prometteur dans une optique Fouille de Données, nous avons présenté trois algorithmes de partitionnement double, qui cherchent simultanément des classes en lignes et des classes en colonnes. Puis, nous nous sommes intéressé à deux algorithmes de classification directe, *One-way* et *Two-way splitting*, cherchant eux aussi une structure en blocs des données, avec une contrainte hiérarchique sur les deux ensembles. Ces deux dernières méthodes ne nécessitent pas de connaître le nombre de blocs à obtenir, et se basent sur un seuil pour leur critère d'arrêt.

Enfin, à la manière de notre démarche dans le cadre classification simple, nous avons présenté l'utilisation de modèles de mélange croisée, présentant ainsi CEM croisé, dans le but de donner une justification théorique aux critères optimisés par les algorithmes précédemment cités. Nous avons ainsi pu introduire notre méthode innovante de classification hiérarchique en blocs, *HBCM*, permettant de s'affranchir de la connaissance du nombre de classes en lignes et en colonnes. Avec ces méthodes ainsi présentées, nous avons étendu la classification mixte au cas croisé, en combinant d'une part l'algorithme CEM croisé avec *Two-way splitting* et en combinant d'autre part CEM croisé et *HBCM*. L'intérêt de ces combinaisons s'est montré flagrant, elles constituent donc des méthodes de recherches de partitions croisées très intéressantes.

À la suite de ce travail, plusieurs perspectives de travail nous semblent intéressantes. Puisque le choix du nombre de classes dans une population sera toujours un problème ouvert, il peut être intéressant de proposer une alternative aux deux solutions présentées ici. L'idée serait de partir avec un nombre de classes $s$ choisi par l'utilisateur, et de faire évoluer celui-ci, avec des règles de création, destruction, réunion ou division de classes. Cet algorithme serait dérivé de EM et proche de la version stochastique de celui-ci, SEM, qui ne fait que supprimer des classes. L'extension proposée permettrait ainsi de pouvoir tester un grand nombre de classes, en se basant sur la connaissance déjà acquise des données. De plus, l'apport stochastique permettrait de résoudre, partiellement, le problème de l'initialisation de l'algorithme.

Ici, nous avons présenté l'approche Classification, avec l'algorithme CEM croisé. Il serait donc intéressant de proposer la version Estimation de celui-ci, qui serait donc l'algorithme EM croisé. Après avoir défini celui-ci, il serait très intéressant de pouvoir développer les critères de choix du modèle et de choix du nombre de classes, tels que BIC et AIC3, dans le cadre de la classification croisée. De plus, il semblerait utile de proposer une gestion des données manquantes, d'une manière similaire à celle utilisée ici, pour la classification simple.

Enfin, les méthodes de classification directe présentées dans ce travail, comme l'algorithme *Two-way splitting*, reposent sur des critères métriques. Il nous semble

nécessaire de réfléchir sur la notion d'un modèle de mélange associé à de telles structures, pour :

- d'une part, apporter un éclairage sur l'algorithme utilisé (*Two-way splitting*), et de réaliser une extension de celui-ci pour aborder des situations plus complexes;
- d'autre part, résoudre le problème du choix du seuil, soit en proposant une solution appropriée, soit en supprimant celui-ci.

L'avantage d'une telle démarche serait aussi de pouvoir, à l'aide de simulations, tester et valider cette méthode, sur différents types de données (continues, qualitatives, binaires, ...).

Dans le cadre de ce travail, et afin de pouvoir comparer les algorithmes utilisés en terme de vitesse, nous avons donc du tous les écrire. Pour ceci, après avoir utilisé *C++* pendant quelques temps, nous nous sommes intéressé à un langage de type matriciel, nommé IML, avec le logiciel de statistiques *SAS System* (SAS Institute).

# Bibliographie

Akaike. Information Theory and an Extension of the Maximum Likelihood Principle. Dans Petrov, B. et Csaki, F., éditeurs, *Second International Symposium on Information Theory*, pages 267–281, 1973.

Arabie, P., Boorman, A., et Levitt, P. Construction Blockmodels: how and why. *Journal of Mathematical Psychology*, 17:21–63, 1978.

Ball, G. H. et Hall, D. J. ISODATA, an Iterative Method of Multivariate Analysis and Pattern Recognition. *Behavior Science*, 153, 1967.

Banfield, J. et Raftery, A. Model-based Gaussian and Non-Gaussian Clustering. *Biometrics*, 49:803–821, 1993.

Bencheikh, Y. *Classification automatique et modèles.* Thèse de Doctorat, Université de Metz, 1992.

Benzecri, J. P. *L'analyse des données : la taxinomie, Tome 1.* Dunod, Paris, 1973.

Bezdek, J. C. *Pattern Recognition with Fuzzy Objective Function Algorithms.* Plenum Press, New York, 1981.

Biernacki, C., Celeux, G., et Govaert, G. Strategies for getting likelihood in mixture models. Dans *JSM'2000*, Indianapolis, 2000.

Biernacki, C., Celeux, G., et Govaert, G. Assessing a Mixture Model for Clustering with the Integrated Classification Likelihood. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(7):719–725, 2001.

Bock, H. *Simultaneous Clustering of Objects and Variables*, pages 187–203. Le Chesnay INRIA, 1979.

Bozdogan, H. *Multi-Sample Cluster Analysis and Approaches to Validity Studies in Clustering Individuals.* Thèse de Doctorat, Department of Mathematics, University of Illinois, Chicago, 1981.

Bozdogan, H. Determining the Number of Component Clusters in the Standard Multivariate Normal Mixture Model Using Model-Selection Criteria. Rapport Technique UIC/DQM/A83-1, Quantitative Methods Department, University of Illinois, Chicago, 1983.

Bradley, P., Fayyad, U., et Reina, C. Scaling EM (Expectation-Maximization) Clustering to Large Databases. Rapport Technique MSR-TR-98-35, Microsoft Research, 1998.

Broniatowski, M., Celeux, G., et Diebolt, J. Reconnaissance de Mélanges de Densités par un Algorithme d'Apprentissage Probabiliste. Dans Diday *et al.*, éditeur, *Data Analysis and Informatics*, volume 3, pages 359–374, 1983.

Bzioui, M. *Classification croisée et modèle.* Thèse de Doctorat, Université Technologique de Compiègne, 1999.

Celeux, G. Classification et modèles. *Revue de statistique appliquée*, 4:43–58, 1988.

Celeux, G. Modèles probabilistes en classification. Dans Droesbeke, J., Fichet, B., et Tassi, P., éditeurs, *Modèles pour l'analyse de données multidimensionnelles*, pages 165–211. Economica, 1992.

Celeux, G., Diday, E., Govaert, G., Lechevallier, Y., et Ralambondrainy, H. *Classification Automatique des Données*. Dunod, 1989.

Celeux, G. et Diebolt, J. The SEM Algorithm: a Probabilistic Teacher Algorithm Derived from the EM Algorithm for the Mixture Problem. *Computational Statistics Quaterly*, 2:73–82, 1985.

Celeux, G. et Diebolt, J. A Stochastic Approximation Type EM Algorithm for the Mixture Problem. *Stochastics and Stochastics Reportes*, 41:119–134, 1992.

Celeux, G. et Govaert, G. Clustering Criteria for Discrete Data and Latent Class Models. *Journal of Classification*, 8:157–176, 1991.

Celeux, G. et Govaert, G. A Classification EM Algorithm for Clustering and two Stochastic Versions. *Computational Statistics & Data Analysis*, 14:315–332, 1992.

Celeux, G. et Govaert, G. Comparison of the Mixture and the Classification Maximum Likelihood in Cluster Analysis. *J. Statis. Comput. Simul.*, 47:127–146, 1993.

Celeux, G. et Govaert, G. Gaussian Parsimonious Clustering Models. *Pattern Recognition*, 28:781–793, 1995.

Cheeseman, P. et Stutz, J. Bayesian Classification (AutoClass): Theory and Results. Dans Fayyad, U., Pitesky-Shapiro, G., et Uthurusamy, R., éditeurs, *Advances in Knowledge Discovery and Data Mining*, pages 61–83. AAAI Press, 1996.

Chickering, D. et Heckerman, D. Efficient Approximations for the Marginal Likelihood of Bayesian Networks with Hidden Variables. *Machine Learning*, 29(2-3):181–212, 1997.

Day, N. E. Estimating the Components of a Mixture of Normal Distributions. *Biometrika*, 56:464–474, 1969.

Dempster, A. P., Laird, N. M., et Rubin, D. B. Maximum Likelihood for Incomplete Data via the EM Algorithm. *Journal of the Royal Statistical Society*, 39(B):1–38, 1977.

Diday, E. Une Nouvelle Méthode en Classification Automatique et Reconnaisance des Formes. *Revue de Statistique Appliquée*, 19(2), 1971.

Diday, E. Classification automatique séquentielle pour grands tableaux. *RAIRO Intellignece Artificielle et Reconnaissance des Formes*, 1975.

Diday, E., Schroeder, A., et Ok, Y. The Dynamic Clusters Method in Pattern Recognition. Dans *Proceedings of IFIP Congress*, 1974.

Diday, E. *et al.*. *Optimisation en Classification Automatique*. Le Chesnay, INRIA, 1980.

Duffy, D. et Quiroz, A. A Permutation-Based Algorithm for Block Clustering. *Journal of Classification*, 8:65–91, 1991.

Dunn, J. C. A Fuzzy Relative of the ISODATA Process and its Use in Detecting Compact Well-Separated Clusters. *J. Cybernet.*, 3:32–57, 1974.

Everitt, B. *An Introduction to Latent Variables Models.* Chapman and Hall, 1984.

Everitt, B. et Hand, B. *Finite Mixture Distributions.* Chapman and Hall, 1981.

Fischer, W. *Clustering and Aggregation in Economics.* The Johns Hopkins Press, Baltimore, 1969.

Forgy, E. W. Cluster Analysis of Multivariate Data : Efficiency versus Interpretability Models. *Biometrics*, 61(3):768–769, 1965.

Fraley, C. et Raftery, A. E. MCLUST: Software for Model-Based Cluster and Discriminant Analysis. Rapport Technique 342, University of Washington, 1999.

Friedman, H. et Rubin, J. On Some Invariant Criterion for Grouping Data. *Journal of the American Statistical Association*, 62, 1967.

Govaert, G. Algorithme de Classification d'un Tableau de Contingence. Dans *Premières journées Internationales Analyse des Données et Informatique*, pages 487–500. Le Chesnay INRIA, 1977.

Govaert, G. *Classification Croisée.* Thèse d'Etat, Université de Paris 6, 1983.

Govaert, G. Classification Binaires et Modèles. *Revue de Statistique Appliquée*, 37:67–81, 1990.

Govaert, G. Simultaneous Clustering of Rows and Columns. *Control and Cybernetics*, 24:437–458, 1995.

Govaert, G. et Nadif, M. Comparison of the Mixture and the Classification Maximum Likelihood in Cluster Analysis with binary data. *Comput. Statis. and Data Analysis*, 23:65–81, 1996.

Govaert, G. et Nadif, M. Block clustering on continuous data. Dans *Proc. of the Workshop on Clustering High Dimensional Data and its applications at the Second SIAM international Conference on Data Mining*, pages 7–16, 2002.

Govaert, G. et Nadif, M. Clustering with Block Mixture Models. *Pattern Recognition*, pages 463–473, 2003.

Guha, S., Rastogi, R., et Shim, K. CURE: an efficient clustering algorithm for large databases. Dans *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 73–84, 1998.

Gyllenberg, M., Koski, T., et Verllaan, M. Classification of Binary Vectors by Stochastic Complexity. Rapport Technique A5, University of Turku (Finland), Institute for Applied Mathematics, 1994.

Han, J. et Kamber, M. *Data Mining : Concepts and Techniques.* Morgan Kaufman, 2001.

Hartigan, J. Direct Clustering of a Data Matrix. *JASA*, 67:337, 123–129, 1972.

Hartigan, J. *Clustering Algorithms.* John Wilez & Sons, 1975.

Hartigan, J. Modal Blocks in Dentition if West Coast Mammals. *Systematic Zoology*, 25:149–160, 1976.

Hartigan, J. et Wong, M. A $K$-Means Clustering Algorithm. *J-APPL-STAT*, 28(1):100–108, 1979.

Hathaway. Another Interpretation of the EM Algorithm for Mixture Distribution. *Journal of Statistics & Probability Letters*, 4:155–176, 1986.

Huang, Z. A Fast Clustering Algorithm to Cluster very large categorical data sets in Data Mining. Dans *SIGMOD Workshop on Research Issues on Data Mining and Knowledge Discovery (SIGMOD-DMKD'97)*, 1997.

Hunt, L. *Clustering using finite mixtures models.* Thèse de Doctorat, University of Waikato, Hamilton, New-Zeland, 1996.

Jennings, A. A sparse matrix shema for the computer analysis of. *International JOURNAL of Computer Methematics*, 2:1–21, 1968.

Jollois, F.-X. et Nadif, M. Clustering large categorical data. Dans *Advances in Knowledge and Data Mining: 6th Pacific Asia Conference: Proceedings / PAKDD 2002, 6-8 Mai, Taipei, Taiwan*, pages 257–263, 2002.

Jollois, F.-X., Nadif, M., et Govaert, G. Assessing the number of clusters of the latent class model. Dans *Classification, Clustering, and Data Analysis: Recent Advances and Applications*, pages 139–146. Springer, 2002.

Jollois, F.-X., Nadif, M., et Govaert, G. Classification croisée sur des données binaires de grande taille. *Extraction et Gestion des Connaissances, RSTI série RIA-ECA*, 17(1-2-3):213–218, 2003.

Karypis, G., Eui-Hong, H., et Kumar, V. Chameleon: Hierarchical Clustering Using Dynamic Modeling. *Computer*, 32(8):68–75, 1999.

Lance, G. et Williams, W. A general theory of classification sorting strategies, I: Hierarchical systems. *The Computer Journal*, 9:373–380, 1967.

Lazarfeld, P. F. *A conceptuel introduction to latent structure analysis*, Chapitre Mathematical Thinking in the Social Sciences, chapter 7. Free Press, New-York, 1954.

Lazarfeld, P. F. et Henry, N. W. *Latent Structure Analysis.* Houghton Mifflin, Boston, 1968.

Little, R. et Rubin, D. *Statistical Analysis with Missing Data.* John Wiley, New-York, 1987.

MacQueen, J. Some Methods for Classification and Analysis of Multivariate Observations. Dans *Proc. 5th Berkeley Symposium on Mathematical Statistics and Probability*, pages 281–297, 1967.

Marchetti, F. *Contribution à la classification de données binaires et qualitatives.* Thèse de Doctorat, Université de Metz, 1989.

Marchotorchino, F. Block seriation problems, a unified approach. *Applied Stochastic Models and Data Analysis*, 3:73–91, 1987.

McCallum, A., Nigam, K., et Ungar, L. H. Efficient Clustering of High-Dimensional Data Sets with Application to Reference Matching. Dans Ramakrishnan, R. et Stolfo, S., éditeurs, *Proceedings of the 6th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 169–178, 2000.

McLachlan, G. et Basford, K. *Mixture Models, Inference and Applications to Clustering.* Marcel Dekker, New-York, 1989.

McLachlan, G. et Krishnan, T. *The EM Algorithm and Extensions.* John wiley & sons, Inc., New-York, 1997.

McLachlan, G. J. et Peel, D. User's guide to EMMIX-Version 1.0. Rapport Technique, University of Queensland, 1998.

Moore, A. Very Fast EM-Based Mixture Model Clustering Using Multiresolution kd-Trees. Dans Kearns, M. S., Solla, S. A., et Cohn, D. A., éditeurs, *Advances in Neural Information Processing Systems 11: Proceedings of the 1998 Conference*, pages 543–549, 1999.

Nadif, M. *Classification automatique et données manquantes.* Thèse de Doctorat, Université de Metz, 1991.

Nadif, M. et Govaert, G. Binary clustering with missing data. *Applied Stochastic Models and Data Analysis*, 13:269–278, 1993.

Nadif, M., Govaert, G., et Jollois, F.-X. A Hybrid system for identifying homogenous blocks in large data sets. Dans *Second Euro-Japanese Workshop on Stochastic Risk Modelling for Finance, Insurance, Production and Reliability, 16-19 septembre, Chamonix, France*, pages 324–333, 2002.

Nadif, M. et Marchetti, F. Classification de Données Qualitatives et Modèles. *Revue de Statistique Appliquée*, XLI(1):55–69, 1993.

Neal, R. et Hinton, G. A View of the EM Algorithm that Justifies Incremental, Sparse, and Other Variants. Dans Jordan, M., éditeur, *Learning in Graphical Models*, pages 355–371, 1998.

Nowlan, S. J. *Soft Competitive Adaptation: Neural Network Learning Algorithms Based on Fitting Statistical Mixtures.* Thèse de Doctorat, School of Computer Science, Carnegie Mellon University, Pittsburgh, 1991.

Pearson, K. Contribution to the Mathematic Theory of Evolution. *Philo. Trans. Soc.*, 185, 1894.

Rae, D. J. M. MICKA, a FORTRAN IV iterative k-means Cluster Analysis Program. *Behavourial Science*, 16:423–424, 1971.

Ralambondrainy, N. Etudes des données qualitatives par les méthodes typologiques. Dans *Actes au congrès de l'Association Française de Marketting, Montpellier*, 1988.

Redner, R. et Walker, H. Mixture densities, maximum likelihood and the EM algorithm. *SIAM Review*, 26(2):195–239, 1984.

Robardet, C. *Contribution à la classification supervisée : proposition d'une méthode de bi-partitionnement.* Thèse de Doctorat, Université Claude Bernard, Lyon 1, 2002.

SAS Institute. `http://www.sas.com`.

Sato, M. Fast Learning of On-Line EM Algorithm. Rapport Technique, ATR Human Information Processing Research Laboratories, 1999.

Sato, M. et Ishii, S. On-Line EM Algorithm for the Normalized Gaussian Network. *Neural Computation*, 12(2):407–432, 2001.

Schafer, J. *Analysis of Incomplete Multivariate Data.* Chapman and Hall, London, 1997.

Schroeder, A. Analyse d'un Mélange de Distributions de Probabilités de Même Type. *Revue de Statistique Appliquée*, 24(1):39–62, 1976.

Scott, A. J. et Symons, M. J. Clustering Methods Based on Likelihood Ratio Criteria. *Biometrics*, 27:387–397, 1971.

Symons, M. J. Clustering Criteria by Multivariate Normal Mixtures. *Biometrics*, 37:35–43, 1981.

Tanner, M. A. Tools for Statistical Inference. *Lecture Notes in Statistics*, 67, 1991.

Tewarson, R. *Sorting and ordering sparse linear systems.* Academic Press, New-York, 1971.

Thiesson, B., Meek, C., et Heckerman, D. Accelerating EM for Large Databases. Rapport Technique MSR-TR-99-31, Microsoft Research, 2001.

Tibshirani, R., Hastie, T., Eisen, M., Ross, D., Botstein, D., et Brown, P. Clustering methods for the analysis of DNA microarray data. Rapport Technique, Stanford University, Department of Statistics, 1999.

Wei, G. C. G. et Tanner, M. A. A Monte Carlo Implementation of the EM Algorithm and the Poor Man's Data Augmentation Algorithms. *Journal of the American Statistical Association*, 85:699–704, 1990.

Wolfe, J. H. Pattern Clustering by Multivariate Mixture Analysis. *Multivar. Behavior. Res.*, 5:329–350, 1970.

Wong, M. A. A Hybrid Clustering Method for Identifying High Density Clusters. *Journal of American Statistical Association*, 77:841–847, 1977.

Zhang, T., Ramakrishnan, R., et Livny, M. BIRCH: an efficient data clustering method for very large databases. Dans *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, pages 103–114, 1996.

**Résumé.** La classification est une étape essentielle dans un processus de Fouille de Données. Les méthodes usuelles que nous décrivons se basent sur des critères métriques. Mais, l'utilisation des modèles de mélange dans la classification est devenue une approche classique et puissante. En traitant la classification sous cette approche, à partir des algorithmes de type EM, nous nous sommes intéressés aux problèmes suivants : recherche du nombre de classes, gestion des données manquantes, stratégies d'initialisation et accélérations de l'algorithme.

L'algorithme EM est devenu quasiment incontournable dans l'approche mélange. Alors que beaucoup de travaux sont effectués sur des données continues, nous avons choisi de nous intéresser principalement aux données binaires et qualitatives. En premier lieu, nous avons proposé une étude comparative de critères de choix du nombre de classes les plus utilisés. Aussi, nous avons développé une méthode hybride s'affranchissant de ce choix en combinant un algorithme de type EM et un autre de type hiérarchique.

Pour la gestion des grandes masses de données, nous avons d'abord étudié plusieurs stratégies d'initialisation et d'accélération de cet algorithme, puis nous avons proposé une nouvelle variante, très efficace. Enfin, nous nous sommes intéressés à la classification croisée et directe. Nous avons rappelé les méthodes classiques qui s'avèrent des cas particuliers de méthodes de type EM issus d'un modèle de mélange croisé. A partir de ce modèle, nous avons proposé une nouvelle méthode de classification croisée hiérarchique, innovante et performante. Celle-ci, combinée aux méthodes de type EM, permet de traiter des grandes bases de données.

**Abstract.** Clustering is an essential step in Data Mining. The classical methods described here are based on metric criteria. But, the use of mixture model in clustering is now a classical and powerful approach. Treating clustering under this approach, with EM type algorithms, we are interested in following problems : search of number of clusters, management of missing data, initialization strategies and speed-up methods.

The EM algorithm is now a popular and efficient method for clustering with mixture models. Even though these methods are often focused on continuous data, we are mainly interested in binary and categorical data. First, we have proposed a comparative study of used criteria for the choice of the number of clusters. Then, we have developed an hybrid method, which avoids this choice, combining an EM type algorithm and a hierarchical algorithm.

To manage large database, we have first studied initialization strategies and speed-up methods of the EM algorithm. And we have proposed a novel variant, very efficient. Then, we are interested in block and direct clustering. We have described classical methods, which are particular case of EM type methods, from a block mixture model. From this model, we have proposed a new hierarchical block clustering method, innovative and efficient. Combined with EM type methods, this algorithm permits to manage large database.

**Résumé.** La classification est une étape essentielle dans un processus de Fouille de Données. Les méthodes usuelles que nous décrivons se basent sur des critères métriques. Mais, l'utilisation des modèles de mélange dans la classification est devenue une approche classique et puissante. En traitant la classification sous cette approche, à partir des algorithmes de type EM, nous nous sommes intéressés aux problèmes suivants : recherche du nombre de classes, gestion des données manquantes, stratégies d'initialisation et accélérations de l'algorithme.

L'algorithme EM est devenu quasiment incontournable dans l'approche mélange. Alors que beaucoup de travaux sont effectués sur des données continues, nous avons choisi de nous intéresser principalement aux données binaires et qualitatives. En premier lieu, nous avons proposé une étude comparative de critères de choix du nombre de classes les plus utilisés. Aussi, nous avons développé une méthode hybride s'affranchissant de ce choix en combinant un algorithme de type EM et un autre de type hiérarchique.

Pour la gestion des grandes masses de données, nous avons d'abord étudié plusieurs stratégies d'initialisation et d'accélération de cet algorithme, puis nous avons proposé une nouvelle variante, très efficace. Enfin, nous nous sommes intéressés à la classification croisée et directe. Nous avons rappelé les méthodes classiques qui s'avèrent des cas particuliers de méthodes de type EM issus d'un modèle de mélange croisé. A partir de ce modèle, nous avons proposé une nouvelle méthode de classification croisée hiérarchique, innovante et performante. Celle-ci, combinée aux méthodes de type EM, permet de traiter des grandes bases de données.

**Abstract.** Clustering is an essential step in Data Mining. The classical methods described here are based on metric criteria. But, the use of mixture model in clustering is now a classical and powerful approach. Treating clustering under this approach, with EM type algorithms, we are interested in following problems : search of number of clusters, management of missing data, initialization strategies and speed-up methods.

The EM algorithm is now a popular and efficient method for clustering with mixture models. Even though these methods are often focused on continuous data, we are mainly interested in binary and categorical data. First, we have proposed a comparative study of used criteria for the choice of the number of clusters. Then, we have developed an hybrid method, which avoids this choice, combining an EM type algorithm and a hierarchical algorithm.

To manage large database, we have first studied initialization strategies and speed-up methods of the EM algorithm. And we have proposed a novel variant, very efficient. Then, we are interested in block and direct clustering. We have described classical methods, which are particular case of EM type methods, from a block mixture model. From this model, we have proposed a new hierarchical block clustering method, innovative and efficient. Combined with EM type methods, this algorithm permits to manage large database.